BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF CIVIL ENGINEERING
FAKULTA STAVEBNÍ

INSTITUTE OF STRUCTURAL MECHANICS
ÚSTAV STAVEBNÍ MECHANIKY

THE EXPLOITATION OF PARALLELIZATION TO NUMERICAL SOLUTIONS REGARDING PROBLEMS IN NONLINEAR DYNAMICS

VYUŽITÍ PARALELIZACE PŘI NUMERICKÉM ŘEŠENÍ ÚLOH NELINEÁRNÍ DYNAMIKY

DOCTORAL THESIS
DIZERTAČNÍ PRÁCE

AUTHOR — **Ing. Václav Rek**
AUTOR PRÁCE

SUPERVISOR — **doc. Ing. Ivan Němec, CSc.**
VEDOUCÍ PRÁCE

BRNO 2018

Abstract
The main aim of this thesis is the exploration of the potential use of the parallelism of numerical computations in the field of nonlinear dynamics. In the last decade the dramatic onset of multicore and multi-processor systems in combination with the possibilities which now provide modern computer networks has risen. The complexity and size of the investigated models are constantly increasing due to the high computational complexity of computational tasks in dynamics and statics of structures, mainly because of the nonlinear character of the solved models. Any possibility to speed up such calculation procedures is more than desirable. This is a relatively new branch of science, therefore specific algorithms and parallel implementation are still in the stage of research and development which is attributed to the latest advances in computer hardware, which is growing rapidly. More questions are raised on how best to utilize the available computing power. The proposed parallel model is based on the explicit form of the finite element method, which naturaly provides the possibility of efficient parallelization. The possibilities of multicore processors, as well as parallel hybrid model combining both the possibilities of multicore processors, and the form of the parallelism on a computer network are investigated. The designed approaches are then examined in addressing of the numerical analysis regarding contact/impact phenomena of shell structures.

Abstrakt
Hlavním cílem této práce je prozkoumání možností využití paralelizace v numerických výpočtech nelineární dynamiky. V poslední dekádě došlo k dramatickému nástupu vícejádrových a výceprocesorových systému v kombinaci s možnostmi, které nyní poskytují moderní počítačové sítě. Komplexnost a velikost řešených modelů se neustále zvyšuje a díky vysoké výpočetní náročnosti úloh dynamiky a statiky konstrukcí, a to především kvůli jejich často nelineárnímu charakteru, je jakákoliv možnost urychlení výpočetních procedur více než žádoucí. Jelikož se jedná o relativně nové odvětví, řada algoritmů a konkrétních paralelních implementací je stále ve stádiu vývoje a výzkumu, a to i proto, že pokroky v oblasti počítačového hardwaru rapidně vzrůstají a s tím vznikají další otázky, jak nejlépe využít dostupný výpočetní výkon. Navržený paralelní model je založený na explicitní formě metodě konečných prvků, která ze své podstaty poskytuje možnost efektivní paralelizace. Zkoumány jsou pak možnosti využití vícejádrovch procesorů, ale i hybridního paralelního modelu kombinujícího možnosti vícejádrových procesorů a paralelní formy na počítačové síti. Navržené přístupy jsou pak testovány při numerickém řešení kontaktní/impaktní úlohy skořepinových konstrukcí.

Keywords
FEM, Explicit Form of FEM, FDM, Dynamics of Structures, Parallel Computing, GPGPU, NVIDIA CUDA, Computer Network, TCP/IP, .NET, C/C++, C#

Klíčová slova
MKP, explicitní forma MKP, MKD, dynamika konstrukcí, paralelní výpočty, GPGPU, NVIDIA CUDA, počítačové sítě, TCP/IP, .NET, C/C++, C#

REK, Václav. *The Exploitation of Parallelization to Numerical Solutions Regarding Problems in Nonlinear Dynamics.* Brno, 2018. 223 p. Doctoral thesis. Brno University of Technology. Faculty of Civil Engineering. Supervisor Ivan NĚMEC.

I declare that I have written my doctoral thesis on the theme of "*The Exploitation of Parallelization to Numerical Solutions Regarding Problems in Nonlinear Dynamics*" independently, under the guidance of the supervisor and using the technical literature and other sources of information which are all quoted in the thesis.

Brno ....
(author's signature)

First of all, I would like to express my gratitude to my dissertation thesis supervisor, Assoc. Prof. Ivan Němec. He has been a constant source of encouragement and insight during my research and helped me with numerous problems and professional advancements.
It is an honor for me that I could become a part of the famous Brno school of the finite element method, whose founders are great persons, headed by Prof. Vladimír Kolář, Prof. Jiří Kratochvíl, Prof. Miloš Zlámal, Prof. Alexander Ženíšek and others.
I would also like to acknowledge academician Prof. Jiří Spurný from the Faculty of Mathematics and Physics of the Charles University and Prof. Jiří Vala from the Faculty of Civil Engineering of the Brno University of Technology for their valuable advice on the theory of nonlinear functional analysis, finite element method and also on related topics of mathematical physics. I would also like to acknowledge software architects Mgr. Ján Naď and to my colleague Ing. Martin Neborák for their valuable advice in the field of IT technologies for the composition of enterprise network systems, GPGPU technology and theoretical computer science, to my friend Dr. Pavel Gruber for his valuable advice in the field of mechanics of materials and sometimes endless discussions on general issues of the mechanics.
I would also like to express my gratitude to CADTeam s.r.o. company, especially to its owner, theoretical physicist Dr. Petr Lorenc for providing his private local area computer network for the testing purposes of the of hybrid-parallel FEM solver, and also to my colleague, system engineer Josef Dvořák for his help related to the computer network configuration.
This work would never have originated without the support of my beloved family, notably the support of my mother Ivana, my wife Kateřina and my daughter Leontýnka, and also without the support of all my dear friends.

Ing. Václav Rek

Brno University of Technology
Faculty of Civil Engineering
Institute of Structural Mechanics

The Exploitation of Parallelization to Numerical Solutions Regarding Problems in Nonlinear Dynamics

by

Václav Rek

A doctoral thesis submitted to
the Faculty of Civil Engineering, Brno University of Technology,
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

Doctoral degree study programme: Structures and Traffic Constructions

Brno, July 2018

Supervisor:
Ivan Němec
Institute of Structural Mechanics
Faculty of Civil Engineering
Brno University of Technology
Veveří 331/95
602 00 Brno
Czech Republic

Dedication

With an honor to the memory of Professor Vladimír Kolář for inspiring my curiosity.

Contents

Introduction

Thanks to rapid advances during the last decade in computer technology, in the field of multicore and multiprocessor technology, a lot of attention has been devoted to the parallel processing of data in many scientific and industrial sectors. This is in great contrast to previous decades when an increase in computing power, especially in personal computers, was achieved by increasing the clock speed of processors and has resulted in increase in energy consumption and thus energy inefficiency. This type of technological evolution has its limitations compared to multi-processors and multi-core computer architectures respectively. Multi-processor platforms are the oldest way how to distribute time intensive computations.

A technology which is completely separate from the previously mentioned technologies is the technology of quantum computers which promises to solve problems that are intractable on digital computers.

Now it is quite common to use a CPU (Central Processing Unit) for the parallel run of computational time-consuming tasks in synchronous and asynchronous order. Parallel programming is especially supported by the software libraries OpenMP (Open Multi-Processing, the oldest multi-platform shared memory multiprocessing programming based on compiler's special directives, see https://msdn.microsoft.com/en-us//library/tt15eb9t.aspx),
MPI (Message Passing Interface, developed and maintained by a consortium of academic, research, and industry partners, see http://www.mpi-forum.org/, [36]), API (Application Programming Interface) of UNIX-like operating systems (POSIX - Portable Operating System Interface, see http://standards.ieee.org/develop/wg/POSIX.html), Microsoft Windows (Windows API, see https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx, [42]) operating systems or the other OS. Nowadays it is also possible to use native threads in C++ standard libraries (since its version 11, see https://isocpp.org/wiki/faq/cpp11#cpp11-what). The aforementioned technlogies are focused mainly on usage of C and C++ programming language. In some form they are also accessible even for the historically most widely used scientific programming language Fortran (see http://www.tutorialspoint.com/fortran/). Especially this programming

language is in the interest of analyses in connection with the implementation of new technologies due to a number of scientific code, occurring in analytical programs, especially in the commercial sector.

Many other compiled or interpreted programming languages just like C# (see https://msdn.microsoft.com/en-us/library/618ayhy6.aspx), Java (see https://www.java.com/en/about/), Python (see https://www.python.org/) or Matlab (see http://www.mathworks.com/products/matlab/) etc. contain libraries which support in some form the parallel run of computing tasks. These libraries, especially MPI, are very well known to interested researchers for the parallel run of a variety of computations in computational mechanics, particularly for the domain decomposition method.

Since the year 2007, when the CUDA (Compute Unified Device Architecture, see http://www.nvidia.com/object/cuda_home_new.html) was introduced by NVIDIA company, it has led to significant increases in the possibilities for the utilization of high-performance multi-core graphical chips not only for computer visualization purposes for 2D and 3D computer graphics. CUDA and related technologies like OpenCL (Open Computing Language, free standard for cross-platform, parallel programming of modern processors, see https://www.khronos.org/opencl/, [33]) and Microsoft DirectCompute (supports general-purpose computing on graphics processing units on Microsoft's operating systems since Windows Vista OS, see https://channel9.msdn.com/Tags/directcompute-lecture-series) are currently being investigated by many researchers. Their applications occur in research involved mainly in information technologies, in algorithms for artificial intelligence (diagnosis and synthesis of voice, image recognition, etc.).

Numerical methods which were used in the past due to their inefficiency at the mere periphery of scientific interest, due to the current technological advances have become more attractive. This involves mainly explicit numerical methods used in computational mechanics. Especially it concerns the explicit method used in dynamics, statics of structures and fluid dynamics respectively.

Explicit algoritms are highly suitable for a solution of short time highly non-linear computations mainly for numerical simulation of the processes of forming casts or the simulation of crash tests in the automotive and aviation industry or for shape finding of thin membranes in civil engineering etc. This method facilitates the consideration of a variety of nonlinearities in an easy and explicit manner. Based on the core of explicit dynamic numerical methods is also the dynamic relaxation numerical method which is used for static numerical analysis of civil and mechanical structures. The conditional stability character of explicit methods leads, in some cases, to the necessity to use an exceptionaly small integration step. As a consequence, explicit methods are time consuming. The availability of a huge number of parallel threads directly implemented in hardware enabled their effective usage (see [104]).

Aims of the Thesis

The view is focused on both the various types of hardware and computer networks, resectively. Today's development tools then enable their effective combination and usage. This concerns primarily to the parallelization on graphic processors due to the GPGPU technology and multicore CPUs. Computer networks then allow another level of parallelization of already parallel models, which are applied on the local machine. This is particularly relevant to the issue of so-called the Big Data.

The field of challenging numerical computations of a crash test simulations in the automotive and aerospace industries seems to be a relevant application area. The highly geometrically non-linear behavior of the shell structures of car and aircraft bodies in the respective dynamic processes requires the most effective way of dealing, and parallelism here is more than desirable.

The rapid availability of results from numerical simulations allows more efficient design of the relevant constructions. In the case of transport constructions it is primarily for the purpose of simulation to estimate the indices of impact severity under the conditions of EN 1317 and also other than those. European Norm EN 1317 defines common testing and certification procedures for a road restraint systems. Thus the aims of the thesis can be summarized into two points, as follows:

(i) The main benefit should be to explore the possibilities of computations of parallelism for the use in computations of statics and dynamics of structures within the framework of an explicit numerical methods applied in scope of the Finite Element Analysis (FEA). Based on an analysis of the current hardware and software opportunities, design a parallel approach to enable their efficient utilization.

(ii) Perform numerical simulations of nonlinear dynamics using the explicit form of FEM for appropriate structures with the help of designed software solution.

Chapter 1

State of the Art

This section briefly deals with the history and advances of science in the field of dynamics, as well as the field of scientific computing.

In the field of dynamics, it deals with its origins dating back to Ancient Greece, up to the modern approaches for solving problems of continuum and quantum mechanics. Considered to be a much younger field of science, computer science on began its growth in recent decades. In this conjunction important persons and inventions behind such developments, especially in the field of computational mechanics are discussed. Finally, it deals with an area of parallelization in scientific computing, which in light of recent developments appears to be the focus of an intensive research.

1.1 Brief History Review of Dynamics

The general study of the dynamics date to the time of Ancient Greece, it can possibly be dated to an even later period of the first ancient Mesopotamian civilization of Sumerians and civilization of Egyptians.

In the case of Ancient Greece, it can be dated from the work of Thales of Miletus (∗624 B.D., †548 B.D.), who is considered as the founder of Greek philosophy. Since philosophy in the early days contained all sciences, both the natural as well as social science, it is considered an important period as Aristotle (∗384 B.D., †322 B.D.) and Archimedes (∗287 B.D., †212 B.D.) put forward intriguing ideas at the time.

Aristotle, in his thinking about motion, stemmed on a faulty concept of classical elements (fire, air, water and earth). In context of advances in his era it is understandable. By these elements he tried to describe the processes which occur naturally. In brief it is good to mention eg. processes where air an bubble breathed underwater floats to the surface or a rock thrown upwards falls back to the Earth. Aristotle's conclusions are described in [109] as follows:

> *Aristotle concluded that heavier objects fall faster than light objects, and that this fall–rate is proportional to their weights: an object twice as heavy falls*

> *twice as fast. He also reasoned that the speed of progression through a medium was inversely proportional to the density of that medium. This reasoning implied that the speed of progression in the void would be in-finite; thus he concluded that the very existence of a void was impossible. In the same section, he wrote that if a void were to exist, heavy objects would fall at the same rate as light ones ("Therefore all will possess equal velocities. But this is impossible."). He used this supposed equality of fall rates to then say by modus tollens that a void cannot exist. He further wrote that, in a void, there would be no reason for a body to stay in one place or move to another, and so motion would continue forever. It is often said, based on this statement, that he enunciated or foresaw a principle of inertia, but this is only possible by a selective reading of his works.*

Other important ideas described by Aristotle are described in his work *Physics* (see [3]).

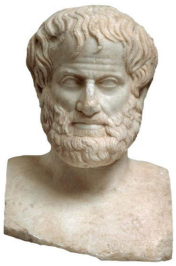


Figure 11: Probable look of Aristotle.

Another important philosopher later on was Alexandrian philosopher, John Philoponus (∗490, †570), who dealt critically with Aristotle's work.

The subsequent period of the early Middle Ages after the fall of the Western Roman Empire in the year 476 consisted of both science and society in Europe's Dark Age. Science was grown rather in the Middle East and in Asia. With respect to this era, we should mention mainly Arabic-Andalusian philosopher Ibn Bajjah and Ibn Rushd (∗1095, †1138), who is known as Avempace. He dealt with Aristotle's ideas.

Science began to develop again during the Renaissance era, which meant a return to the ideals of the ancient world and a deviation from an ecclesiastical dogmas in terms of general view of human life and knowledge, when free thinking was labeled as heresy. A fresh wind blowing into society was caused mainly by Church reformers like eg. John Wycliffe (∗1330, †1384) in the Kingdom of England, Jan Hus (∗1370, †1415) in the Czech Kingdom or Martin Luther (∗1483, †1546) in the Holy Roman Empire of the German Nation.

Certain departures from the Aristotelian dogma at the end of the 13th century meant a harbinger of changes to the view of the kinematics. This then led to further define the mean-speed theorem, which is attributed to the English philosopher William of Heytesbury (∗1313, †1373). An important achievement in this field then was the work of French philosopher Jean Buridan (∗1300, †1358), which deeply engaged Aristotle's writings and laid down the basic concept of the idea of inertia. His view on the motion would be similar to Newton's first law of motion in today's view.

The personality, which today we can call the founder of modern dynamics is the Italian philosopher and physicist Galileo Galilei (∗1564, †1642). Galileo extended principles from static into dynamic concepts, where he took behaviour of bodies and their natural positions of rest, described by Archimedes, and applied it to bodies in motion. Although his greatest contribution is in the field of kinematics, his work in dynamics influenced many of his successors.

Very important personalities in the field of mechanics at that time was a Dutch physicist and mathematician Christiaan Huygens (∗1629, †1695), who first explained the oscillation of the final pendulum. In this context discreet systems and their interactions were also dealt with.

The most prominent person in dynamics, and we could say, even of classical physics, as the mastermind of the fundamental laws of motion is the English physicist and mathematician Isaac Newton (∗1643, †1728). It is necessary to mention the publication *Philosophæ Naturalis Principia Mathematica* in the year 1687, where he published his thoughts regarding the fundamental laws of motion. The whole is divided into three books, namely the first two have the title *Of the Motion of Bodies*, and the third is *The System of the World*. He explained that the laws that are applied to the motion of the planets in universe are generally valid even in conditions of the processes on Earth. His conclusions had a fundamental influence and have become a milestone in physics in general.

Newton also developed the calculus of mathematics, and the "changes" expressed in the second law are most accurately defined in differential forms. Calculus can also be used to determine the velocity and location variations experienced by an object subjected to an external force.

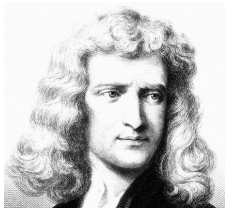


Figure 12: Sir Isaac Newton.

It should be mentioned that the case of angular momentum, as one of the cornerstones of mechanics, Newton did not explain. This is a natural consequence of linear momentum. Newton used the third law to derive the law of the conservation of momentum. From a deeper perspective, the conservation of momentum is the more fundamental idea derived via theorem derived by German mathematician Emmy Noether's (∗1882, †1935) from Galilean invariance, and holds in cases where Newton's third law appears to fail, eg. when force fields as well as particles carry momentum, and in quantum mechanics.

German mathematician Gottfried Wilhelm Leibniz (∗1646, †1716), who was a contemporary of Newton, proposed another quantity the *vis viva* (living force). It is a theory which served as an elementary and limited early formulation of the principle of conservation of energy. His living force has the dimension of energy and is scalar in nature. In this

approach, referred to as *analytical mechanics*, the laws of mechanics are expressed in terms of work done and energy expended. Very significant contributions to the further development of mechanics laid mathematicians and physicists namely Swiss Leonhard Paul Euler (∗1707, †1783) and Frenchmen Joseph Louis Lagrange (∗1736, †1813) and Jean Le Rond d'Alembert (∗1717, †1783). In later time the Irish physicist William Rowan Hamilton (∗1805, †1865) expressed the laws of mechanics using the variational statement. Newton's and Hamilton's statements are equivalent, they differ in the sense that while the formers are in the form of cause and effect relationships at each instant of time, the latters is in the form of extremum conditions of a functional over an arbitrary period of time.

This concerned mainly solving the problems of flexible bodies such as vibrating string or catenary curve. Newton tried to deal with these problems, but apparatus that was available to him was not sufficient. Further contributions to the development of mechanics were made by German mathematician Carl Gustav Jacob Jacobi (∗1804, †1851) and in more recent times by Albert Einstein (∗1879, †1955), who, in his Theory of Relativity (or Theory of Invariance), brought the concepts of length, time and simultaneity of events under critic review (see [124]).

Since the primary concern of this work is the field of mechanics of continua, so it is also needed to mention some important milestones that are related to this theme. In order to describe continuum it is therefore necessary to use instruments of mathematical analysis, mainly differential and integral calculus. This analysis has however, been focused for a long period on functions of only one variable which in a general consideration of three-dimensional space together with the time, was no longer adequate. As a result, the concept of partial derivatives was introduced. This we essentially owe to the Bernoulli's—John (∗1667, †1748) and Daniel (∗1700, †1782), John's son—and Jean Le Rond d'Alembert.

Jean Le Rond d'Alembert formulated the first equation of wave motion—a second-order partial differential equation of the so-called hyperbolic type (finite velocity of propagation)—with its paradigmatic solution. It should be mentioned that this work is in its basis strongly tied with the works of Leonhard Paul Euler, Joseph Louis Lagrange and French mathematician Augustin Louis Cauchy (∗1789, †1857). The aforementioned scientists then were able to formulate the standard theory of perfect fluids and perfectly elastic solids, two cases in which ideal descriptions cope with what we now call nondissipative behaviours.

One of the most essential results of this period was the formulation of *variational principles*. At the beginning there was a person of French mathematician Pierre de Fermat (∗1601, †1665), who studied law of reflection and the refraction of light. It was previously formulated by the Dutch mathematician Willebrord Snellius (∗1580, †1626) based on experimental observations in 1621. Fermat used rigorous mathematical methods for analysis by minimizing the path of the light compared to Snellius's experimental approach. In this context it is necessary to mention the fact that the use of the calculus of variation in mechanics has its basis in a solution of problems of dynamics. The first person to use it was a French mathematician Pierre Louis Moreau de Maupertui (∗1698, †1759) in 1744 when he enunciated the principle of *least action* for an analysis of the collision of elastic balls. It was later published in his *Essai de cosmologie* in 1750.

Later it were Euler and Lagrange, with the work on variational calulus culminating in

Lagrange's book *Méchanique analitiqu* (published in 1788). It provided an essential tool in the general field theory developed by William Rowan Hamilton and others, and also it introduced the necessary physical basis of methods such as the FEM for solving partial differential equations in Sobolev spaces (a special class of Hilbert spaces), usually used in engineering applications of fluid dynamics and with macroscopic structural problems. The "imperfect" cases developed mainly in the second half of the 20th century, which are related to the theory of *thermodynamically irreversible* behaviours (fluid viscosity, visco-elasticity of solids, plasticity of solids, etc.) cannot, in principle, be deduced from a variational formulation in the manner of Lagrange and Hamilton. Euler and Lagrange are also considered responsible for the kinematic descriptions of continua called Eulerian and Lagrangian.

Then the next important result was the introduction of the notion of *stress tensor* by Cauchy in his first theory of continua (1822, published in 1828).

Figure 13: Leonhard P. Euler, Joseph L. Lagrange, Jean L. R. d'Alembert, Augustin L. Cauchy and William R. Hamilton.

It is also necessary to mention advances of *nonlinear dynamics* in continuum that relate to the late 19th and early 20th century. These include, primarily, the German mathematician Georg Bernhard Riemann (∗1826, †1866), Scottish mechanical engineer William John Macquorn Rankine (∗1820, †1872), French mathematician and physicist Pierre Henri Hugoniot (∗1815, †1887), and French engineer Jacques Charles Émile Jouguet (∗1871, †1943). They proved the existence and propagation of shock and detonation waves. French physicist Pierre Duhem (∗1861, †1916) also had a significant influence mainly dealing with the wave propagation in nonlinear elasticity.

As already mentioned earlier, dynamics, but also physics in general underwent a radical change in view on the fundamental laws that until then relied heavily on Newtonian physics. A new perspective on the laws of physics, which came with Albert Einstein in his *Theory of Relativity* published in two forms, namely, *Special Theory of Relativity* (published in 1905) and *General Theory of Relativity* (published in 1915, focused on the origin and description of gravity) meant a change of view also with respect to mechanics continua in general. Albert Einstein later realized (letter to E. Zschimmer, 30 September 1921), that the name of his first work was unfortunate and suggested the new name the *Invariantheorie*. He failed to do so due to the strong rooting of the original name among physicists and also the general public.

The mathematical description of spacetime significantly helped German-Polish mathematician and Enstein's professor of mathematical analysis Hermann Minkowski (∗1864, †1909) with the introduction of the four-dimensional version of special relativity and that of energy-moment tensor, to which must be added the fact that general relativity is per se

a continuum theory so there was need for a true relativistic theory of the continuum. This area relates mainly to particle physics and the fundamentals laws of thermodynamics. Due to this modern trend it led to a reassessment of classical continuum mechanics, i.e. primarily in the axiomatic field and the construct of relativistic elasticity, and its generalization to more complex thermomechanical schemes.

With Enstein's theory as a revolutionary idea also linked to it is the so-called *Quantum Theory*. For the "father" of this theory can be regarded the German physicist Max Karl Ernst Ludwig Planck (∗1858, †1947), who was the first to introduce it into the notation of physics and quantum of energy. Today, this theory provides a deep insight into the stucture of atoms and atomic nuclei as well as that of bodies of sizes familiar to our everyday experience. This revolutionary theory is not still incomplete, particularly with regard to interconnection with the Theory of Relativity and the problem of elementary particles being stalled by tremendous difficulties encountered on the way toward further development. In this context it is also necessary to mention the next pioneers of quantum theory, namely, the Danish physicist Niels Henrik David Bohr (∗1885, †1962), who extended Planck's idea of the quantization of radiant energy to the description of mechanical energy of electrons within an atom. He introduced the specific "quantization rules" for the mechanical systems of atomic sizes and achieved a logical interpretation of the New Zealand physicist Ernest Rutherford's (∗1871, †1937) planetary model of an atom, which rested on a solid basis but on the other hand stood in sharp contradiction to all the fundamental concepts of classical physics. As another scientist it is important to mention the Austrian physicist Erwin Schrödinger (∗1887, †1961), who extended the ideas of French physicist Louis Victor Pierre Raymond de Broglie (∗1892, †1987) about the driven motion of electrons called *pilot waves* into a more exact mathematical form, the so-called *Schrödinger's Equations*. His theory became known as *Wave Mechanics*. In fact, wave mechanics provided a complete and perfectly self-consistent theory of all atomic phenomena, and could also explain the phenomena of radiactive decay and artifical nuclear transformations. Schrödinger's wave mechanics explained all the atomic phenomena for which Bohr's theory failed, and in addition predicted some new phenomena which had not even been dreamed of, either in classical physics or in Planck-Bohr quantum theory. We should also mention the German physicist Werner Karl Heisenberg (∗1901, †1976) and his very important *Principle of Uncertainty* (published in 1927).

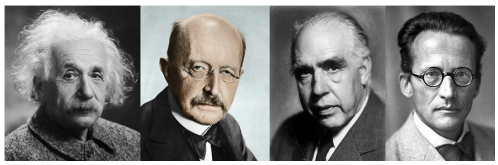


Figure 14: Albert Einstein, Max K. E. L. Planck, Niels H. D. Bohr, Erwin Schrödinger.

These modern theories affect not only the view of dynamics in its general form, but also computer science, especially considering the revolutionary technology of quantum

computers as a next level of thinking about parallelism, which is considered further. Form more see [82] and [32].

1.2 Brief History Review of Computer Science

It is difficult to precisely date the efforts of people who tried to automate various tasks occuring in human activities.

Among the first predecessors of modern computer techniques belongs French mathematician and physicist Blaise Pascal (∗1623, †1662) with his mechanical calculator, which was constructed in the 17th century (1642) and later Charles Babbage (∗1791, †1871). He was *Lucasian Chair of Mathematics* at University of Cambridge with Isaac Newton being before him.

Probably the first known attempt to organize information processing on a large scale using human computers was the production of mathematical tables, such as logarithmic and trigonometric tables. Logarithmic tables revolutionized mathematical computation in the 16th and 17th centuries by enabling time-consuming arithmetic operations, such as multiplication and division and the extraction of roots, to be performed using only the simple operations of addition and subtraction.

During the 19th century, the English mathematician and engineer Charles Babbage dealt with these issues. Babbage came up with the idea to build a programmable computer machine. He composed a computer machine named a Difference Engine, because it used the same method of differences that Gaspard de Prony (∗1755, †1839, interested in the task of producing logarithmic and trigonometric tables for the French Cadastre) and others used in table making.

Figure 15: Charles Babbage.

The difficulties Charles Babbage encountered during the development are as follows (see [69]):

> *Unfortunately, the engineering was more complicated than the conceptualization. Babbage completely underestimated the financial and technical resources he would need to build his engine. While building the Difference Engine in the 1820s was not in any sense impossible, Babbage was paying the price of being a first mover; it was rather like building the first computers in the mid-1940s:*

difficult and extremely expensive. In the London Science Museum today, one can see evidence of this complexity in hundreds of Babbage's machine drawings for the engines and in thousands of pages of his notebooks.

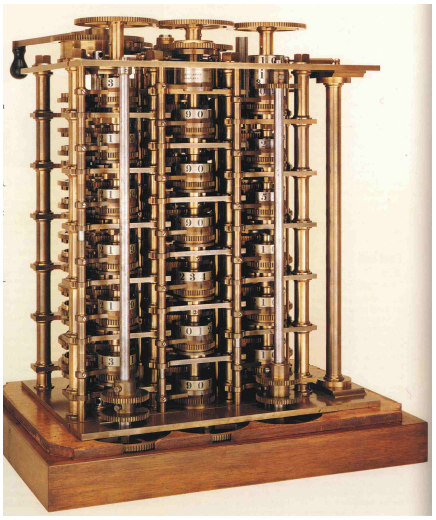


Figure 16: Babbage's Difference Engine (Niagara College Canada).

Due to the huge population expansion in the United States of America (USA) during the second half of the 19th century, the problem of how to evaluate large amounts of report tables arose. One person who was acutely aware of the census problem was a remarkable young engineer by the name of Herman Hollerith (∗1859, †1929). He later developed a mechanical system for census data processing, commercialized his invention by establishing the Tabulating Machine Company in 1896, and laid the foundations of the IBM (abbreviation of the International Business Machines corporation) company.

A breakthrough period in this branch was the period during the Second World War and the subsequent epoch that followed it. World War II was a scientific war: its outcome was determined largely by the effective deployment of scientific research and technical developments. The best known wartime scientific programs were the Manhattan Project at Los Alamos to develop the atomic bomb, and the program of developing radar mainly at the Radiation Laboratory at MIT (abbreviation of the Massachusetts Institute of Technology).

Emphasis on these major programs overshadowed the rich tapestry of the scientific war effort. One of the threads running through this tapestry was the need for mathematical computation. For the atomic bomb, for example, massive computations had to be performed to perfect the explosive lens that assembled a critical mass of plutonium. At the outbreak of war, the only computing technologies available were analogue machines such as differential analyzers, primitive digital technologies such as punched-card installations, and teams of human computers equipped with desktop calculating machines. Even relatively slow one-of-a-kind mechanical computers such as the Harvard Mark I lay some years in the future.

A milestone work in this area became the labor of English mathematician Alan Turing (∗1912, †1954) and John von Neumann (∗1903, †1957). Alan Turing attempted to decipher the German code of Enigma and on the basis of his theories an electrical computer called Colossus was created. John von Neumann was the youngest member of the Institute for Advanced Study in Princeton, where he was a colleague of Albert Einstein and other eminent mathematicians and physicists. Unlike most of the other people involved in the early development of computers, von Neumann had already established a world-class scientific reputation (for providing the mathematical foundations of quantum mechanics and other mathematical studies) and strongly influenced the work of Alan Turing, whose work in theoretical developments of mathematical logic later had a major impact on the progress of computer science as an academic discipline.

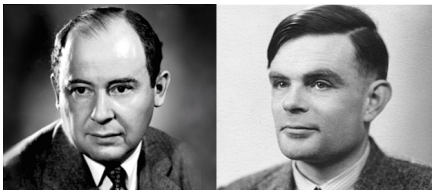


Figure 17: John von Neumann and Alan Turing.

In this period the first electronic computer ENIAC (abbreviation of the Electronic Numerical Integrator And Computer) was invented, developed in the period 1943 - 1946 for general computing based on Turing's theory. It was later used in the development of thermonuclear bomb (respective period of the level and state of the development in computer technology can now be likened to recent developments, which relates to the technology of quantum computers).

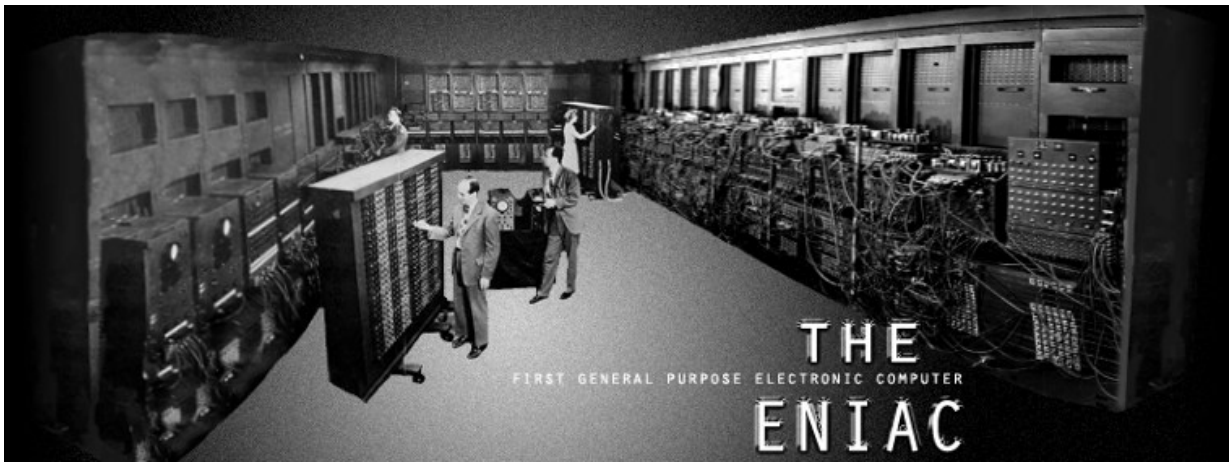
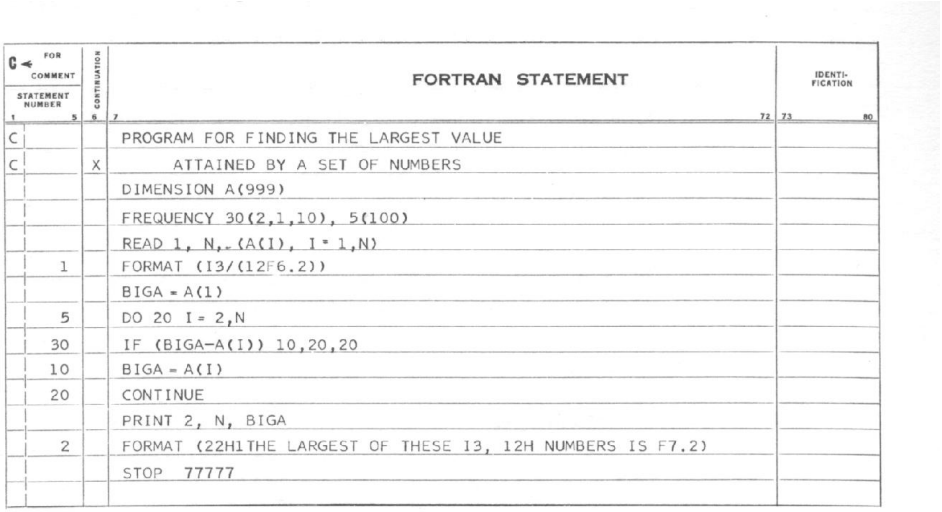


Figure 18: The ENIAC computer.

This powerful new technology of electronic computers strongly influenced the defeat of Nazi Germany, shortened the hardships of the World War II, laid down the foundations

of today's digital computers and also began to realize a great boom in computer science as well as the newly established science of cybernetics, whose founder was the American mathematician Norbert Wiener (∗1894, †1964). This work was especially groundbreaking as it laid down the foundations of Cybernetics [130].

In this new field, IBM began to be the most active, primarily in the USA, and invented the first high-level programming language, FORTRAN (abbreviation of the FORmula TRANslator), which was released in 1957 [49]. Until that time, machine or assembly language were used for computer programming.



C ← FOR COMMENT	STATEMENT NUMBER 1–5	CONTINUATION 6	FORTRAN STATEMENT 7–72	IDENTIFICATION 73–80
C			PROGRAM FOR FINDING THE LARGEST VALUE	
C		X	ATTAINED BY A SET OF NUMBERS	
			DIMENSION A(999)	
			FREQUENCY 30(2,1,10), 5(100)	
			READ 1, N, (A(I), I = 1,N)	
	1		FORMAT (I3/(12F6.2))	
			BIGA = A(1)	
	5		DO 20 I = 2,N	
	30		IF (BIGA-A(I)) 10,20,20	
	10		BIGA = A(I)	
	20		CONTINUE	
			PRINT 2, N, BIGA	
	2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)	
			STOP 77777	

Figure 19: FORTRAN code for punch cards.

The next epoch of the development of operating systems [69] is described as follows:

> *By the end of the 1960s a new set of concerns had emerged about software and software development, to the point that many industry observers were talking openly about a looming "software crisis" that threatened the future of the entire computer industry. For the next several decades, the language of the "software crisis" would shape many of the key developments—technological, economic, and managerial—within electronic computing. One obvious explanation for the burgeoning software crisis was that the power and size of computers were growing much faster than the capability of software designers to exploit them. If the 1960s was the decade of the software debacle, the biggest debacle of all was IBM's operating system OS/360. The "operating system" was the raft of supporting software that all the mainframe manufacturers were supplying with their computers by the late 1950s. It included all the software, apart from programming languages, that the programmer needed to develop applications programs and run them on the computer. One component of the operating system, for example, was the input/output subroutines, which enabled the user to organize files of data on magnetic tapes and disk drives. The programming of these*

peripheral devices at the primitive physical level was very complex—involving programmer-years of effort— but the operating system enabled the user to deal not with physical items, such as the specific location of bits of data on a magnetic disk, but with logical data, such as a file of employee records. Another operating system component was known as a "monitor" or "supervisor," which organized the flow of work through the computer.

In context, the development of operating systems had begun, especially with the advent of personal computers, which were the consequence of the progress in the technology of semiconductors and miniaturization. Until the 70's, it was customary to write operating systems in assembly language, mainly due to having maximum control over the operating system and also due to the inability of programming languages at that time to support the programming of the hardware equipment at the lowest level. Due to this fact, operating systems were unduly linked with the underlying hardware for which they were written to operate on.

Then, a major event was the introduction of the C programming language (published in 1978, [60]), whose authors were Brian W. Kernighan (∗1942) and Dennis M. Ritchie (∗1941, †2011) from Bell's laboratories. They also deserve to rewrite the UNIX (originally project Unics, abbreviation the Unary Information and Computing Service) operating system, which was originally released in the year 1969. It subsequently became the first modern portable operating system written using high-level programming language (C programming language for system programming) compared to the commonly used assembly programming language.

Figure 110: Brian W. Kernighan and Dennis M. Ritchie.

It was followed by the DOS (abbreviation of the Disk Operating System) operating system and later MS-DOS, of the newly emerging company Microsoft, which had became the base of the Windows OS in the 90s. It later became the most used operating system on personal computers worldwide.

In the 70's and subsequent years many software companies were created, which led to the significant development of computer networks. Access to the UNIX operating system's source codes has provided significant information relating to the design principles of operating systems to a broad scientific community. On the basis of the UNIX operating system, a Finnish student of computer science, Linus Torvalds (∗1969), had composed the kernel of the operating system Linux (abbreviation of the Linus Is Not UniX). It has become a freely available operating system. Since the turn of the millennium it has become the most widely used operating system worldwide, mainly due to mobile devices of various types.

Today the core of the Linux operating system is presented in many versions (Android OS, Ubuntu, Red Hat and Fedora, etc.).

Figure 111: Creator of Linux OS kernel Linus Torvalds.

1.2.1 Scientific Computing in Computational Mechanics

During the 60's of the 20th century a variety of programs began to appear, especially in the military field for the US space program and applications, which were focused on a solution to a number of tasks for stress analysis in the mechanics of structures and the dynamics of fluids, as well as automatic control. The American National Aeronautics and Space Administration (NASA) began with the development of the now well-known commercial program NASTRAN (abbreviation of the NAsa STRucture ANalysis, developed by the Computer Sciences Corporation) for its space program, based on FEM, which at that time began to experience a great boom just because of the development of computer technology. The period of the 60's of the 20th century can be considered as one of the most important periods in modern society. Czechoslovakia in this development played a significant role, mainly due to the people working around the Brno University of Technology.

At this time, research institutes in the USA and Europe began to persue the development of FEM and its computer realization. In this regard, the global development centers of FEM belonged the Swansea University (Prof. Olgierd C. Zienkiewicz, ∗1921, †2009) in the United Kingdom, the University of Stuttgart (Prof. John H. Argyris, ∗1913, †2004) in Germany and Brno University of Technology (Prof. Vladimír Kolář (∗1928, †2000), Prof. Miloš Zlámal (∗1924, †1997), Assoc. Prof. Ivan Němec (∗1945) and Prof. Jiří Kratochvíl (∗1929)) in Czechoslovakia. Later, other prominent scientists joined, mainly researchers from the University of California, Berkeley (Prof. Robert L. Taylor) and MIT (Prof. Klaus-Jürgen Bathe, ∗1943), both from the USA.

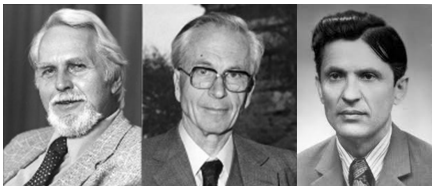


Figure 112: Prof. Olgierd C. Zienkiewicz, Prof. John H. Argyris and Prof. Miloš Zlámal.

As a pioneer in the development of the world's first FEM programs was Prof. Edward L. Wilson (∗1931) from University of California, Berkeley. His first programs had no name,

later they were transformed into the system under the name SAP (abbreviation of the Structural Analysis Program) and NONSAP (abbreviation of the NONlinear Structural Analysis Program). As the first commercial FEM software for solving nonlinear problems can be considered program MARC developed by the Marc Analysis Research Corporation named after its creator Pedro V. Marcal from Brown University.

With regard to the explicit computational form of the FEM, it is necessary to remember the program SADCAT, which was developed in the early 1970s at Argonne as one of the first three-dimensional structural dynamics programs which employed explicit time integration to achieve great efficiency for the computational simulation of impulsively loaded shells. This program closely relates to the work of Prof. Ted Bohdan Belytschko (∗1943, †2014) from the Northwestern University, who focused on the development of effective finite elements for transient nonlinear dynamics. It is important to note his effective C^0 triangular shell finite element, which is used in numerical simulations in this thesis (see [136]).

Figure 113: Professor Ted Bohdan Belytschko.

For completeness, it is necessary to mention the well-known programs based on FEM, which at that time and some years later began to emerge. It is a group of programs e.g. ABAQUS, ANSYS (founded by John A. Swanson), ADINA see http://www.adina.com/index.shtml (founded by Prof. Klaus-Jürgen Bathe), and especially software DYTRAN for explicit dynamics and fluid structure interaction (see http://www.mscsoftware.com/product/dytran).

From a national perspective, it is necessary to acknowledge the work of Prof. Vladimír Kolář and Assoc. Prof. Ivan Němec, who began with the development of software tools for a solution to civil and mechanical structures by FEM.

The NE-XX system (NE means initial letters of the author's surname Němec, see http://www.fem.cz/historie/?lang=en) became a widespread computational analytical tool based on FEM, both in Europe and now around the world. The NE-XX system is now the numerical computational core of the commercial program RFEM (see https://www.dlubal.com/en/rfem-5xx.aspx) of the German company Dlubal Software and the Scia Engineer (see http://www.scia.net/en/software/product-selection/scia-engineer) of the Belgian company Nemetschek Scia which occupy an important position in the European marketplace with this type of software.

Professor Vladimír Kolář belongs among the pioneers of FEM. He was persecuted and removed from academic life during the totalitarian communist regime after the Soviet

invasion in the year 1968. The publication of his books and articles were prohibited.

Despite all the hardships he had educated many of their successors through the secret lectures. He founded the famous Brno school of FEM which has gone on to become top class. In the time of freedom, his name, legacy and glory live on again, without persecution of the totalitarian regime. A team of academicians from the Brno University of Technology, which was built around his person, significantly contributed to the theoretical and a practical development of FEM in a global context.

Figure 114: Professor Vladimír Kolář in 1976.

In connection with Prof. Vladimír Kolář acknowledgement also need to be given to the work of Prof. Ivo Babuška (∗1926) from the University of Texas, Austin and Prof. Jindřich Nečas (∗1929, †2002) from the Faculty of Mathematics and Physics of Charles University.

Prof. Ivo Babuška is still active today and his most recent major achievement is laying down the foundations of the XFEM method (shortcut of the eXtended Finite Element Method, see [61]) primarily used for a numerical solution of problems containing discontinuous fields of primary variables (see [85]) applied primarily to the field of Fracture Mechanics.

Prof. Jindřich Nečas then significantly contributed in the field of partial differential equations (see mainly [90]) and nonlinear functional analysis (see [45]) in a global context, i.e. in the fields strongly related to the development of FEM.

It is also necessary to mention the development of software tools at the Czech Technical University in Prague (CTU in Prague) in the Department of Mechanics, Faculty of Civil Engineering. These are primarily general analytical software tools OOFEM (abbreviation of the Object Oriented Finite Element solver, see http://www.oofem.org, [98]) developed by Prof. Bořek Patzák and SIFEL (abbreviation of the SImple Finite ELements, see http://mech.fsv.cvut.cz/~sifel/) developed by Prof. Jaroslav Kruis. Especially, it is necessary to mention the work of Prof. Jaroslav Kruis focused on the parallelization using the FETI based method (abbreviation of the Finite Element Tearing and Interconnecting, [63]). Also worth mentioning is the project MuPIF (abbreviation of the Multi-Physics Integration Framework) for implementation of multi-physics and multi-level simulations assembled from independently developed applications components (see [99]).

Among the international open source projects it is necessary to mention the project Kratos (see [68], [53] atc., http://www.cimne.com/kratos/), which is a framework for building multi-disciplinary finite element programs. It is free multi-physic FEM C++ open

source code. Kratos is parallelized for Shared Memory Machines (SMMs) and Distributed Memory Machines (DMMs).

Kratos is being developed within the framework of the international project CIMNE (The International Center for Numerical Methods in Engineering, research organization created in 1987 at the heart of the prestigious Technical University of Catalonia (UPC) and a Partnership Between the Government of Catalonia and the UPC; see www.cimne.com). The aim of CIMNE is the development of numerical methods and computational techniques for advancing knowledge and technology in engineering and applied sciences.

From the point of view of the utilization of the GPGPU for numerical computations, it is necessary to mention the open-source finite element toolkit NiftySim (see https://sourceforge.net/projects/niftysim/). The toolkit is founded on the total Lagrangian explicit dynamics (TLEDs) algorithm. Total Lagrangian explicit dynamics is an efficient and accurate approach for simulation of soft tissues in biomedical applications (see [56]) as well as for the contact/impact simulations of various structures. Another open-source finite element toolkit based on explicit dynamics is project Impact (see http://www.impact-fem.org/).

1.2.2 Multiprocessor and Multicore Technologies in Scientific Programming

At first, it is necessary to mention supercomputers based on the multiprocessor architecture, and currently even in combination with the multicore processor technology. These include supercomputer ILLIAC IV as a first massively parallel computer (University of Illinois, [16]); The UNIVAC division of Sperry Rand Corporation delivered the first multiprocessor 1108 containing up to 3 CPUs and EXEC 8 OS supporting multithread program execution for respective hardware [38]; PEPE (shortcut of the Parallel Element Processing Ensemble) developed for the ballistic missile defense environment [22]; The Connection Machine [44] as a series of supercomputers that grew out of Danny Hillis's doctoral research at MIT in the early 1980s; BBN Butterfly built by Bolt, Beranek and Newman [74] in the late 1980s; Evans and Sutherland who came up with the world's first general supercomputer machine (ES-1) on market in 1989 [117]; Intel iPSC (Intel Personal SuperComputer) [25]; etc.

Figure 115: The world's fastest and most powerful digital supercomputer Tianhe - 2.

Currently, the most powerful computers belong to Titan-Cray XK7 (built by Cray at Oak Ridge National Laboratory in the USA), which is designed for various physical simulations (modeling global atmospheric phenomena, nuclear physics, etc.) and the most powerful supercomputer Tianhe-2 (MilkyWay-2) in China.

On the other side stands the technology of quantum computers [67] in comparison to the already mentioned computer technology. Respective technology is built principally on a different basis than the current digital computers. The first such commercially used quantum computer is D-Wave of the world's first quantum computing company D-Wave Systems in Canada. Their systems are being used by world-class organizations and institutions including Lockheed-Martin (first customer), Google, NASA, and USRA (abbreviation of the Universities Space Research Association).

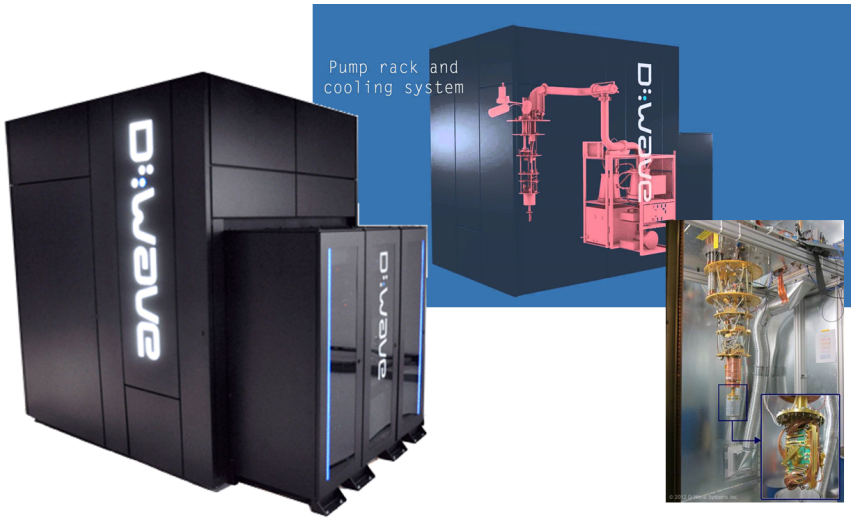


Figure 116: Quantum computer D-Wave.

The original idea came in the year 1982 from American physicist Richard Feynmann (∗1918, †1988). He proposed using a quantum system to simulate the other one. The reason was the difficulty of simulations of quantum systems with standard digital computers. Simulation algorithms of quantum systems have huge memory requirements (storing information about the quantum states of particles) and non-polynomial computational complexity of the quantum system's time evolution. Quantum computing is absolutely a breakthrough technology that is capable of performing amazing computations, unthinkable with current digital computers. The computer D-Wave is currently not a device for a general-purpose application as is the case of supercomputers Tianhe-2 and the others. Due to the high complexity of such devices, they are designed to solve specific optimization problems where they effectively use the main idea of quantum theory.

The effectivity of quantum computers lies in the concept of q-bits and quantum parallelism. It concerns effects of the so-called *quantum interconnection* and *quantum tunneling* [57]. Despite all the difficulties, it seems that this trend in hardware development is the right one, especially due to the incredible speed doing while specific computations. It is

therefore expectable that further development will go in such a direction. It is especially about the development of new algorithms suitable for this type of hardware.

In contrast with the above, the most advanced technologies for parallel data processing are personal computers (PC) that had been built for many years on a single core CPU and single processor architecture. Thus a large part of the development of the FE tools had been primarily concentrated on the sequential code execution. Multiprocessor systems have existed for decades, but until recently they were mostly found only in supercomputers and in large server systems. Multicore desktop computers, and even multicore embedded devices, are now increasingly prevalent. Until the advent of multicore processors in the first decade after the year 2000, it was not possible to use the real parallelism on common PCs.

The only one way to make an application run parallel was by using a so-called pseudo parallelism. However it does not cause an increase in the rate of data processing. Such a type of parallelism (it is concurrency, more precisly) is used to improve the program responsiveness, especially in programs with graphical UI. It then stays active even during the concurently running time-consuming background tasks. This applies to accessing the databases, data in networks and all other purposes where it is necessary to manage time-consuming processes. These days things have changed and programmers must take heed, and those who have hitherto ignored the theme of concurently running code within the programs, must add such programming skills into their toolbox. Herb Sutter (prominent C++ expert, who served for a decade as chair of the ISO C++ standards committee) aptly captured the situation by his quote *"The free lunch is over!"*.

The first attempt to parallelize finite element computations was the project Finite Element Machine at NASA in the late 70s and 80s which contained 32 16-bit processors [122]. Despite the mentioned attempt, the vast majority of codes were written for the sequential running of computing tasks. Then a big breakthrough was the advent of multicore processors in the first decade of the new millennium. In the previous years, many algorithms had begun to be developed which allowed parallel processing of computational tasks. These are primarily domain-decomposition methods, which mainly relate to graph partitioning of the finite element mesh (see below [59], [11], [126]).

An important numerical method related to a domain decomposition, it is the FETI method for the numerical solution of a large linear systems arising in linearized engineering problems, which in the early 90s developed C. Farhat and F.X. Roux [29]. The FETI method was originally proposed as a dual discrete nonoverlapping domain decomposition method for the parallel finite element solution of static equilibrium equations. Later arose its various modifications as TFETI (abbreviation of the Total FETI; see [23]) or HTFETI (abbreviation of the Hybrid Total FETI; see [110]). These are the computations performed solely on the CPU (MPI, OpenMP, etc.). Later, the FETI method was used for the other purposes than only for a parallel solution of a finite element models. These are primarily the work of scientists from CTU in Prague, namely the teams around Prof. Jaroslav Kruis and Prof. Zdeněk Bittnar (see [63], [64], [37]).

With the advent of technologies supporting general computation on graphic cards (GP-GPU), the development of algorithms has began for image processing (see [51], [100]),

algorithms for numerical mathematics and linear algebra (see [76], [96], [88] [2], [19]). Another field of application are numerical solvers for partial differential equations (see [26], [128], [119]). A high degree of importance is put on software applications for CFD, where the demanding requirements for the sparsity of the finite volume of finite element mesh and thus the high requirements for numerical processing of the problem (see [127], [75], [55], [111], [101]). Another field of application relates to the particle dynamics in many branches of physics (see [78], [73], [107]). Last but not least, inventions in the field of a code transfer between various software technologies and their combinations, is needed to be cited (see [24], [35]).

1.3 Summary of Chapter

In this chapter the view was focused to the issues concerning both the pillars of dynamics and computer science, respectively. A considerable part of this chapter is devoted to prominent scientists from the field of mathematical physics, whose work is often intertwined with each other or upon each other they directly depend. Interweaving the different areas of physics enabled so marked development in the 20th century. A reminder of some historical causalities then provides insight into the often complex beginnings.

It is hard to imagine how the world would look today if Euler, Lagrange, Cauchy, and the others lost their lifes in the difficult revolutionary years, from which came most of their work. Mainly with consideration of the fact they themselves were often politically engaged. The same also applies to the first half of the 20th century. The era suffered under the weight of two cruel world wars, especially under the second one. It relates primarily to scientists of Jewish origin, of whom a significant number perished in Nazi concentration camps.

Today's technological advances still accelerate and it puts pressure on the education of a broad and deep scope. Things become more complex and often further progress based on a creative approach to find interconnections which were previously almost unimaginable. Such an example is the scientific field belonging to quantum computers and their possible future usage in everyday life, i.e. the generalization of its functionality, development of algorithms, etc. Associated with such possibilities is the development of a more sophisticated models of thermodynamics and material models, which due to their current computational cost remain with its usage, rather in academia.

Personally, I hope such a comprehensive survey correctly creates the impression of complexity, scope, advances and future direction of the special interdisciplinary science represented by the name Computational Mechanics. It should also provide the proper introduction to further reading.

Chapter 2

Dynamics of Structures in the Language of Continuum Mechanics

Material or matter is composed of discrete molecules, which in turn are made up of atoms. An atom consists of negatively charged electrons, positively charged protons, and neutrons. Electrons form chemical bonds. The study of matter at molecular or atomistic levels is very useful for understanding a variety of phenomena, but studies of these scales are not useful to solve common engineering problems. Continuum mechanics is concerned with the study of various forms of matter at the macroscopic level. It studies motion of a medium that consists of matter subjected to forces. Traditionally, continuum mechanics has been divided into two groups: solids and fluids (liquids and gases), but the fundamental equations of continuum mechanics are the same for both of these. For many decades, solid and fluid mechanics have been treated independently from each other.

Central to this study is the assumption that the discrete nature of matter can be overlooked, provided the length scales of interest are large compared to the length scales of discrete molecular structures. Thus, matter at sufficiently large length scales can be treated as a continuum, which means a differentiable manifold with a boundary. Inherent in this assumption is that material particles that are neighbors will remain neighbors during the motion.

In the continuum representation of a physical system we can assume that although the material bodies through which fluids percolate, heat conduction takes place, chemicals diffuse, waves propagate, etc., it possess an affine structure. Quantities of interest such as density, temperature, concentration, displacements, internal forces, etc., can be defined in relation to the abstract mathematical concept of a point within the medium. The spatial and time derivatives are continuously differentiable almost everywhere and they are assumed to exist up to any required order. Thus a systematic study of the mathematical and computational properties of a given class of problems leads to a formal treatment that is more abstract then the treatment of a specific problem.

Nowadays it is necessary to simulate more complex materials, that have characteristics of solids and fluids simultaneously. These materials obeying the constitutive law for solids,

and also exhibit characteristics of fluids due to their viscosity. Next branch of continuum mechanics has emerged, which is related to multiphysics problems, characterized by phase change.

Continuum mechanics, based on certain principles, attempts to formulate the equations that govern given physical problems by means of partial differential equations. To these it is necessary to add the boundary and initial conditions in order to guarantee the uniqueness of the problem. This set of partial differential equations and the boundary initial conditions make up the Initial Boundary Value Problem (IBVP). In the case of the solution of dynamics and statics of structures by the explicit form of FEM, IBVP is a shared formulation. The aim of this chapter is to give a brief overview of the continuum mechanical background of the thesis and to introduce used notation.

2.1 Continuum Kinematics

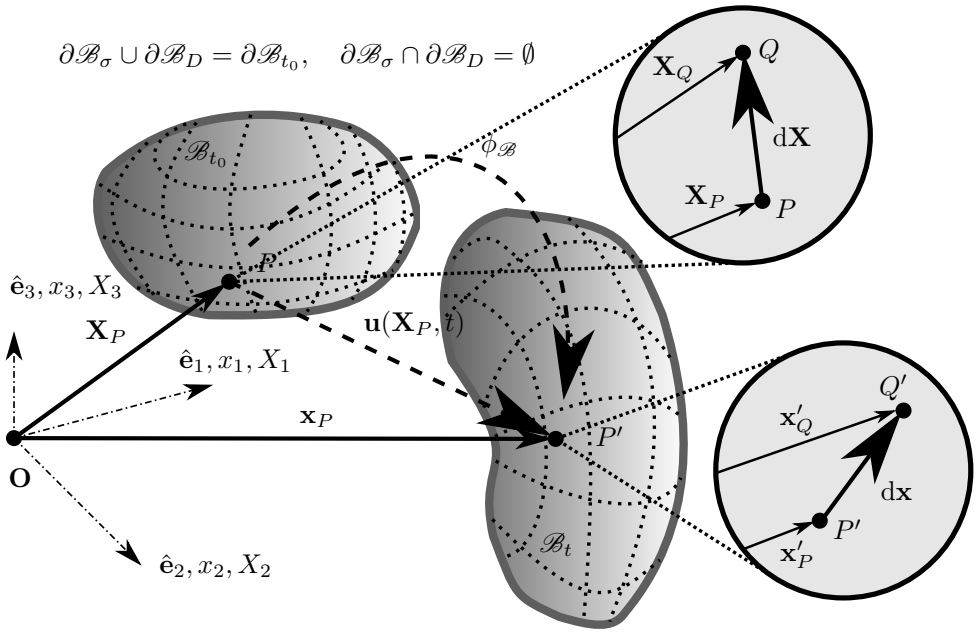


Figure 21: Continuum body $\mathscr{B}$ kinematics.

Consider a body $\mathscr{B}$ in a three-dimensional Euclidian space $\mathbb{E}^3$, which is composed of an infinite number of material elements $\mathcal{X}$. It is viewed as a compact measurable open set of particles. Each particle representing a large collection of molecules with a continuous distribution of matter in space and time. Under the influence of external forces, the body $\mathscr{B}$ will undergo macroscopic geometric changes. If the applied loads are time dependent, the deformation and geometry of the body $\mathscr{B}$ will be a function of time.

A material body $\mathscr{B}$ in motion starting from a so-called *initial configuration* $\mathscr{B}_{t_0}$ at time t_0 and as a time proceed with the application of external forces, the body will occupy a

different region $\mathscr{B}_t$ at time t, which is called *current configuration*. Particle $\mathcal{X}$ of the body $\mathscr{B}$ in initial configuration $\mathscr{B}_{t_0}$ occupies position $\mathbf{X}(\mathbf{x}, t)$, which is referred to a reference frame of right-handed, rectangular Cartesian axes X_i at a fixed origin $\mathbf{O}$ with orthonormal basis vectors $\hat{\mathbf{E}}_i$ then $\mathbf{X} = \sum_i X_i \hat{\mathbf{E}}_i$.

Deformed configuration is characterized by the mapping, which represents the bijective function $\phi_{\mathscr{B}} : \mathscr{B}_{t_0} \to \mathscr{B}_t$, which is also assumed to be reversible mapping. Mapping $\phi_{\mathscr{B}}$ takes the position vector $\mathbf{X}$ from the reference configuration $\mathscr{B}_{t_0}$ and places the same point in the deformed configuration, which we can formally characterize as $\phi_{\mathscr{B}}(\mathbf{X}, t) : \mathbf{X} \to \mathbf{x}(\mathbf{X}, t) = \mathbf{x} = \phi_{\mathscr{B}}(\mathbf{X}, t)$ with orthonormal basis vectors $\hat{\mathbf{e}}_i$ then we can write $\mathbf{x} = \sum_i x_i \hat{\mathbf{e}}$. Reversible mapping can be defined as $\phi^{-1}_{\mathscr{B}}(\mathbf{x}, t) : \mathbf{x} \to \mathbf{X} = \phi^{-1}_{\mathscr{B}}(\mathbf{x}, t)$.

The classical treatment of motion is in tracking individual particles which is referred to as the Lagrangian view, where the current coordinates $\mathbf{x} \in \mathscr{B}$ are expressed in terms of the reference coordinates $\mathbf{X} \in \mathscr{B}_{t_0}$, or in watching the motion properties in a fixed space point of Eulerian view. In an Eulerian view, coordinates $\mathbf{x}$ are termed the spatial coordinates. For a fixed value of $\mathbf{x} \in \mathscr{B}$, mapping $\phi_{\mathscr{B}}$ gives the value associated with a fixed point $\mathbf{x}$ in a space. It is associated with different material particles $\mathbf{X}$ at different times. Thus value of $\phi_{\mathscr{B}}$ is observed at the same spatial location $\mathbf{x} \in \mathscr{B}$, but occupied by different material particle.

In the study of solid bodies, the Eulerian description is not useful because the configuration $\mathscr{B}_t$ is unknown. It is the preferred description for the study of motion of fluids where configuration is known and remains unchanged. It is there that changes happen in the field of fluid velocities, pressure, density, and so on. Thus, in the Eulerian description, attention is focused on a given region of space instead of a given body of matter.

Motion of a continuous medium is also denoted by deformation, which is characterized by the rigid body motion, where the original shape of the body after the motion preserving the distance between particles, and by the motion with deformation, which is characterized by changes of distance between particles. The last effect is usually accompanied by stresses that are induced in the body. In general, motion is characterized by deformation and rigid body motion simultaneously.

For deriving the geometrical relations, so let us consider two neighboring particles in the reference configuration, which are denoted by P and Q. Let $\mathrm{d}\mathbf{X}$ be a vector joining two points P and Q in the reference configuration. After motion, particles P and Q occupy new positions P' and Q', respectively. In the new configuration, the vector joining the points P' and Q' is represented by $\mathrm{d}\mathbf{x}$. Magnitudes of $\mathrm{d}\mathbf{X}$ and $\mathrm{d}\mathbf{x}$ are denoted as follows:

$$\|PQ\| = \|\mathrm{d}\mathbf{X}\| = \mathrm{d}S, \ \|P'Q'\| = \|\mathrm{d}\mathbf{x}\| = \mathrm{d}s. \tag{2.1}$$

For the next description it is needed to define another parameters which are related to the magnitude of line elements. It is about Stretch ratio λ

$$\lambda = \frac{\|\mathrm{d}\mathbf{x}\|}{\|\mathrm{d}\mathbf{X}\|} = \frac{\mathrm{d}s}{\mathrm{d}S}, \quad 0 < \lambda < \infty,$$

where $\lambda \neq 0$ otherwise two particles would occupy the same place at the same time which has no physical meaning. Let us define the Unit Extension ϵ as follows:

$$\epsilon = \frac{\|\mathrm{d}\mathbf{x}\| - \|\mathrm{d}\mathbf{X}\|}{\|\mathrm{d}\mathbf{X}\|} = \frac{\mathrm{d}s - \mathrm{d}S}{\mathrm{d}S} = \lambda - 1 \Leftrightarrow \mathrm{d}s = (\epsilon + 1)\mathrm{d}S = \lambda \mathrm{d}S, \tag{2.2}$$

which is in range $-1 < \lambda < \infty$.

To find out the relationship between the line elements $\|\mathrm{d}\mathbf{X}\|$ and $\|\mathrm{d}\mathbf{x}\|_{\mathbb{E}^3}$ it is necessary to introduce the concept of a two-point tensor material deformation gradient $\mathbf{F}$. The deformation gradient represents a linear mapping of infinitesimal line elements $\mathrm{d}\mathbf{X}$ of the reference configuration to infinitesimal line elements $\mathrm{d}\mathbf{x}$ of the actual configuration. It is based on deformation of a continuum as relative displacements and changes in geometry under the influence of an action of forces. The displacement of the particle $\mathbf{X}$ is defined by relationship as follows:

$$\boldsymbol{X}^Q = \mathbf{X}^P + \mathrm{d}\mathbf{X}, \quad \mathrm{d}\mathbf{x} = \mathbf{x}^Q(\mathbf{X}^Q, t) - \mathbf{x}^P(\mathbf{X}^P, t). \tag{2.3}$$

Then can be observed that $\mathbf{x}^Q(\mathbf{X}^Q, t) = \mathbf{x}^P(\mathbf{X}^P + \mathrm{d}\mathbf{X}, t) = \mathbf{x}(\mathbf{X} + \mathrm{d}\mathbf{X}, t)$, the vector field $\mathrm{d}\mathbf{x}$ in the current configuration becomes:

$$\mathrm{d}\mathbf{x} = \mathbf{x}(\mathbf{X} + \mathrm{d}\mathbf{X}, t) - \mathbf{x}(\mathbf{X}, t). \tag{2.4}$$

By applying Taylor series (higher order terms are discarded) to represent the function in (2.4) leads to expression:

$$\begin{aligned} \mathrm{d}\mathbf{x} &= \left(\sum_j \frac{\partial \mathbf{x}_i}{\partial \mathbf{X}_j} \mathrm{d}\mathbf{X}_j \right) \hat{\mathbf{e}}_i + \mathcal{O}(\|\mathrm{d}\mathbf{X}\|^2) \\ \mathrm{d}\mathbf{x} &= \mathbf{F} \cdot \mathrm{d}\mathbf{X}, \end{aligned} \tag{2.5}$$

which is defined in terms of mapping $\phi_{\mathscr{B}}$ as

$$\mathbf{F} = \mathbf{F}(\mathbf{X}, t) \equiv \partial_{\mathbf{X}} \phi_{\mathscr{B}} = \partial_{\mathbf{X}} \mathbf{x} = \mathrm{Grad}_{\mathbf{X}}\ \mathbf{x}. \tag{2.6}$$

Due to one-to-one mapping, the deformation gradient $\mathbf{F}$ is not allowed to be singular so its inverse is

$$\mathbf{F}^{-1}(\mathbf{X}, t) \equiv \partial_{\mathbf{x}} \mathbf{X} = \mathrm{Grad}_{\mathbf{x}}\ \mathbf{X}, \tag{2.7}$$

thus a sufficient condition for the existence of the inverse of $\mathbf{F}$ is that the *Jacobian* J of $\mathbf{F}$ is not equal to zero and with the continuity of the mapping $\phi_{\mathscr{B}}$

$$J = \det\ \mathbf{F}(\mathbf{X}, t) > 0. \tag{2.8}$$

Similarly as for the transformation of infinitesimal line element, the same applies to the area and the volume element of the respective continuum subjected by the deformation. The normal to an infinitesimal material area element $\mathrm{d}\mathbf{A} = \hat{\mathbf{N}} \cdot \mathrm{d}A$, with the material unit outward normal vector $\hat{\mathbf{N}}$ are mapped to the normal of an infinitesimal spatial area

element $\mathrm{d}\mathbf{a} = \hat{\mathbf{n}} \cdot \mathrm{d}a$ with the spatial unit outward normal vector $\hat{\mathbf{n}}$, via *Nanson's Formula* as follows:

$$\mathrm{d}\mathbf{a} = J\mathbf{F}^{-T} \cdot \mathrm{d}\mathbf{A}. \tag{2.9}$$

In the case of a mapping of infinitesimal volume elements, the situation is somewhat more complicated. It represents mapping of the infinitesimal material and spatial volume elements $\mathrm{d}V$ and $\mathrm{d}v$, respectively. These are defined as triple product of the corresponding infinitesimal line elements $\mathrm{d}V = \mathrm{d}\mathbf{X}_1 \cdot (\mathrm{d}\mathbf{X}_2 \times \mathrm{d}\mathbf{X}_3)$ and $\mathrm{d}v = \mathrm{d}\mathbf{x}_1 \cdot (\mathrm{d}\mathbf{x}_2 \times \mathrm{d}\mathbf{x}_3)$. By the considering the relation (2.5), it yields to the mapping of an infinitesimal volume elements as follows (proof of this conclusion is listed in the appendix):

$$\mathrm{d}v = \det \mathbf{F} \cdot \mathrm{d}V = J\mathrm{d}V. \tag{2.10}$$

The deformation gradient can be also expressed in terms of the displacement vector $\mathbf{u} = \phi_{\mathscr{B}}(\mathbf{X}, t) - \mathbf{X}$ as

$$\mathbf{F} = \mathrm{Grad}\ \mathbf{x} = \mathrm{Grad}\ \mathbf{u} + \mathbf{I}. \tag{2.11}$$

The change in the squared lengths that occurs as a body deforms from the reference to the current configuration can be expressed relative to the original length as

$$(\mathrm{d}s)^2 - (\mathrm{d}S)^2 = 2\mathrm{d}\mathbf{X} \cdot \mathbf{E} \cdot \mathrm{d}\mathbf{X}, \tag{2.12}$$

where $\mathbf{E}$ denotes nonlinear (quadratic) function of deformation. It is named as Green–Lagrange (Green–St. Venant) symmetric second-order (finite) strain tensor, which is an alternative to Euler-Almansi strain tensor defined in spatial coordinates of Eulerian description. It can be expressed as follows:

$$\begin{aligned} \mathbf{E}(\mathbf{F}) = \mathbf{E} &= \frac{1}{2}\left(\mathbf{F}^T \cdot \mathbf{F} - \mathbf{I}\right) \\ &= \frac{1}{2}\left((\mathbf{I} + \mathrm{Grad}\ \mathbf{u}) \cdot (\mathbf{I} + \mathrm{Grad}\ \mathbf{u})^T - \mathbf{I}\right) \\ &= \frac{1}{2}\left(\mathrm{Grad}\ \mathbf{u} + (\mathrm{Grad}\ \mathbf{u})^T + (\mathrm{Grad}\ \mathbf{u}) \cdot (\mathrm{Grad}\ \mathbf{u})^T\right). \end{aligned} \tag{2.13}$$

For the case of specific material location $\mathbf{X}'$, the density of Helmholtz free energy $\Psi \circ \phi_{\mathscr{B}}(\mathbf{X}')$ need to reflect the deformation behavior in an infinitesimal neighborhood of $\mathbf{X}'$. Then is it possible to approximate the deformation map $\phi_{\mathscr{B}}$ using a first-order Taylor expansion as follows:

$$\begin{aligned} \phi_{\mathscr{B}}(\mathbf{X}) \approx \phi_{\mathscr{B}}(\mathbf{X}') + \left.\frac{\partial \phi_{\mathscr{B}}}{\mathbf{X}}\right|_{\mathbf{X}'} (\mathbf{X} - \mathbf{X}') &= \mathbf{x}' - \mathbf{F}(\mathbf{X}')\left(\boldsymbol{X} - \mathbf{X}'\right) \\ &= \mathbf{x}' - \mathbf{F}(\mathbf{X}')\mathbf{X} + \mathbf{F}(\mathbf{X}')\mathbf{X}' \\ &= \mathbf{F}'\mathbf{X} + \mathbf{t}. \end{aligned} \tag{2.14}$$

The respective equation suggests that $\Psi \circ \phi_{\mathscr{B}}(\mathbf{X}')$ is expressible as a function of $\mathbf{F}'$ and $\mathbf{t}$. These values fully parameterize the local Taylor approximation of $\phi_{\mathscr{B}}$ near $\mathbf{X}'$. Vector $\mathbf{t}$

indicates deformations that differ only by a constant translation and thus is irrelevant. It is referred to as rigid transformation. It causes the production the same deformed shape and the same strain energy. Thus, the energy density function in finite elasticity should be expressible as a function of the local deformation gradient $\Psi \circ \phi_{\mathscr{B}}(\mathbf{X}) = \Psi \circ \mathbf{F}(\mathbf{X})$. Expression of the function $\Psi(\mathbf{F})$ is determinated by the specific material model. Finite elasticity is primarily taken into consideration in this paper.

Thus when a body undergoes rigid body motion (rotated and translated) from its reference position $\phi_{\mathscr{B}} = \mathbf{R}\mathbf{X} + \mathbf{t}$ without changing its shape (the distance between particles are constant during motion), which is a specific case of homogenous deformation, then the Cauchy-Green deformation tensor $\mathbf{C} = \mathbf{F}^T \cdot \mathbf{F}$ related to rigid body motion bacome $\mathbf{F} = \mathbf{R}, \mathbf{R}^T \cdot \mathbf{R} = \mathbf{I} \Rightarrow \mathbf{E} = \mathbf{0}$. Where $\mathbf{R}$ represents rotation matrix.

$$\mathbf{E}(\mathbf{F})|_{\mathbf{F}=\mathbf{R}} = \frac{1}{2}(\mathbf{C} - \mathbf{I}) = \frac{1}{2}(\underbrace{\mathbf{R}^T \cdot \mathbf{R}}_{=\mathbf{I}} - \mathbf{I}) = \mathbf{0}. \tag{2.15}$$

Non-singular second-order tensor $\mathbf{F}$ can be decomposed multiplicatively by means of the *Polar decomposition theorem* for a separation of rotating part without deformation energy which caueses moving of the body $\mathscr{B}$ in space only. Polar decomposition has a form as follows:

$$\mathbf{F} = \underbrace{\mathbf{R} \cdot \mathbf{U}}_{\text{right polar decomposition}} = \underbrace{\mathbf{V} \cdot \mathbf{R}}_{\text{left polar decomposition}}, \tag{2.16}$$

where $\mathbf{R}$, as mentioned earlier, is denoted as a proper orthogonal tensor (rotation tensor), which must meet the characteristics orthogonality

$$\mathbf{R}^T \cdot \mathbf{R} = \mathbf{R} \cdot \mathbf{R}^T = \mathbf{I} \Rightarrow \mathbf{R}^T = \mathbf{R}^{-1}, \quad \det \mathbf{R} = 1. \tag{2.17}$$

Tensor $\mathbf{U}$ is the right stretch tensor (*Lagrangian stretch tensor*) and $\mathbf{V}$ is the left stretch tensor (*Eulerian stretch tensor*). Both are the symmetric positive definite tensors. Polar decomposition is necessary to perform for unisotropic materials.

Complexity of constitutive models that are constructed based on this kind of deformation measure, will lead to discretizations with nodal forces being nonlinear functions of nodal positions. In an effort to remedy this, it is possible to construct a linear approximation of equation (2.13) by forming a Taylor expansion around the undeformed configuration $\mathscr{B}_{t_0}$ without initial stress, where $\mathbf{F} = \mathbf{I}$, then

$$\mathbf{E}(\mathbf{F}) \approx \operatorname{lin} \mathbf{E} \equiv \mathbf{E}(\mathbf{I}) + \left.\frac{\partial \mathbf{E}}{\partial \mathbf{F}}\right|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F} - \mathbf{I}), \tag{2.18}$$

where

$$\partial_{\mathbf{F}}\mathbf{E} : \delta\mathbf{F} = \delta\mathbf{E} = \frac{1}{2}\left(\delta\mathbf{F}^T\mathbf{F} + \mathbf{F}^T\delta\mathbf{F}\right), \tag{2.19}$$

thus

$$\mathbf{E}(\mathbf{I}) + \left.\frac{\partial \mathbf{E}}{\partial \mathbf{F}}\right|_{\mathbf{F}=\mathbf{I}} = \frac{1}{2}\left((\mathbf{F} - \mathbf{I})^T\mathbf{I} + \mathbf{I}^T(\mathbf{F} - \mathbf{I})\right) = \frac{1}{2}\left(\mathbf{F} + \mathbf{F}^T\right) - \mathbf{I} = \boldsymbol{\varepsilon}. \tag{2.20}$$

All displacement gradients are small and the nonlinear terms in the definition of the Green–Lagrange strain tensor $\mathbf{E}$ (2.13) are neglected. This leads to obtaining the linearized form of Green–Lagrange strain tensor $\boldsymbol{\varepsilon}$ (infinitesimal strain tensor)

$$\mathbf{E} \approx \text{lin } \mathbf{E} \equiv \boldsymbol{\varepsilon} = \frac{1}{2}\left(\text{grad } \mathbf{u} + (\text{grad } \mathbf{u})^T\right) = \text{grad}^{\text{sym}} \mathbf{u}. \tag{2.21}$$

The displacement gradient tensor can be expressed as the sum of a symmetric tensor and a skew symmetric tensor. We have

$$\begin{aligned}(\text{grad } \mathbf{u})^T &= \frac{1}{2}\left((\text{grad } \mathbf{u})^T + \text{grad } \mathbf{u}\right) + \frac{1}{2}\left((\text{grad } \mathbf{u})^T - \text{grad } \mathbf{u}\right) \\ &= \text{grad}^{\text{sym}} \mathbf{u} + \text{grad}^{\text{skew}} \mathbf{u} \\ &= \tilde{\boldsymbol{\varepsilon}} + \boldsymbol{\Omega}; \quad |\text{Grad } \mathbf{u}| \approx |\text{grad } \mathbf{u}| \ll 1; \quad \boldsymbol{\Omega}^T = -\boldsymbol{\Omega},\end{aligned} \tag{2.22}$$

where the symmetric part is similar to the infinitesimal strain tensor, and the skew symmetric part is infinitesimal rotation (spin) tensor.

2.2 Stress Measure

When an external force is acting on a body $\mathscr{B}$, the atoms or molecules that make up the continuum are affected and undergo a position change to achieve balance. Resistance to this movement depends on the characteristics of the atoms or molecules that make up the continuum. Internal resistance is characterized by internal force, and is usually interpreted as the average of the interatomic forces of a handful of atoms, thereby characterizing internal force as a macroscopic variable. The internal force at each material point of the continuum is represented by the traction vector field, which is the starting point to establish the stress state at a material point. Stress can be measured per unit deformed area or undeformed area.

For the further general formulation, it is necessary to consider the stress in the deformed configuration. Stress at a point in a three-dimensional continuum can be measured in terms of nine quantities, three per plane, on three mutually perpendicular planes at the point. These nine quantities may be viewed as the components of a second-order stress tensor.

Consider a body $\mathscr{B}$ in the current configuration which has been divided into two parts by a plane. Plane is defined by point $P(\mathbf{x}, t)$ and by the normal (unit vector) $\hat{\mathbf{n}}$ to that plane, which denote the direction of a plane area. Then the traction vector (stress vector) has a form defined by the *Cauchy's Fundamental Postulate* as $\mathbf{t}(\mathbf{x}, t, \hat{\mathbf{n}})$. It is defined at the point $P(\mathbf{x}, t)$ where it is associated with the normal vector $\hat{\mathbf{n}}$ to the respective plane. Then it can be defined as follows:

$$\mathbf{t}(\mathbf{x}, t, \hat{\mathbf{n}}) = \lim_{\Delta a \to \infty} \left(\frac{\mathbf{f}}{\Delta a}\right), \tag{2.23}$$

$$\mathrm{d}\mathbf{f} = \mathbf{t}(\mathbf{x}, t, \hat{\mathbf{n}}) \cdot \mathrm{d}a. \tag{2.24}$$

Result of Cauchy's fundamental postulate is the *Principle of Action and Reaction*. Traction vector on three mutually perpendicular planes passing through the point $P(\mathbf{x}, t)$ can fully describe the stress state at that point. Then it leads to obtaining three traction vectors associated with each direction by which is defined a symetric second-order Cauchy stress tensor (called the *true stress tensor*) $\boldsymbol{\sigma}$ as

$$\boldsymbol{\sigma} = \sigma_{ij} \left(\hat{\mathbf{e}}_i \otimes \hat{\mathbf{e}}_j \right), \ \boldsymbol{\sigma} = \boldsymbol{\sigma}^T. \tag{2.25}$$

The Cauchy stress tensor is the most natural and physical measure of the state of stress at a point in the deformed configuration $\mathscr{B}_t$ of the body $\mathscr{B}$ and measured per unit area of the deformed configuration. Although, for the purposes of solid mechanics the Lagrangian description is primarily used as it is necessary to define a measure of stress in a deformed configuration regard to the reference configuration of the body $\mathscr{B}$, which is known prior.

Let us define a stress vector $\mathbf{T}(\mathbf{X}, \hat{\mathbf{N}})$ over the area element $\mathrm{d}A$ with normal $\hat{\mathbf{N}}$ in the undeformed (reference) configuration such that it result in the same total force as (2.24)

$$\mathrm{d}\mathbf{f} = \mathbf{T}(\mathbf{X}, \hat{\mathbf{N}}) \cdot \mathrm{d}A. \tag{2.26}$$

The vector $\mathbf{T} = \mathbf{P} \cdot \hat{\mathbf{N}}$ is known as the *pseudo stress vector*, measured per unit undeformed area $\mathrm{d}A$ and it is associated with the first Piola-Kirchhoff stress tensor $\mathbf{P}$, which is unsymmetric tensor defined as follows:

$$\begin{aligned} \mathbf{P} \cdot \mathrm{d}\mathbf{A} &= \boldsymbol{\sigma} \cdot \mathrm{d}\mathbf{a} = J\boldsymbol{\sigma} \cdot \mathbf{F}^{-T} \cdot \mathrm{d}A \\ \mathbf{P} &= J\boldsymbol{\sigma} \cdot \mathbf{F}^{-T}. \end{aligned} \tag{2.27}$$

For practical use in nonlinear mechanics, however, the second Piola-Kirchoff stress is used, which is associated with the force in the undeformed area $\mathrm{d}A$. It is symetric second-order tensor defined as follows:

$$\begin{aligned} \mathbf{F}^{-1} \cdot \mathrm{d}\mathbf{f} &= \mathbf{F}^{-1} \cdot (\mathbf{P} \cdot \mathrm{d}\mathbf{A}) = \mathbf{S} \cdot \mathrm{d}\mathbf{A} \\ \mathbf{S} &= \mathbf{F}^{-1} \cdot \mathbf{P} = J\mathbf{F}^{-1} \cdot \boldsymbol{\sigma} \cdot \mathbf{F}^{-T}. \end{aligned} \tag{2.28}$$

2.3 Governing Equations of Structural Dynamics

As already mentioned in the introduction to chapter 2, the pursuit of the most general formulation of problems in continuum mechanics leads in the case of the dynamics of deformable bodies to the shared formulation with the dynamics of fluids, which is generally known by the name of Navier-Stokes equations. This is important due to the physical similarities between the deformable bodies and liquids. These conclusions are primarily used in the case of modeling various types of constitutive relations in the modern concept of continuum mechanics. This observation has also significant influence on the general view of the problem under the study of the dynamics of structures, mainly in connection with some generalizations of some of the approaches that are presented in the following chapters.

The motion of particles without looking at how the relative motion between particles changed is described by the material or total time derivative and is denoted as $\frac{\mathrm{D}}{\mathrm{D}t}[\bullet]$. Statement in brackets $[\bullet]$ denotes scalar, vector, tensor, atc. function at an arbitrary point of studied continuum. This description of the motion in general sense is especially important for the derivation of equations of motion. It is the superset for deriving the geometrical relations that concerns the deformation as a change in distance between particles. It differs, however, if it is considered in the Lagrangian or Eulerian meaning. Respective equations of motion will be listed in both the Eulerian and the Lagrangian framework in order to show differences between both formulations. In the rest of this paper the Lagrangian formulation is prioritized.

The difference is presented in the definition of a total time (or material) derivatives as follows:

1. Eulerian description:

$$\begin{aligned} \mathbf{X} &= \mathbf{X}(\mathbf{x},t) = \phi_{\mathscr{B}}(\mathbf{x},t) \\ \frac{\mathrm{D}}{\mathrm{D}t}\Big[\phi_{\mathscr{B}}(\mathbf{x},t)\Big] &= \partial_t\phi_{\mathscr{B}}(\mathbf{x},t) + \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}\partial_{\mathbf{x}}\phi_{\mathscr{B}}(\mathbf{x},t) \\ &= \underbrace{\partial_t\phi_{\mathscr{B}}}_{\text{local rate of change}} + \underbrace{\mathbf{v}\cdot\operatorname{grad}\,\phi_{\mathscr{B}}}_{\text{convective rate of change}} . \end{aligned}$$

2. Lagrangian description:

$$\begin{aligned} \mathbf{x} &= \mathbf{x}(\mathbf{X},t) = \phi_{\mathscr{B}}(\mathbf{X},t) \\ \frac{\mathrm{D}}{\mathrm{D}t}\Big[\phi_{\mathscr{B}}(\mathbf{X},t)\Big] &= \partial_t\phi_{\mathscr{B}}. \end{aligned}$$

Consider the body $\mathscr{B}_t$ in motion subjected by the volume forces $\mathbf{b}(\mathbf{x},t)$ and the contact surface traction forces $\mathbf{t}(\mathbf{x},t)$, acting on the boundary $\partial\mathscr{B}_\sigma$ of a body, in Eulerian kind of view. Considering the velocity field $\mathbf{v}(\mathbf{x},t)$ let us define the linear momentum $\mathbf{p}(\mathbf{x},t)$ of the body $\mathscr{B}$ as follows:

$$\mathbf{p}(\mathbf{x},t) = \int_{\mathscr{B}_t} \varrho\mathbf{v}(\mathbf{x},t)\,\mathrm{d}t = \int_{\mathscr{B}_t} \varrho\mathbf{v}\,\mathrm{d}v.$$

Now let's consider the *Principle of Balance of Linear Momentum*, which is based on the Newton's second law of motion. This law expresses the meaning that the rate of change of linear momentum $\mathbf{p}$ of an arbitrarily part of a continuum body $\mathscr{B}_t$ is proportional to the sum of volume and surface forces

$$\underbrace{\frac{\mathrm{D}}{\mathrm{D}t}\left[\int_{\mathscr{B}_t}\mathbf{p}\,\mathrm{d}v\right]}_{\text{rate of change of linear momentum}} = \frac{\mathrm{D}}{\mathrm{D}t}\left[\int_{\mathscr{B}_t}\varrho\mathbf{v}\,\mathrm{d}v\right] = \underbrace{\int_{\partial\mathscr{B}_\sigma}\mathbf{t}\,\mathrm{d}a}_{\text{traction forces on boundary }\partial\mathscr{B}_\sigma} + \underbrace{\int_{\mathscr{B}_t}\varrho\mathbf{b}\,\mathrm{d}v}_{\text{volume forces}}.$$

The rate of change of linear momentum is expressed by the *Reynolds transport theorem* as follows:

$$\begin{aligned}\frac{\mathrm{D}}{\mathrm{D}t}\left[\int_{\mathscr{B}_t}\varrho\mathbf{v}\ \mathrm{d}v\right] &= \int_{\mathscr{B}_t}\partial_t\left(\varrho\mathbf{v}\right)\ \mathrm{d}v+\int_{\partial\mathscr{B}_\sigma}\varrho\mathbf{v}\otimes\left(\mathbf{v}\cdot\hat{\mathbf{n}}\right)\ \mathrm{d}a\\ &= \int_{\mathscr{B}_t}\Big(\partial_t\left(\varrho\mathbf{v}\right)+\mathrm{div}\left(\varrho\mathbf{v}\otimes\mathbf{v}\right)\Big)\ \mathrm{d}v\\ &= \int_{\mathscr{B}_t}\varrho\left(\partial_t\mathbf{v}+\mathbf{v}\cdot\mathrm{grad}\ \mathbf{v}\right)\ \mathrm{d}v\\ &= \int_{\mathscr{B}_t}\varrho\mathbf{a}\ \mathrm{d}v,\end{aligned}\tag{2.29}$$

where $\mathbf{a}=\mathbf{a}(\mathbf{x},t)$ is acceleration of the body $\mathscr{B}_t$, ϱ is material density and $\hat{\mathbf{n}}$ is outward normal to the boundary $\partial\mathscr{B}_\sigma$. The equation (2.29) can be rewritten to the form:

$$\int_{\mathscr{B}_t}\varrho\mathbf{a}\ \mathrm{d}v=\int_{\partial\mathscr{B}_\sigma}\mathbf{t}\ \mathrm{d}a+\int_{\mathscr{B}_t}\varrho\mathbf{b}\ \mathrm{d}v,$$

where

$$\int_{\partial\mathscr{B}_\sigma}\mathbf{t}\ \mathrm{d}a=\int_{\partial\mathscr{B}_\sigma}\boldsymbol{\sigma}\cdot\hat{\mathbf{n}}\ \mathrm{d}a=\int_{\partial\mathscr{B}_\sigma}\hat{\mathbf{n}}\cdot\boldsymbol{\sigma}^T\ \mathrm{d}a=\int_{\mathscr{B}_t}\mathrm{div}\ \boldsymbol{\sigma}^T\ \mathrm{d}v,$$

then

$$\int_{\mathscr{B}_t}\left(\mathrm{div}\ \boldsymbol{\sigma}^T+\varrho\mathbf{b}-\varrho\mathbf{a}\right)\ \mathrm{d}v=\mathbf{0}.\tag{2.30}$$

Equation (2.30) can be expressed in differential form, which is now valid locally

$$\boxed{\mathrm{div}\ \boldsymbol{\sigma}^T+\varrho\mathbf{b}=\varrho\mathbf{a}\ \text{in}\ \mathscr{B}\times[0,\tau]}.\tag{2.31}$$

Equations (2.31) are called *Cauchy's equations of motion* with explicitly included forces from the effects of viscous damping. It is expressed in Eulerian space description and they are counterpart to Newton's second law of motion valid in continuum. For the solvability of these equations it is necessary to prescribe the essential (Dirichlet) and natural (Neumann, Newton, Robin) boundary conditions and initial condition at time t_0

$$\mathbf{u}(\mathbf{x},t)=\overline{\mathbf{u}}\ \text{on}\ \partial\mathscr{B}_D\times[0,\tau],\ \ \mathbf{t}=\boldsymbol{\sigma}\cdot\hat{\mathbf{n}}\ \text{on}\ \partial\mathscr{B}_\sigma\times[0,\tau],\ \ \mathbf{v}(t_0)=\overline{\mathbf{v}}_0\ \text{in}\ \mathscr{B},\ \ \mathbf{u}(t_0)=\overline{\mathbf{u}}_0\ \text{in}\ \mathscr{B}.$$

We have to also consider the *Principle of Balance of Angular Momentum*. The principle yields, in the absence of body couples, symmetry of of the Cauchy stress tensor. It is expressed as

$$\int_{\partial\mathscr{B}_\sigma}\mathbf{x}\times\mathbf{t}\ \mathrm{d}a+\int_{\mathscr{B}_t}\mathbf{x}\times\varrho\mathbf{b}\ \mathrm{d}v=\frac{\mathrm{D}}{\mathrm{D}t}\int_{\mathscr{B}_t}\mathbf{x}\times\varrho\mathbf{v}\ \mathrm{d}v,\tag{2.32}$$

which yields, in view of the equations of motion (2.31), the symmetry of the Cauchy stress tensor

$$\boldsymbol{\sigma}=\boldsymbol{\sigma}^T,\quad \sigma_{ij}=\sigma_{ji}.$$

The form of the equations of motion with boundary and initial conditions in the Lagrangian description a form:

$$\boxed{\text{Div } \mathbf{P}^T + \varrho_{\mathbf{X}}\mathbf{B} = \varrho_{\mathbf{X}}\mathbf{A} \text{ in } \mathscr{B}_{\mathbf{X}} \times [0,\tau]}.$$

$\mathbf{u}(\mathbf{X},t) = \overline{\mathbf{u}}$ on $\partial\mathscr{B}_D \times [0,\tau]$, $\mathbf{T} = \mathbf{P}\cdot\hat{\mathbf{N}}$ on $\partial\mathscr{B}_\sigma \times [0,\tau]$, $\mathbf{V}(t_0) = \overline{\mathbf{V}}_0$ in $\mathscr{B}_{\mathbf{X}}$, $\mathbf{u}(t_0) = \overline{\mathbf{u}}_0$ in $\mathscr{B}_{\mathbf{X}}$.

2.4 Constitutive Equations

Constitutive equations generally represent different ways of idealizing the response of a material based on a macroscopic point of view from experimental evidence. Mathematically, its purpose is to establish connections between kinematic, thermal and mechanical variables. In mechanics of deformable bodies it is about the bijective relationships between stress and strain. The equations that relate state functions to state variables are called the equations of state or constitutive equations and state variables, the selection of which depends on the respective problem, are those that depend only on themselves.

Strains and temperature are used as independent variables for solids. In thermodynamics state functions are considered and are determined by the state variables. Let us consider Helmholtz free energy Ψ, Cauchy stress tensor $\boldsymbol{\sigma}$, entropy η, and the heat flux $\boldsymbol{q}$. Then a material respond is completely defined by the fields $\Psi, \boldsymbol{\sigma}, \eta$ and $\boldsymbol{q}$. For the specific occassions it may be more appropriate to use other thermodynamic potentials. Constitutive equations must also respect some restrictions (see [50]).

2.4.1 Generalized Standard Materials

The so-called *Generalized Standard Materials* (GStM) is a term for locally valid material models which are modeled by some differentiable potential; they often depend on a finite number of parameters (see [120]). The concept of this type of materials was introduced by Halphen and Nguyen (see [40]). They showed the analogies between the properties of some workhardening *Standard Materials* (StM) and those of perfectly plastic or viscoplastic standard ones. This is mainly due to capturing the multivalued constitutive laws, for example plastic flow rules, they were extended to GStM modelled by lower-semi-continuous convex potentials (see [125]). Simply said, the whole problem of specifying a constitutive law is reduced to specifying two potentials—the *elastic energy density* (free energy) and the *dissipation potential*. A material obeying such a law is called just GStM (see [123], [86]).

Consider strain-like variable $\varepsilon \in X_\varepsilon$, where X_ε is a real Banach space and a stress-like variable $\sigma \in X_\sigma^*$, where X_σ^* is the continuous dual space, which are subset of linear space $\mathcal{L}_{\sigma:\varepsilon} : X_\sigma^* \times X_\varepsilon$, where a respective subset has properties of maximal Lagrangian submanifold of the $\mathcal{L}_{\sigma:\varepsilon}$, then there exists a differentiable function Ψ (let us call it *Helmholtz free energy* potential), such that

$$\boxed{\sigma = \delta\Psi(\varepsilon)}. \tag{2.33}$$

Just the material model of such a type whose behavior can be described by a differentiable potential is then referred to StM as a subset of GStM. Respective material from this class is then considered in the thesis.

2.4.2 Thermodynamics of (Hyper-) Elastic Materials

Materials such as elastomers, polymers, or biological matter may be subject to large deformations with non-dissipative behaviour $\mathcal{R} = 0$ (dissipative rate), where there is no entropy and temperature change as a typical characteristic of elastic (reversible) processes. These materials are not dependent on the history of loading, but only on the current values of the state variables. This is so called a class of *Hyperelastic materials* and also known as *Green* or *nonlinear elasticity* materials. These materials belong to the class of StM described in chapter 2.4.1.

Description of the internal behavior of hyperelastic materials presupposes the existence of a certain thermodynamic energy potential, which is known as density of Helmholtz free energy Ψ (strain energy density), which depends only on deformation gradient $\mathbf{F}$

$$\Psi = \Psi\left(\mathbf{F}\right).$$

Consider a thermodynamic system with known forms of involved energy. Then we can compose energy bilance, which is known as the *first law of thermodynamics*. It is the principle of conservation of energy for continuum thermodynamics, and it is given in the following form:

$$\frac{\mathrm{D}}{\mathrm{D}t}\left(\mathcal{K}+\mathcal{U}\right) = \mathcal{W} + \mathcal{Q}. \tag{2.34}$$

It express the equality between the rate of change of kinetic $\mathcal{K}$ and internal $\mathcal{U}$ energy, and the thermal power done by system $\mathcal{W}$ and added to the system $\mathcal{Q}$. Where the thermal power $\mathcal{W}$ can be rewritten as

$$\begin{aligned}
\mathcal{W} &= \int_{\mathscr{B}_{\mathbf{X}}} \varrho_{\mathbf{X}}\mathbf{B}\cdot\mathbf{V}\,\mathrm{d}V + \int_{\partial\mathscr{B}_{\mathbf{X}}} \mathbf{T}\cdot\mathbf{V}\,\mathrm{d}A = \int_{\mathscr{B}_{\mathbf{X}}} \varrho_{\mathbf{X}}\mathbf{B}\cdot\mathbf{V}\,\mathrm{d}V + \int_{\partial\mathscr{B}_{\mathbf{X}}} (\hat{\mathbf{N}}\cdot\mathbf{P}^T)\cdot\mathbf{V}\,\mathrm{d}A \\
&= \int_{\mathscr{B}_{\mathbf{X}}} \Big(\varrho_{\mathbf{X}}\mathbf{B}\cdot\mathbf{V} + \mathrm{Div}(\mathbf{P}^T\cdot\mathbf{V})\Big)\,\mathrm{d}V \\
&= \int_{\mathscr{B}_{\mathbf{X}}} \Big((\varrho_{\mathbf{X}}\mathbf{B} + \mathrm{Div}\ \mathbf{P}^T)\cdot\mathbf{V} + \mathbf{P}^T : \mathrm{Grad}\ \mathbf{V}\Big)\,\mathrm{d}V \\
&= \int_{\mathscr{B}_{\mathbf{X}}} \Big((\mathrm{Div}\ \mathbf{P}^T)\cdot\mathbf{V} + \mathbf{P}^T : \mathrm{Grad}\ \mathbf{V} + \varrho_{\mathbf{X}}\mathbf{B}\cdot\mathbf{V}\Big)\,\mathrm{d}V,
\end{aligned} \tag{2.35}$$

where

$$\mathbf{P}^T : \mathrm{Grad}\ \mathbf{V} = \mathbf{P}^T : \partial_t\mathbf{F} = (\mathbf{S}\cdot\mathbf{F}^T) : \partial_t\mathbf{F}$$

and the thermal power $\mathcal{Q}$ can be rewritten as

$$\begin{aligned}\mathcal{Q} &= \int_{\mathscr{B}_\mathbf{X}} \varrho_\mathbf{X} r_h \,\mathrm{d}V - \int_{\partial\mathscr{B}_\mathbf{X}} \hat{\mathbf{N}}\cdot\mathbf{q}_\mathbf{X}\,\mathrm{d}A \\ &= \int_{\mathscr{B}_\mathbf{X}} \left(\varrho_\mathbf{X} r_h - \mathrm{Div}\ \mathbf{q}_\mathbf{X}\right)\,\mathrm{d}V, \end{aligned} \tag{2.36}$$

where r_h is internal heat generation per unit mass and $\boldsymbol{q}$ is heat flux vector.

By expressing all members of the equation (2.34), where $\boldsymbol{V}$ is considered as velocity field in Lagrangian description, we get

$$\begin{aligned}\frac{\mathrm{D}}{\mathrm{D}t}\underbrace{\left(\int_{\mathscr{B}_\mathbf{X}}\frac{1}{2}\varrho_\mathbf{X}(\mathbf{V}\cdot\mathbf{V})\,\mathrm{d}V + \int_{\mathscr{B}_\mathbf{X}}\varrho_\mathbf{X} e\,\mathrm{d}V\right)}_{\text{kinetic }\mathcal{K}\text{ and internal }\mathcal{U}\text{ energy}} &= \underbrace{\int_{\mathscr{B}_\mathbf{X}}\varrho_\mathbf{X}\mathbf{B}\cdot\mathbf{V}\,\mathrm{d}V + \int_{\partial\mathscr{B}_\mathbf{X}}\mathbf{T}\cdot\mathbf{V}\,\mathrm{d}A}_{\text{thermal power }\mathcal{W}} + \\ &+ \underbrace{\int_{\mathscr{B}_\mathbf{X}}\varrho_\mathbf{X} r_h\,\mathrm{d}V - \int_{\partial\mathscr{B}_\mathbf{X}}\hat{\mathbf{N}}\cdot\mathbf{q}_\mathbf{X}\,\mathrm{d}A}_{\text{thermal power }\mathcal{Q}},\end{aligned}$$

then substituting equations (2.35) and (2.36) we get

$$\begin{aligned}\int_{\mathscr{B}_\mathbf{X}}\varrho_\mathbf{X}\partial_t e\,\mathrm{d}V &= \int_{\mathscr{B}_\mathbf{X}}\Big(\Big((\mathrm{Div}\ \mathbf{P}^T)\cdot\mathbf{V} + \mathbf{P}^T : \partial_t\mathbf{F} + \varrho_\mathbf{X}\mathbf{B}\cdot\boldsymbol{V}\Big) + \\ &+ \varrho_\mathbf{X} r_h - \mathrm{Div}\ \mathbf{q}_\mathbf{X} - \varrho_\mathbf{X}(\mathbf{V}\cdot\mathbf{A})\Big)\,\mathrm{d}V.\end{aligned}$$

Additionally, by rearranging the above equation we obtain

$$\begin{aligned}\int_{\mathscr{B}_\mathbf{X}}\varrho_\mathbf{X}\partial_t e\,\mathrm{d}V &= \int_{\mathscr{B}_\mathbf{X}}\Big(\underbrace{(\mathrm{Div}\ \mathbf{P}^T + \varrho_\mathbf{X}\mathbf{B} - \varrho_\mathbf{X}\mathbf{A})}_{\substack{\text{the equations}\\ \text{of motion}\Rightarrow\mathbf{0}}}\cdot\mathbf{V} + \\ &+ \mathbf{P}^T : \partial_t\mathbf{F} + \varrho_\mathbf{X} r_h - \mathrm{Div}\ \mathbf{q}_\mathbf{X}\Big)\,\mathrm{d}V\end{aligned}$$

The local form of the above equation is known as the *energy equation*

$$\varrho_\mathbf{X}\partial_t e = \mathbf{P}^T : \partial_t\mathbf{F} + \varrho_\mathbf{X} r_h - \mathrm{Div}\ \mathbf{q}_\mathbf{X}. \tag{2.37}$$

As the next step we consider the *second law of thermodynamics* (entropy inequality) in the form:

$$\Gamma(t) \equiv \int_{\mathscr{B}_\mathbf{X}}\varrho_\mathbf{X}\partial_t\eta\,\mathrm{d}V \geq \int_{\mathscr{B}_\mathbf{X}}\theta^{-1} r_h\,\mathrm{d}V + \int_{\mathscr{B}_\mathbf{X}}\mathrm{Div}\left(\theta^{-1}\mathbf{q}_\mathbf{X}\right)\,\mathrm{d}V,$$

where η is the *specific entropy*, and θ is the absolute temperature. An alternative form of entropy inequality is that expressed in terms of the thermodynamic potential per unit mass, which is known as *Helmholtz free energy*.

$$\Psi(\mathbf{F}) = \varrho_\mathbf{X}\psi, \quad \psi = e - \theta\eta.$$

Then we can write

$$\begin{aligned}\partial_t\Psi \equiv \frac{\mathrm{D}(\varrho_{\mathbf{X}}\psi)}{\mathrm{D}t} = \psi\underbrace{\partial_t\varrho_{\mathbf{X}}}_{=0} + \varrho_{\mathbf{X}}\partial_t\psi &= \varrho_{\mathbf{X}}\partial_t\psi \\ \partial_t\psi = \partial_t e - \eta\partial_t\theta - \theta\partial_t\eta \Rightarrow \theta\varrho_{\mathbf{X}}\partial_t\eta &= \varrho_{\mathbf{X}}\partial_t e - \varrho_{\mathbf{X}}\eta\partial_t\theta - \varrho_{\mathbf{X}}\partial_t\psi \\ &= \varrho_{\mathbf{X}}\partial_t e - \varrho_{\mathbf{X}}(\eta\partial_t\theta + \partial_t\psi).\end{aligned}$$

Then by also considering the energy equation in (2.37), we obtain the *Clausius-Duhem inequality* in terms of the Helmholtz free energy

$$\mathbf{P}^T : \partial_t\mathbf{F} - \varrho_{\mathbf{X}}(\eta\partial_t\theta + \partial_t\psi) - \theta^{-1}\mathbf{q}_{\mathbf{X}} \cdot \mathrm{Grad}\ \theta \geq \mathbf{0}.$$

If we take into account that the dissipation of energy is equal to zero in a isothermal, reversible (elastic) process, then from the *Clausius-Planck* inequality

$$\mathcal{R} \equiv \mathbf{P}^T : \partial_t\mathbf{F} - \varrho_{\mathbf{X}}(\eta\partial_t\theta + \partial_t\psi) \geq \mathbf{0} \Rightarrow \boxed{\mathcal{R} \equiv \mathbf{P}^T : \partial_t\mathbf{F} - \underbrace{\varrho_{\mathbf{X}}\partial_t\psi}_{=\partial_t\Psi(\mathbf{F})} = \mathbf{0}}, \tag{2.38}$$

where the term $\partial_t\Psi(\mathbf{F})$ is expressed as follows:

$$\partial_t\Psi(\mathbf{F}) \equiv \varrho_{\mathbf{X}}\partial_t\psi = \mathbf{P}^T : \partial_t\mathbf{F},$$

where $\partial_t\Psi = \partial_{\mathbf{F}}\Psi : \partial_t\mathbf{F}^T$. Next, by substituting the therm of the $\partial_t\Psi$ equation into the internal energy dissipation given by the equation (2.38) we get

$$\mathbf{P}^T : \partial_t\mathbf{F} - \partial_{\mathbf{F}}\Psi : \partial_t\mathbf{F}^T = \mathbf{0} \Rightarrow \mathbf{P}^T : \partial_t\mathbf{F} = \partial_{\mathbf{F}}\Psi : \partial_t\mathbf{F}^T \Rightarrow \boxed{\mathbf{P} = \partial_{\mathbf{F}}\Psi}. \tag{2.39}$$

2.4.3 Simplified Saint Venant–Kirchhoff Material Model

The Saint Venant-Kirchhoff material model is the simplest phenomenologically-motivated compressible material model based on generalized Hooke's Law, which exhibits nonlinear behaviour due to the dependence on non-linear strain measure. This material model possesses well-known limitations, particularly some instabilities when subjected to pure compression (critical compression threshold is reached $\approx 58\%$ of undeformed dimensions, when compression occurs along a single axis).

It is defined by the strain potential density function, which depends on deformation gradient only. Its complexity is in contrast with the Neo-Hookean type of materials, which on the other hand have quite complicated formulations for their compressible type of behaviour. The Saint Venant-Kirchhoff material model is characterized by the energy function given by

$$\Psi = \frac{\lambda}{2}\left(\mathrm{tr}\ \mathbf{E}\right)^2 + \mu\mathbf{E} : \mathbf{E} \text{ or more generally } \Psi = \frac{1}{2\varrho_0}\left(\mathbf{E} : \mathbb{C}_{ijkl} : \mathbf{E}\right),$$

where λ and μ are Lames's material constants, $\mathbb{C}_{ijkl}$ is the Hooke's fourth-order constitutive elastic tensor. Lames's material constants for a general spatial stress problems are defined as follows:

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)},$$

where E is the *Young's modulus* as a measure of stretch resistance and ν is the *Poisson's ratio*, which expresses a measure of incompressibility. Respective Hooke's tensor expressed in its canonical form using the Kronecker-delta δ_{ij} is as follows:

$$\mathbb{C}_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right).$$

The more frequent form of expressing the constitutive Hooke's tensor is then in its relation to the Helmholtz's free energy

$$\boxed{\mathbb{C}_{ijkl} \equiv \partial^2_{\mathbf{E}}\Psi = \lambda(\mathbf{I}\otimes\mathbf{I}) + 2\mu\mathbb{I}_{ijkl}}. \tag{2.40}$$

The first Piola-Kirchhoff stress is a function of the deformation gradient and it is directly related to Helmholtz free energy via formulas as follows (see terms 2.33 and 2.39):

$$\begin{aligned}
\delta\Psi &= \lambda \mathrm{tr}\ \mathbf{E}\mathrm{tr}\ \delta\mathbf{E} + 2\mu\mathbf{E} : \delta\mathbf{E} = \underbrace{\mathbf{F}\left(\lambda \mathrm{tr}\ \mathbf{E}\mathbf{I} + 2\mu\mathbf{E}\right)}_{=\partial_{\mathbf{F}}\Psi} : \delta\mathbf{F} \\
\mathbf{P} &= \partial_{\mathbf{F}}\Psi \Rightarrow \frac{\partial\Psi(\mathbf{E}\circ\mathbf{F})}{\partial\mathbf{F}} \equiv \mathbf{F}\left(\lambda \mathrm{tr}\ \mathbf{E}\mathbf{I} + 2\mu\mathbf{E}\right)
\end{aligned}$$

The second Piola-Kirchhoff stress tensor can be obtained by

$$\mathbf{S} = \mathbf{F}^{-1}\cdot\mathbf{P} \equiv \partial_{\mathbf{E}}\Psi \Rightarrow \frac{\partial\Psi(\mathbf{E}\circ\mathbf{F})}{\partial\mathbf{E}} = \boxed{\mathbb{C}_{ijkl} : \mathbf{E}}. \tag{2.41}$$

The linearization of the Green-Lagrange strain tensor $\mathbf{E}$ in Saint Venant-Kirchhoff material model provides a *linear elastic* material model, which is used in the thesis

$$\boxed{\Psi(\mathrm{lin}\ \mathbf{E}) = \frac{\lambda}{2}\left(\mathrm{tr}\ \boldsymbol{\varepsilon}\right)^2 + \mu\boldsymbol{\varepsilon} : \boldsymbol{\varepsilon}} \Rightarrow \boldsymbol{\sigma} \equiv \varrho\partial_{\boldsymbol{\varepsilon}}\Psi(\boldsymbol{\varepsilon}) = \mathbb{C}_{ijkl} : \boldsymbol{\varepsilon}, \tag{2.42}$$

where $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$ denote Cauchy's stress tensor and linear (engineering) strain tensor, respectively. The Saint Venant-Kirchhoff material model can be viewed as an improvement of the linear elasticity model, where infinitesimal strain tensor $\boldsymbol{\varepsilon}$ is replaced by rotationally invariant Green-Lagrange strain tensor $\mathbf{E}$.

2.5 Nonlinear Boundary Conditions

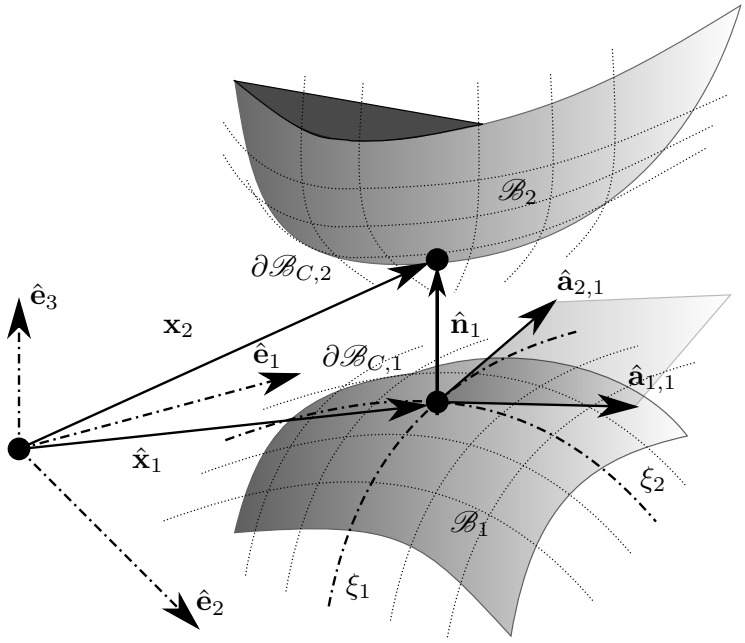


Figure 22: The Normal type of contact between $\mathscr{B}_1$ (master body) and $\mathscr{B}_2$ (slave body).

Consideration of a nonlinear boundary conditions represented by their change caused by the contact of surface portion $\partial\mathscr{B}_C$ of the body $\mathscr{B}$ with neighboring barrier or with the other body and optionally also by contact body $\mathscr{B}$ with itself. It is a critical area in dynamic simulations. The contact causes the formation of additional forces, which at the time of their creation are suitably applied into the process of numerical integration of the equations of motion. Here, the problem of general contact is considered, especially for the purposes of collision with the barrier and subsequent post-critical behavior of the respective structure.

Methodologies typically utilized for contact interactions are relatively immature in comparison to other components of a nonlinear finite element package, such as large deformation kinematics, inelastic material modeling, nonlinear equation solving, or linear solver technology. It is the field of *contact/impact* mechanics or *Computational Contact Mechanics* (CCM), which is a critical part of the safety analysis of vehicles or aircrafts. Car, aircraft impact or steel connections require consideration of inelastic constitutive equations and sometimes also finite deformations.

In the case of car safety it then leads to the optimization of respective vehicle structure, which has an effect to lower transmitted forces on the occupant and the design of a less aggressive restraint system. In civil engineering it also contains the phenomenas of the car or aircrafts impact against building structures. This is particularly relevant for buildings of higher importance, such as nuclear power plants, etc.

Computational contact mechanics encompass such topics as tribological complexity, thermomechanical coupling on interfaces, energy-momentum treatment of transient impact events, and new techniques for spatial discretization of contact phenomena.

The first mathematical analysis of the problem under the consideration was carried out by Leonard Paul Euler, who assumed triangular section asperities for the representation of surface roughness in his articles *Sur la diminution de la resistance du frottement* and *Sur le frottement des corps solides* from 1748. As the first one consideration of the contact in solid mechanics, the work of Hertz in 1882 (see [43]) can be considered. He applied the elasticity theory in the contact mechanics, examining the contact of two spheres. On this basis he was able to derive the pressure distribution in the contact area.

Historically the very important problem for the development of CCM, is the problem of contact with rigid foundation, the so-called Signorini's problem from 1933 (see [116]), where it considers node on the boundary with condition prohibiting penetration the rigid obstacle.

Consider two points $\mathbf{X}_1$ and $\mathbf{X}_2$, in the initial configuration of the elastic bodies $\mathscr{B}_1 \subset \mathbb{E}^3$ and $\mathscr{B}_2 \subset \mathbb{E}^3$ occupying bounded domains, which are distinct and occupy the same position in the current configuration $\phi_{\mathscr{B}_1}(\mathbf{X}_1, t) = \phi_{\mathscr{B}_2}(\mathbf{X}_2, t)$, where contact comes into play on the boundaries $\partial\mathscr{B}_{C,1}$ and $\partial\mathscr{B}_{C,2}$.

Due to the applied contact conditions in the thesis, consider normal contact without friction. For such two bodies in contact, the non-penetration condition is given by

$$(\mathbf{x}_2 - \mathbf{x}_1) \cdot \mathbf{n}_1 \geq 0,$$

where $\mathbf{x}_\alpha, \alpha = 1, 2$ denotes the coordinates of the current configuration $\phi_{\mathscr{B}_\alpha}$ of body $\mathscr{B}_\alpha$: $\mathbf{x}_\alpha = \mathbf{X}_\alpha + \mathbf{u}_\alpha$, where $\mathbf{u}_\alpha$ denotes displacement field, $\mathbf{n}_1$ is the normal vector associated with body $\mathscr{B}_1$. By assuming that the contact boundary locally describes a convex region, then relate every point $\mathbf{x}_2 \in \partial\mathscr{B}_{C,2}$ to a point $\hat{\mathbf{x}}_1 = \mathbf{x}_1(\boldsymbol{\xi}) \in \partial\mathscr{B}_{C,1}$ via the minimum distance problem for a usage of defining the gap or penetration between the two bodies

$$\hat{\Delta}(\xi_1, \xi_2) = \|\mathbf{x}_2 - \hat{\mathbf{x}}_1\| = \min_{\mathbf{x}_1 \subseteq \partial\mathscr{B}_{C,1}} \|\mathbf{x}_2 - \mathbf{x}_1(\boldsymbol{\xi})\|,$$

where $\boldsymbol{\xi} = (\xi_1, \xi_2)$ denotes the parameterization of the boundary $\partial\mathscr{B}_{C,1}$ via convective coordinates. The point $\overline{\mathbf{x}}_1$ is computed from the necessary condition for the minimum of the distance function

$$\partial_{\xi_\alpha}\hat{\Delta}(\xi_1, \xi_2) = \frac{\mathbf{x}_2 - \mathbf{x}_1(\xi_1, \xi_2)}{\|\mathbf{x}_2 - \mathbf{x}_1(\xi_1, \xi_2)\|} \cdot \partial_{\xi_\alpha}\mathbf{x}_1(\xi_1, \xi_2) = 0,$$

where orthogonality of the first and second terms is required, the first term must have the same direction as the normal vector $\mathbf{n}_1$ at the minimum point, $\partial_{\xi_\alpha}\mathbf{x}_1(\xi_1, \xi_2)$ denotes the tangent vector $\mathbf{a}_{\alpha,1}$. Then the orthogonal projection of a given slave point $\mathbf{x}_2$ onto the current master surface $\phi_{\mathscr{B}_1}(\partial\mathscr{B}_{C,1}, t)$ is $-\mathbf{n}_1(\xi_1, \xi_2) \cdot \mathbf{a}_{\alpha,1}(\xi_1, \xi_2) = 0$. Thus the outward unit normal on the current master surface at the master point has form as follows:

$$\hat{\mathbf{n}}_1 = \frac{\hat{\mathbf{a}}_{1,1} \times \hat{\mathbf{a}}_{2,1}}{\|\hat{\mathbf{a}}_{1,1} \times \hat{\mathbf{a}}_{2,1}\|}.$$

It can only be used in relation with the penalty method, which is here in the main focus. Once the point $\hat{\mathbf{x}}_1$ is known, the inequality constraint of the non-penetration condition can be defined as

$$g_N = (\mathbf{x}_2 - \hat{\mathbf{x}}_1) \cdot \hat{\mathbf{n}}_1 \geq 0, \tag{2.43}$$

or similarly penetration condition which is important for penalty method has a form:

$$g_{N,\epsilon} = \begin{cases} (\mathbf{x}_2 - \hat{\mathbf{x}}_1) \cdot \hat{\mathbf{n}}_1 & \text{if } (\mathbf{x}_2 - \hat{\mathbf{x}}_1) \cdot \hat{\mathbf{n}}_1 < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.44}$$

For the variation of the normal gap it is necesary to take into account the projection of point $\mathbf{x}_2$ onto the master surface parameterized by the convective coordinates, which leads to the following expression:

$$\begin{aligned} \delta g_N &= [\delta\mathbf{x}_2 - \delta\mathbf{x}_1 - \partial_{\xi_\alpha}\hat{\mathbf{x}}_1 \delta\xi_\alpha] \cdot \hat{\mathbf{n}}_1 + [\mathbf{x}_2 - \hat{\mathbf{x}}_1] \cdot \delta\hat{\mathbf{n}}_1, \text{ where } \partial_{\xi_\alpha}\hat{\mathbf{x}}_1 \cdot \hat{\mathbf{n}}_1 = \hat{\mathbf{n}}_1 \cdot \delta\hat{\mathbf{n}}_1 = 0 \\ \delta g_N &= [\delta\mathbf{x}_2 - \delta\mathbf{x}_1] \cdot \hat{\mathbf{n}}_1. \end{aligned}$$

The normal contact pressure cannot be computed from a constitutive equation, but is then obtained as a reaction in the contact area, and hence can be deduced from the constraint equations as a classical way to formulate contact constraints. The condition for non-penetration is stated as $g_N \geq 0$, which precludes the penetration of body $\mathscr{B}_1$ into body $\mathscr{B}_2$. Contact takes place when $g_N = 0$. This leads to conditions providing the basis to treat frictionless contact problems in the context of constraint optimization known as Hertz–Signorini–Moreau or Kuhn–Tucker–Karush condition

$$g_N \geq 0, \quad p_N \leq 0, \quad p_N g_N = 0, \tag{2.45}$$

where p_N is the the associated normal component of the stress traction vector $\mathbf{t} = \boldsymbol{\sigma} \cdot \mathbf{n}$ in current configuration or $\mathbf{T} = \mathbf{P} \cdot \mathbf{N}$ in reference configuration. The traction vector $\mathbf{t}$ at boundary $\partial\mathscr{B}_C$ is usually decomposed into *contact pressure* and *tangential traction* components, respectively. Thus, it leads to the term as follows:

$$\mathbf{t} \to \mathbf{t}_{C,1} = p_N \hat{\mathbf{n}}_1 + \mathbf{t}_\tau = p_N \hat{\mathbf{n}}_1 \underbrace{+ t_\tau^{\xi_1} \boldsymbol{\tau}_{C,1}^{\xi_1} + t_\tau^{\xi_2} \boldsymbol{\tau}_{C,1}^{\xi_2}}_{\text{tangential traction components are not considered here}} .$$

In order to obtain a contact pressure, it is necessary to include a constitutive equation that appropriately represents the micromechanical behaviour on the contact surface. The micromechanical behaviour depends in general upon material parameters like hardness and also on geometrical parameters like surface roughness. The most commonly used constitutive equation for a normal contact (see [132]) includes the following term:

$$p_N = c_N d^n = c_N \left(\zeta - g_N\right)^{\frac{-1}{n}},$$

where d denotes the distance function defined by equation 2.43 and 2.44, ζ is the initial mean plane distance in the contact area, then c_N and n are constitutive parameters determinated by experiments. Setting of parametrs $\zeta = 0$ and $n = -1$ leads to a standard penalty method $p_N = -c_N g_N$, which is considered in the thesis.

In the view to the definition of the contact constitutive equations and interface modeling, it is worth mentioning the work of P. Gruber and J. Zeman (see [137], [65]), focused on application the FETI method for such a subject. Although this work is rather directed to the field of fracture mechanics, it is even closely related to the field of contact mechanics.

2.6 Summary of Chapter

In this chapter the mathematical-physical model of a continua, with which will be worked with in the implementation of parallel computations was briefly introduced. Discussed will be equations of motion of a flexible body which undergoes small deformation but large rotations due to the consideration of the convected coordinates for the nonlinear transient FEA of shell structures. In the context with the geometrically nonlinear behavior of a solid continua represented by a large rotational kinematics, nonlinear contact conditions are also involved for the modeling the contact/impact phenomena represented by the frictionless normal contact. Other nonlinearities will not be further considered in the thesis.

The Saint Venant-Kirchhoff material model is only described here primarily due to the generality in its close relation to the small strain linear elasticity as linearized form. Small strain linear elastic material model is further applied in numerical models.

Chapter 3

Mathematical Modeling

The problem being studied of dynamics in general terms of Continuum Mechanics, which generally contains various possible types of nonlinear behavior, generally leads to the IBVP as mentioned in the preceeding chapter. Solving such problems therefore requires advanced tools of mathematical analysis.

First and foremost, it is the study of certain topological-algebraic structures and of the methods by which knowledge of these structures can be applied to analytical problems. It deals with large classes of objects such as a function, a measure or operator. Most of the interesting classes that occur in this way turn out to be vector spaces which are supplied with metrics, or at least with topologies, that bear some natural relation to the objects of which the spaces are made up. The simplest and most importatnt way of doing this is to introduce a norm. The resulting structure is called a normed vector space, or a normed linear space, or simply a normed space.

This analysis is necessary especially with regard to the numerical solution of IBVP. These are mainly the finite element method, which is directly related to the issue of the relationship between a strong solution in terms of a general analytical solution of a partial differential equations and their solution by a so-called weak formulation. It provides a formalism for generating discrete algorithms for approximating the solutions of differential equations. It relates to the concept of generalized functions and the definition of derivative in the sense of distributions and related topics belonging to the branch of Banach algebra.

In mathematical physics, this type of analysis relates mainly to finding solutions to some of the functional as its stationary point. This theme is thus closely tied to the various methods of calculus of variations.

It concerns finding extremes generally nonlinear functionals based on the local linearity of functions. Functional must satisfy a condition that is weakly lower semicontinuous. This property plays a fundamental role in calculus of variations. It is derived from the functional coercivity and convexity of the integrand.

3.1 Brief Introduction to Mathematical Theory of Variational Calculus

The use of variational calculus for the purposes of mechanics is inextricably linked to several important topics in mathematical analysis, namely to non-linear functional analysis. In short, it refers to generalized normed vector spaces, functionals, and for vector valued functions there are two main version of derivatives: Gâteaux (or weak) derivatives and Fréchet (or strong) derivatives.

The Gâteaux derivative is further used to derive the necessary conditions for a minimum of the given functional, which is a generalization of the directional derivative from differential calculus, where teh Gâteaux derivative depends on an arbitrary function $\boldsymbol{\eta}$. In contrast, the gradient (i.e., the derivative) of an ordinary function does not have such an arbitrary entity.

Definition 3.1.1. A function $\varphi : \mathbb{R}^n \to \mathbb{R}$ is said to be *locally Lipschitz continuous* if $\forall x \in \mathbb{R}^n$ there is a neighborhood $X \subset \mathbb{R}^n$ of such x, and there is a *Lipschitz constant* $L_f \geq 0$ such that $\|\varphi(x_1) - \varphi(x_2)\| \leq L_f \|x_1 - x_2\|$, $\forall x_1, x_2 \in X$.

Proposition 3.1.2. Let X be a Banach space. Suppose that X has a simple Lipschitz local approximate Fréchet subdifferential sum rule. Then X is Asplund.

It provides the easy direction which essentially says that if an approximate local sum rule holds in X then X is an Asplund space. The proof of proposition 3.1.2 can be found in [15].

Theorem 3.1.3. Every real-valued Lipschitz function on an Asplund space has points of Fréchet differentiability.

Let X, Y be normed linear spaces. The Fréchet derivative of an operator $F : X \to Y$ is the bounded linear operator $\mathrm{D}_{\mathcal{F}} F(x_0) : X \to Y$ which satisfies the following relation:

$$\lim_{h \to 0} \frac{\|F(x_0 + h) - F(x_0) - \langle \mathrm{D}_{\mathcal{F}} F(x_0); h \rangle\|}{\|h\|} = 0, \tag{3.1}$$

where $\mathrm{D}_{\mathcal{F}}$ is called the Fréchet differential. The Fréchet derivative, as defined in (3.1) extends concepts of the derivative to operators in general normed spaces, for example, infinite-dimensional function spaces. To establish the relationship to the Gâteaux differential, take $h = \epsilon \boldsymbol{\eta}$. This is of great importance to computational methods for solving nonlinear operator equations.

Definition 3.1.4. Let X be a Banach space and A be a subset of X (X^*). We say that A is dentable provided for any $\varepsilon > 0$ there exists $x^* \in X^*$ ($x \in X$) and $\alpha > 0$ such that $\dim S(x^*, A, \alpha) < \varepsilon$ ($\dim S(x, A, \alpha) < \varepsilon$).

Definition 3.1.5. Let X be a Banach space and A be a subset of X. We say A has the *Radon-Nikodym property* if every nonempty bounded subset of A is *dentable*.

Theorem 3.1.6. Every Lipschitz map from a separable Banach space X into a space Y with the *Radon-Nikodym property* is Gâteaux differentiable almost everywhere (except for a null set).

Consider a function $\Phi : X \to Y$, where $X, Y \in \mathbb{R}^n$. It maps a Banach space X into a Banach space Y. The Gâteaux differential $\mathcal{G}\Phi(x(t))(\boldsymbol{\eta}(t))$ of Φ at $x(t) \in X$ in the direction $\boldsymbol{\eta}(t) \in X$ is

$$\begin{aligned}\mathcal{G}\Phi(x(t))(\boldsymbol{\eta}(t)) \equiv \lim_{\epsilon\to 0}\frac{\Phi(x(t)+\epsilon\boldsymbol{\eta}(t),t)-\Phi(x(t),t)}{\epsilon} &= \frac{\mathrm{d}}{\mathrm{d}\epsilon}\Phi(x(t)+\epsilon\boldsymbol{\eta}(t),t)|_{\epsilon=0}\\ &= \mathrm{D}_{\mathcal{F}}\Phi(x(t),t)\cdot\boldsymbol{\eta}(t)\\ &= \langle \mathrm{D}_{\mathcal{F}}\Phi(x(t),t);\boldsymbol{\eta}(t)\rangle.\end{aligned}$$

where $\epsilon \in \mathbb{R}$.

If the limit exists for all $\boldsymbol{\eta} \in X$ then Φ is called Gâteaux differentiable at the point x. Gâteaux differentiability does not imply Fréchet differentiability. For the function $\Phi : \mathbb{R}^n \to \mathbb{R}^m, n$ and $m \in \mathbb{N}$, the Fréchet derivative $\mathrm{D}_{\mathcal{F}}\Phi(x_0)$ is the *Jacobian* of Φ, a linear operator which is represented by an $m \times n$ matrix as showen bellow

$$\mathrm{D}_{\mathcal{F}}\Phi(x_0) = \begin{bmatrix}\partial_{x_1(t)}\Phi_1(x_0) & \cdots & \partial_{x_n(t)}\Phi_1(x_0)\\ \cdots & \cdots & \cdots\\ \partial_{x_1(t)}\Phi_m(x_0) & \cdots & \partial_{x_n(t)}\Phi_m(x_0)\end{bmatrix}.$$

Theorem 3.1.7. The Fréchet derivative exists at $x = x_0$ if all Gâteaux differentials are continuous functions of x at $x = x_0$.

The Gâteaux differential defines a map $\mathcal{G}\Phi(x(t)) : X \to Y$, which assigns to each element $\boldsymbol{\eta} \in X$ a value $\mathcal{G}\Phi(x(t))(\boldsymbol{\eta}(t)) \in Y$. Thus, the x in $\mathcal{G}\Phi(x(t))$ does not refer to an independent variable, rather it is a part of the name of the map. The differential is linear, i.e $\mathcal{G}\Phi(x(t))(\alpha\boldsymbol{\eta}(t)) = \alpha\mathcal{G}\Phi(x(t))(\boldsymbol{\eta}(t))$ and $\mathcal{G}\Phi(x(t))(\boldsymbol{\eta}_1(t)+\boldsymbol{\eta}_2(t)) = \mathcal{G}\Phi(x(t))(\boldsymbol{\eta}_1(t)) + \mathcal{G}\Phi(x(t))(\boldsymbol{\eta}_2(t))$, differential of a constant is zero $\mathcal{G}c = 0$.

Furthermore, for a given $x_0 \in X$ a function $\Phi_{x_0} : X \to Y$

$$\Phi_{x_0}(x(t)) = \Phi(x_0) + \mathcal{G}\Phi(x_0)(x(t) - x_0),$$

which is a linear approximation to Φ in the point x_0.

Theorem 3.1.8. Every continuous real-valued function on a compact set attains its extreme values on that set.

Definition 3.1.9. Let X be a normed vector space. The (continuous) *dual space* X^* is defined to be $X^* : X \to \mathbb{R}$ if X is a real vector space. Elements of X^* are known as continuous linear functionals (or bounded linear functionals) on X.

Let $\Omega \subset \mathbb{R}^n$ be an open and bounded set. We consider the integral expressions as follows:

$$\mathfrak{J}(\Phi) \equiv \int_\Omega \boldsymbol{\mathcal{L}}(x, \Phi(x), \mathrm{D}_{\mathcal{F}}\Phi(x))\ \mathrm{d}x, \tag{3.2}$$

where we assume $\Phi \in C^1(\overline{\Omega})$ and $\mathrm{D}_{\mathcal{F}}\Phi(x) = (\partial_{x_1}\Phi, \ldots, \partial_{x_n}\Phi)$ denotes the gradient of Φ. Here $\boldsymbol{\mathcal{L}} : \overline{\Omega} \times \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$, $(x, \Phi, \phi) \mapsto \boldsymbol{\mathcal{L}}(x, \Phi, \phi)$ is some given function. The function $\boldsymbol{\mathcal{L}}$ is often called *Lagrangian*, especially in association with the physics. Associated functional map $\mathfrak{J}$ is called *action functional*.

The main interest is in finding functions $\Phi(x)$ that minimize among set $\mathscr{C} \subset C^1(\overline{\Omega})$ of admissible functions. Let us process more precise definition:

- A subset $\mathscr{C} \subset C^1(\overline{\Omega})$ is called admissible if for every $\Phi \in \mathscr{C}$ and $\boldsymbol{\eta} \in C^\infty(\Omega)$ there is some $e > 0$ such that $\Phi + \epsilon\boldsymbol{\eta} \in \mathscr{C}$ for all $\epsilon \in (-e, e)$.
- Let $\mathscr{C} \subset C^1(\overline{\Omega})$ be admissible. Suppose that $\Phi \in \mathscr{C}$ is a (global) minimizer of $\mathfrak{J}$ in $\mathscr{C}$ if $\mathfrak{J}(\Phi) \leq \mathfrak{J}(\Phi')$ for all $\Phi' \in \mathscr{C}$. Then suppose that $\Phi \in \mathscr{C}$ is local minimizer of $\mathfrak{J}$ in $\mathscr{C}$ if there is some δ such that $\mathfrak{J}(\Phi) \leq \mathfrak{J}(\Phi')$ for all $\Phi' \in \mathscr{C}$ with $\|\Phi - \Phi'\|_{C^1(\Omega)} \leq \delta$. A global minimizer Φ is automatically a local minimizer, but not vice versa in general.

The main problem of the calculus of variations consists of the task to minimize $\mathfrak{J}(\Phi)$ among all admissible functions $\Phi \in \mathscr{C}$. The main difficulty is the fact that we try to minimize a functional $\mathfrak{J}(\Phi)$, and not just a function $f : \Phi \subset \mathbb{R}^n \to \mathbb{R}$ as done in multi-variable calculus. The set of admissible function $\mathscr{C} \subset C^1(\overline{\Omega})$ belongs to an *infinite-dimensional Banach space*. Bounded and closed set $X \subset C^1(\overline{\Omega})$ are not necessarily compact. This complicates the analysis substantially and entirely new methods (weak convergence, refxivity, lower semicontinuity, Sobolev spaces etc.) have to be developed.

As a first step towards the minimization problem for $\mathfrak{J}(\Phi)$, let us introduce the notion of a derivative for functionals as follows:

Definition 3.1.10. Let $\boldsymbol{\mathcal{L}}$ and Φ be as above. For $\boldsymbol{\eta} \in C^\infty(\Omega)$, define

$$\delta\mathfrak{J}(\Phi)(\boldsymbol{\eta}) \equiv \frac{\mathrm{d}}{\mathrm{d}\epsilon}\mathfrak{J}(\Phi + \epsilon\boldsymbol{\eta})\bigg|_{\epsilon=0} \tag{3.3}$$

to be the *first variation* of $\mathfrak{J}(\Phi)$ in the direction given by $\boldsymbol{\eta} \in C^\infty(\Omega)$, provided that the derivative above exists.

The term $\delta\mathfrak{J}(\Phi)$ generalizes the notion of the *directional derivative*. We suppose that the space of so-called *test functions* $\boldsymbol{\eta} \in C^\infty(\Omega)$ is large enough to be capable to use the fundamental lemma of calculus. Let us derive the necessary condition for local minimizers (the *Principle of Stationary Action*):

Theorem 3.1.11. Let F be a map $F : X \to X^*$, where X and X^* are Banach spaces, and $\mathscr{C} \subset C^1(\overline{\Omega})$ admissible. If $\Phi \in \mathscr{C}$ is a local minimizer of $\mathfrak{J}(\Phi)$ in $\mathscr{C}$, then

$$\delta\mathfrak{J}(\Phi)(\boldsymbol{\eta}) = 0 \text{ for all } \boldsymbol{\eta} \in C^\infty(\Omega), \tag{3.4}$$

provided the first variation $\delta\mathfrak{J}$ exists.

The disappearance of the first variation $\delta\mathfrak{J}(\Phi)(\boldsymbol{\eta}) = 0$ for all $\boldsymbol{\eta} \in C^\infty(\Omega)$ is only a necessary condition for Φ to be a local minimizer. If the respective condition is satisfied, the function Φ is (weak) critical point or point of stationary action of $\mathfrak{J}(\Phi)$ in $\mathscr{C}$. There might be stationary points that are neither local minimizers or local minimizers (saddle point).

Proposition 3.1.12. Let $\boldsymbol{\mathcal{L}} \in C^1(\overline{\Omega} \times \mathbb{R} \times \mathbb{R}^n)$ and $\Phi \in C^1(\overline{\Omega})$. Then the first variation of $\mathfrak{J}(\Phi)$ exists for any $\boldsymbol{\eta} \in C^\infty(\Omega)$ and it is given by the integral expression

$$\delta\mathfrak{J}(\Phi)(\boldsymbol{\eta}) = \int_\Omega \Big(\partial_\Phi \boldsymbol{\mathcal{L}}(x, \Phi(x), \mathrm{D}_{\mathcal{F}}\Phi(x))\boldsymbol{\eta}(x) + \partial_{\mathrm{D}_{\mathcal{F}}\Phi}\boldsymbol{\mathcal{L}}(x, \Phi(x), \mathrm{D}_{\mathcal{F}}\Phi(x)) \cdot \mathrm{D}_{\mathcal{F}}\boldsymbol{\eta}(x)\Big)\ \mathrm{d}x.$$

By combination of the Principle of Stationar Action and the proposition above, then we obtain

Theorem 3.1.13. Let $\boldsymbol{\mathcal{L}} \in C^1(\overline{\Omega} \times \mathbb{R} \times \mathbb{R}^n)$ and $\mathscr{C} \subset C^1(\overline{\Omega})$ be an admissable set. If $\Phi \in \mathscr{C}$ is local minimizer of $\mathfrak{J}(\Phi)$ in $\mathscr{C}$, then Φ satisfies the *weak Euler-Lagange equation* given by

$$\int_\Omega \Big(\partial_\Phi \boldsymbol{\mathcal{L}}(x, \Phi(x), \mathrm{D}_{\mathcal{F}}\Phi(x))\boldsymbol{\eta}(x) + \partial_{\mathrm{D}_{\mathcal{F}}\Phi}\boldsymbol{\mathcal{L}}(x, \Phi(x), \mathrm{D}_{\mathcal{F}}\Phi(x)) \cdot \mathrm{D}_{\mathcal{F}}\boldsymbol{\eta}(x)\Big)\ \mathrm{d}x = 0$$

for all $\boldsymbol{\eta} \in C^\infty(\Omega)$.

Now let us state the *Fundamental Lemma of Calculus of Variations*

Lemma 3.1.14. If $f \in C(\Omega)$ satisfies

$$\int_\Omega \Phi(x)\boldsymbol{\eta}(x)\ \mathrm{d}x = 0 \text{ for all } \boldsymbol{\eta} \in C^\infty(\Omega),$$

then $\Phi \equiv 0$ on Ω.

Theorem 3.1.15. Let $\boldsymbol{\mathcal{L}} \in C^2(\overline{\Omega} \times \mathbb{R} \times \mathbb{R}^n)$ and $\Phi \in C^1(\overline{\Omega})$ be an admissable set. If $\Phi \in \mathscr{C}$ is local minimizer of $\mathfrak{J}(\Phi)$ in $\mathscr{C}$ and $\Phi \in C^2(\Omega)$, then Φ solves the *classical Euler-Lagrange equation* given by

$$\boxed{\partial_\Phi \boldsymbol{\mathcal{L}}(x, \Phi(x), \mathrm{D}_{\mathcal{F}}\Phi(x)) + \frac{\mathrm{d}}{\mathrm{d}x}\partial_{\mathrm{D}_{\mathcal{F}}\Phi}\boldsymbol{\mathcal{L}}(x, \Phi(x), \mathrm{D}_{\mathcal{F}}\Phi(x)) = 0}$$

for all $x \in \Omega$.

3.2 Variational Formulation of an Inertial Problem

To obtain equations of motion in mechanics of continua the same variational principles are used as in the case of discrete models. It is about their generalizations to an infinite dimensional space.

First it is necessary to find the first variation of the relevant functional $\delta\mathfrak{J}(\mathbf{q}(t))$

$$\delta\mathfrak{J}(\mathbf{q}(t),t) \equiv (\mathrm{D}_{\mathcal{F}}\mathfrak{J}(\mathbf{q}(t),t);\boldsymbol{\eta}(t)) = \mathcal{G}\mathfrak{J}(\mathbf{q}(t),t)(\boldsymbol{\eta}(t)) = \frac{\mathrm{d}}{\mathrm{d}\epsilon}\mathfrak{J}(\mathbf{q}(t)+\epsilon\boldsymbol{\eta}(t),t)|_{\epsilon=0},$$

where $\epsilon\boldsymbol{\eta}(t)$ denotes a variation of the function $\mathbf{q}(t)$ and resulting functional gradient is the first variation of $\mathfrak{J}(\mathbf{q}(t),t)$, where the inner product is the standard L^2 inner product for real functions in Hilbert spaces, $(f;g) = \int f(x)g(x)\,\mathrm{d}x$.

Theorem 3.2.1. Let $\mathbf{q}(t) : [t_0,t_1] \to \mathbb{R}^n$ be a function of class $C^2[a,b]$ satisfying the boundary conditions $\mathbf{q}(t_0) = \overline{\mathbf{q}}_1, \mathbf{q}(t_1) = \overline{\mathbf{q}}_1$. Then the first variation of the functional $\mathfrak{J}(\mathbf{q},t)$ is given by the functional gradient, and leads to the form of Euler-Lagrange equations.

$$\boxed{\delta\mathfrak{J}(\mathbf{q}(t),t) \equiv \partial_{\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t) - \frac{\mathrm{d}}{\mathrm{d}t}\left(\partial_{\partial_t\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t)\right)},$$

where $\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t) \in C^2[a,b]$ is the associated Lagrangian.

Proof. Let $\boldsymbol{\eta}(t) \in C^2[a,b]$ be a variation of $\mathbf{q}(t)$. Then $\boldsymbol{\eta}(t) : [t_0,t_1] \to \mathbb{R}^n$ satisfies the boundary conditions $\overline{\mathbf{q}}_0 = \mathbf{q}(t_0) + \epsilon\boldsymbol{\eta}(t_0)$ and $\overline{\mathbf{q}}_1 = \mathbf{q}(t_1) + \epsilon\boldsymbol{\eta}(t_1)$. By definition, the first variation of $\mathfrak{J}(\mathbf{q}(t),t)$ is given by the inner product

$$\begin{aligned}
\delta\mathfrak{J}(\mathbf{q}(t),t) \equiv (\mathrm{D}_{\mathcal{F}}\mathfrak{J}(\mathbf{q}(t),t);\boldsymbol{\eta}(t)) &= \frac{\mathrm{d}}{\mathrm{d}t}\mathfrak{J}(\mathbf{q}(t)+\epsilon\boldsymbol{\eta}(t),t)\bigg|_{\epsilon=0} \\
&= \frac{\mathrm{d}}{\mathrm{d}\epsilon}\left(\int_{t_0}^{t_1}\boldsymbol{\mathcal{L}}(\mathbf{q}(t)+\epsilon\boldsymbol{\eta}(t),\partial_t\mathbf{q}(t)+\epsilon\partial_t\boldsymbol{\eta}(t),t)\,\mathrm{d}t\right)\bigg|_{\epsilon=0} \\
&= \int_{t_0}^{t_1}\frac{\mathrm{d}}{\mathrm{d}\epsilon}\boldsymbol{\mathcal{L}}(\mathbf{q}(t)+\epsilon\boldsymbol{\eta}(t),\partial_t\mathbf{q}(t)+\epsilon\partial_t\boldsymbol{\eta}(t),t)\bigg|_{\epsilon=0}\mathrm{d}t
\end{aligned}$$

for $\epsilon \in \mathbb{R}$. Applying the chain rule and evaluating at $\epsilon = 0$ it yields

$$\begin{aligned}
(\mathrm{D}_{\mathcal{F}}\mathfrak{J}(\mathbf{q}(t),t);\boldsymbol{\eta}(t)) &= \int_{t_0}^{t_1}\Big(\boldsymbol{\eta}(t)\partial_{\mathbf{q}+\epsilon\boldsymbol{\eta}}\boldsymbol{\mathcal{L}}(\mathbf{q}(t)+\epsilon\boldsymbol{\eta}(t),\partial_t\mathbf{q}(t)+\epsilon\partial_t\boldsymbol{\eta}(t),t) + \\
&+ \partial_t\boldsymbol{\eta}(t)\partial_{\partial_t\mathbf{q}+\epsilon\partial_t\boldsymbol{\eta}}\boldsymbol{\mathcal{L}}(\mathbf{q}(t)+\epsilon\boldsymbol{\eta}(t),\partial_t\mathbf{q}(t)+\epsilon\partial_t\boldsymbol{\eta}(t),t)\Big)\bigg|_{\epsilon=0}\mathrm{d}t \\
&= \int_{t_0}^{t_1}\Big(\boldsymbol{\eta}(t)\partial_{\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q},\partial_t\mathbf{q},t) + \partial_t\boldsymbol{\eta}(t)\partial_{\partial_t\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q},\partial_t\mathbf{q},t)\Big)\,\mathrm{d}t,
\end{aligned}$$

where

$$\begin{aligned}
\int_{t_0}^{t_1}\partial_t\boldsymbol{\eta}(t)\partial_{\partial_t\mathbf{q}(t)}\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t)\,\mathrm{d}t &= \Big(\boldsymbol{\eta}(t)\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t)\Big)\bigg|_{t_0}^{t_1} - \\
&- \int_{t_0}^{t_1}\boldsymbol{\eta}(t)\frac{\mathrm{d}}{\mathrm{d}t}\Big(\partial_{\partial_t\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t)\Big)\,\mathrm{d}t.
\end{aligned}$$

In the equations above $\mathbf{q}(t) + \epsilon\boldsymbol{\eta}(t)$ and $\mathbf{q}(t)$ satisfy the same boundary conditions $\boldsymbol{\eta}(t_0) = \boldsymbol{\eta}(t_1) = 0$. Thus respective inner product leads to

$$\begin{aligned}
(\mathrm{D}_{\mathcal{F}}\mathfrak{J}(\mathbf{q}(t),t);\boldsymbol{\eta}(t)) &= \int_{t_0}^{t_1} \left(\boldsymbol{\eta}(t)\partial_{\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q},\partial_t\mathbf{q},t) - \boldsymbol{\eta}(t)\frac{\mathrm{d}}{\mathrm{d}t}\Big(\partial_{\partial_t\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t)\Big)\right) \mathrm{d}t \\
&= \int_{t_0}^{t_1} \boldsymbol{\eta}(t)\left(\partial_{\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q},\partial_t\mathbf{q},t) - \frac{\mathrm{d}}{\mathrm{d}t}\Big(\partial_{\partial_t\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t)\Big)\right) \mathrm{d}t \Rightarrow \\
\Rightarrow \int_{t_0}^{t_1} \boldsymbol{\eta}(t)\mathrm{D}_{\mathcal{F}}\mathfrak{J}(\mathbf{q}(t),t) &= \int_{t_0}^{t_1} \boldsymbol{\eta}(t)\left(\partial_{\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q},\partial_t\mathbf{q},t) - \frac{\mathrm{d}}{\mathrm{d}t}\Big(\partial_{\partial_t\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t)\Big)\right) \mathrm{d}t.
\end{aligned}$$

Respective equality must hold for all choices of $\boldsymbol{\eta}(t)$. Thus we get

$$\delta\mathfrak{J}(\mathbf{q}(t),t) = \partial_{\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q},\partial_t\mathbf{q},t) - \frac{\mathrm{d}}{\mathrm{d}t}\Big(\partial_{\partial_t\mathbf{q}}\boldsymbol{\mathcal{L}}(\mathbf{q}(t),\partial_t\mathbf{q}(t),t)\Big).$$

□

In order to be a critical function, $\mathbf{q}(t)$ must satisfy $\delta\mathfrak{J}(\mathbf{q}(t),t) = 0$. The majority of problems in the calculus of variations involve finding a local minima of functionals, satisfying Euler-Lagrange equations is a condition which will be required for a solution.

Hamilton's Variational Principle for Structural Dynamics

Hamilton's principle and Hamilton's law of varying action in elastodynamics are deduced from an analogy to the Lagrangian form of D'Alembert's principle. The analogy in elastodynamics corresponds to the Bubnov Galerkin weighted residual form, which leads to the principle of virtual work.

It takes a double weighted residual statement in space and time domain. Once discretization in space is carried out in the sense of Galerkin projection, Hamilton's principle or Hamilton's law of varying action naturally results in a weighted residual form in time, and the governing equation in time equals the finite element equation of motion. It can be stated for an arbitrary continuous medium, restricted only by the assumption that it does not exhibit microstructural effects.

Such a variational principle is one of the most widely used variational techniques in the dynamics. His formal derivation can be done from the first law of thermodynamics as its natural consequence. The application of Hamilton's variational principle is extensively addressed by A. Bedford in [4] or J. Har and K. Tamma in [41].

In order to obtain a more general variational form for finite element discretization applicable to a wider group of material models such as viscoelastic or thermoelastic materials, it is necessary to use the postulate of *balance of energy* from the first law of thermodynamics derived in chapter 2.4.2, which says

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\mathscr{B}_{\mathbf{X}}} \varrho_{\mathbf{X}} e \,\mathrm{d}V = \underbrace{\boxed{\int_{\mathscr{B}_{\mathbf{X}}} \mathbf{S} : \dot{\mathbf{E}} \,\mathrm{d}V}}_{\text{deformative power}} + \int_{\mathscr{B}_{\mathbf{X}}} \varrho_{\mathbf{X}} r_h \,\mathrm{d}V - \int_{\mathscr{B}_{\mathbf{X}}} \mathrm{Div}\ \mathbf{q}_{\mathbf{X}} \,\mathrm{d}V,$$

where the last two terms are neglected here. Then Hamilton's variational principle expressed in the form of virtual powers leads to the following expression:

$$\delta \int_{\tau-\Delta\tau}^{\tau} (\mathcal{K}_P + \mathcal{U}_P + \mathcal{U}_{P,C}) \ \mathrm{d}\tau = 0, \tag{3.5}$$

where power functional $\mathcal{K}_P : \mathcal{K}_P \to \mathbb{R}$ represents *kinetic power* and functional $\mathcal{U}_P : \mathcal{U}_P \to \mathbb{R}$ represents powers of an internal (*deformative power*) and external forces, $\mathcal{U}_{P,C} : \mathcal{U}_{P,C} \to \mathbb{R}$ represents constraint contact forces, respectively. Variation of $\mathcal{K}_P$ and $\mathcal{U}_P$ have the following form:

$$\delta\mathcal{K}_P = \int_{\mathscr{B}_\mathbf{X}} \varrho_\mathbf{X} \mathbf{A} \cdot \delta\mathbf{V} \ \mathrm{d}V, \ \delta\mathcal{U}_P = \oint_{\partial\mathscr{B}_\mathbf{X}} \mathbf{P} \cdot \hat{\mathbf{N}} \cdot \delta\mathbf{V} \ \mathrm{d}S + \int_{\mathscr{B}_\mathbf{X}} \varrho_\mathbf{X} \mathbf{B} \cdot \delta\mathbf{V} \ \mathrm{d}V - \int_{\mathscr{B}_\mathbf{X}} \mathbf{S} : \delta\dot{\mathbf{E}} \ \mathrm{d}V.$$

Equation 3.5 can be rewritten to the standard form using Lagrangian $\mathcal{L}_P \to \mathbb{R}$ as follows:

$$\int_{\tau-\Delta\tau}^{\tau} \delta\mathcal{L}_P \ \mathrm{d}\tau = 0, \quad \delta\mathbf{V}(\tau - \Delta\tau) = \delta\mathbf{V}(\tau) = 0.$$

Equation 3.6 is then used for finite element discretization, where the test function representing velocity field is used as $\delta\mathbf{V} \in C^\infty([\tau - \Delta\tau, \tau] \times \Omega; \mathbb{R}^3)$. The resulting set of nonlinear ODE (shortcut of the Ordinary Differential Equations) of the second order is then used for explicit time integration considered in the thesis.

3.3 The Finite Element Method

Consider the approximation of the geometry of the body $\mathscr{B}$ in the initial configuration, where continuous body is subdivided into n_e finite elements in configuration $\Omega_e \subset \Omega$

$$\mathscr{B} \approx \Omega = \bigcup_{e=1}^{n_e} \Omega_e. \tag{3.6}$$

The boundary of the region $\partial\Omega$ is composed of the curves or areas $\partial\Omega_e$ of the elements $\Omega_e : \partial\Omega = \bigcup_{e=1}^{n_r} \partial\Omega_e$, which generally approximate the real geometry of the boundary $\partial\mathscr{B}$ without an overlapping of finite elements. Let us define a finite element (see [17]):

Definition 3.3.1. Let

(i) $K \subseteq \mathbb{R}^n$ is a bounded closed set with nonempty interior and piecewise smooth boundary (the *element domain* Ω_e),

(ii) $\mathcal{P}$ is a finite-dimensional space of functions on K (the space of *shape functions*) and

(iii) $\mathcal{N} = \{\mathsf{N}_1, \mathsf{N}_2, \ldots, \mathsf{N}_k\}$ is a basis for $\mathcal{P}'$ (the set of *nodal variables*).

Then $(K, \mathcal{P}, \mathcal{N})$ is called a *finite element*.

It is implicitly assumed that the nodal variables, N_i, lie in the dual space of some larger function space, e.g., a Sobolev space.

Definition 3.3.2. Let $(K, \mathcal{P}, \mathcal{N})$ be a finite element. The basis $\{\psi_1, \psi_2, \ldots, \psi_k\}$ of $\mathcal{P}$ dual to $\mathcal{N}$ (i.e., $\mathsf{N}_i(\psi_j) = \delta_{ij}$) is called the *nodal basis* of $\mathcal{P}$.

Lemma 3.3.3. Let $\mathcal{P}$ be a d-dimensional vector space and let $\{\mathsf{N}_1, \mathsf{N}_2, \ldots, \mathsf{N}_d\}$ be a subset of the dual space $\mathcal{P}'$. Then the following two statements are equivalent.

(a) $\{\mathsf{N}_1, \mathsf{N}_2, \ldots, \mathsf{N}_d\}$ is a basis for $\mathcal{P}'$.

(b) Given $v \in \mathcal{P}$ with $\mathsf{N}_i v = 0$ for $i = 1, 2, \ldots, d$, then $v = 0$.

Proof. Let $\{\psi_1, \ldots, \psi_d\}$ be some basis for $\mathcal{P}$. $\{\mathsf{N}_1, \ldots, \mathsf{N}_d\}$ is a basis for $\mathcal{P}'$ if given any L in $\mathcal{P}'$,

$$L = \alpha_1 \mathsf{N}_1 + \cdots + \alpha_d \mathsf{N}_d, \quad d = \dim \mathcal{P} = \dim \mathcal{P}'.$$

This equation is equivalent to

$$y_i \equiv L(\psi_i) = \alpha_1 \mathsf{N}_1(\psi_i) + \cdots + \alpha_d \mathsf{N}_d(\psi_i), \quad i = 1, \ldots, d.$$

Let $\mathbf{B} = (\mathsf{N}_j(\psi_i))$, $i, j = 1, \ldots, d$. Thus (a) is equivalent to $\mathbf{B}\alpha = y$ is always solvable, which is the same as $\mathbf{B}$ being invertible.

Given any $v \in \mathcal{P}$, we can write $v = \beta_1 \psi_1 + \cdots + \beta_d \psi_d$. $\mathsf{N}_i v = 0$ means that $\beta_1 \mathsf{N}_i(\psi_1) + \cdots + \beta_d \mathsf{N}_i(\psi_d) = 0$. Therefore, (b) is equivalent to

$$\beta_1 \mathsf{N}_i(\psi_1) + \cdots + \beta_d \mathsf{N}_i(\psi_d) = 0, \text{ for } i = 1, \ldots, d \Rightarrow \beta_1 = \cdots = \beta_d = 0.$$

Let $\mathbf{C} = (\mathsf{N}_i(\psi_j))$, $i, j = 1, \ldots, d$. Then (b) is equivalent to $\mathbf{C}x = 0$ only has trivial solutions, which is the same as $\mathbf{C}$ being invertible. But $\mathbf{C} = \mathbf{B}^T$. Therefore, (a) is equivalent to (b). □

Definition 3.3.4. We say that $\mathcal{N}$ determines $\mathcal{P}$ if $\psi \in \mathcal{P}$ with $\mathsf{N}(\psi) = 0 \ \forall \mathsf{N} \in \mathcal{N}$ implies that $\psi = 0$.

Lemma 3.3.5. Let P be a polynomial of degree $d \leq 1$ that vanishes on a hyperplane L ($x : L(x) = 0$, where L is a non-degenerate linear function). Then we can write $P = LQ$, where Q is a polynomial of degree $(d - 1)$.

Using the basic paradigm of the FEM leads to the selection of such interpolation functions in order to approximate the primary field of variables, namely the displacement field in deformation variant of FEM. Hence the exact solution of the mathematical model is approximated within one finite element by

$$\mathbf{u}(\mathbf{X}) \approx \mathbf{u}(\mathbf{X}) = \sum_{i=1}^{n} \mathsf{N}_i(\mathbf{X}) \mathbf{u}_i, \tag{3.7}$$

where $\mathbf{X}$ denotes the position vector with respect to the initial configuration in Ω_e, $\mathsf{N}_i(\mathbf{X})$ are the shape functions which are defined in Ω_e and the unknown nodal quantities of the primary variable are represented by $\mathbf{u}_i$.

The basic requirement for the choice of the approximation $\boldsymbol{u}_h$ is the convergence of the finite element solution to the true solution of the underlying partial differential equation. For convergence reasons, these functions have to be completed up to the approximation order. From the perspective of application to nonlinear problems and widespread in engineering practice the isoparametric finite elements are considered. Interpolation of geometry and of the kinematic variables is performed in the classical concept of the natural coordinate as follows:

$$\mathbf{X}_e = \sum_{i=1}^{n} \mathsf{N}_i(\boldsymbol{\xi})\mathbf{X}_i, \quad \mathbf{x}_e = \sum_{i=1}^{n} \mathsf{N}_i(\boldsymbol{\xi})\mathbf{x}_i. \tag{3.8}$$

The mapping of an element from the initial configuration Ω_e to the current configuration $\phi_{\mathscr{B}}(\Omega_e)$ is performed using the approximate deformation map ϕ_Ω which is described by $\phi_{\mathscr{B},e}$ to show its relation to a specific finite element Ω_e. For the mapping, the deformations gradient F_e related to the element as a discrete version of the continuum mechanical description is required

$$\mathsf{F}_e = \mathsf{j}_e \mathsf{J}_e^{-1} \text{ and } \mathsf{J}_e = \det \mathsf{F}_e = \frac{\det \mathsf{j}_e}{\det \mathsf{J}_e}. \tag{3.9}$$

The deformation gradient F_e is defined by the isoparametric mapping from reference configuration to the initial configuration Ω_e and to the current configuration $\phi_{\mathscr{B}}(\Omega_e)$, thus

$$\begin{aligned} \mathsf{j}_e &= \operatorname{Grad}_\xi \mathbf{x}_e = \partial_{\boldsymbol{\xi}}\mathbf{x} = \sum_{i=1}^{n} \mathsf{N}_{i,\xi}(\boldsymbol{\xi})\mathbf{x}_i \otimes \mathsf{E}_\xi, \\ \mathsf{J}_e &= \operatorname{Grad}_\xi \mathbf{X}_e = \partial_{\boldsymbol{\xi}}\mathbf{X} = \sum_{i=1}^{n} \mathsf{N}_{i,\xi}(\boldsymbol{\xi})\mathbf{X}_i \otimes \mathsf{E}_\xi \end{aligned} \tag{3.10}$$

3.4 Numerical Threatment of Solution to Problems in Structural Dynamics

Obtaining an analytical solution of IBVP is very difficult or even impossible due to the complexity of the respective problem, but in simple cases it provides a good tool for the verification and validation of the numerical model, which comes from the numerical approximation by means of suitable numerical methods.

In the field of statics and dynamics of structures it mostly concerns to FEM for discretization of the space domain into the FE subdomains and the Finite Difference Method (FDM), which is frequently used for discretization in the domain of time. The technique when the spatial discretization using FEM is performed as first and discretization of time domain by FDM is performed subsequently is called the *Method of Lines*, which oppose to the *Method of Rothe*. The method of lines is considered.

3.4.1 Numerical Solution to a Set of Semidiscrete Nonlinear Orinary Differential Equations of the Second Order

Cauchy's equations of motion represent a dynamic equilibrium in the current configuration of a body $\mathscr{B}_t$. Respective body $\mathscr{B}_t$ is represented by the domain of its finite element discretization Ω. If the given equations of motion are nonlinear, then the resulting form of semidiscrete nonlinear ordinary differential equations (ODE) of the second order is thus the following:

$$\mathbb{M}\ddot{\mathbf{u}} + \mathbf{f}_{int}(\mathbf{u}) = \mathbf{f}_{ext}(\mathbf{u}) \in \Omega. \tag{3.11}$$

Integration methods are classified according to the structure of the time difference equation. The difference equations for first and second derivatives, respectively, can be written in the general functional expressions (see [7])

$$\sum_{i=0}^{m} \left(\alpha_i \mathbf{u}^{m-i} - \Delta t \beta_i \dot{\mathbf{u}}^{m-i}\right) = \mathbf{0}, \quad \sum_{i=0}^{m} \left(\overline{\alpha}_i \mathbf{u}^{m-i} - \Delta t^2 \overline{\beta}_i \ddot{\mathbf{u}}^{m-i}\right) = \mathbf{0},$$

where m is the number of steps in the difference equation. The difference formulas for the first or second derivatives are called explicit if $\beta_0 = \overline{\beta}_0 = 0$. Thus a difference formula is called explicit if the equation for the function at time step m involves only the derivatives at previous time steps. In the explicit central difference formula for the second derivative $\beta_0 = \overline{\beta}_2 = 0$ and $\overline{\beta}_1 = 1$, respectively.

Incremental form of 3.11 is obtained by subtracting two equations 3.11 expressed for the two subsequent time levels τ and $\tau - \Delta\tau$, respectively. It leads to the following form:

$$\mathbb{M}\Delta\ddot{\mathbf{u}} + \Delta\mathbf{f}_{int} - \Delta\mathbf{f}_{ext} = \Delta\mathbf{f} \in \Omega. \tag{3.12}$$

The nonlinear behaviour of the body from a global view is expressed through the residual forces $\Delta\mathbf{f}|_\Omega$, which are produced by the imbalance between the internal forces $\mathbf{f}_{int}|_\Omega$, forces induced by the mass $\mathbf{f}_{mass}|_\Omega = \mathbb{M}\ddot{\mathbf{u}}$ and external forces $\mathbf{f}_{ext}|_\Omega$, respectively. This imbalance at time τ can be eliminated by linearizing of these residual forces around the current equilibrium state, where suitable norm of residual forces is less than or equal to some tolerance $\|\Delta\mathbf{f}|_\Omega\| \leq \epsilon$.

The linearization is carried out by the expansion of the Taylor's polynomial considering the first variation only. It can be expressed in the following compact form using the tangential operators (see [93]):

$$\begin{aligned}
{}^{i+1}[\Delta\mathbf{f}]_\Omega^\tau &\approx {}^{i}[\Delta\mathbf{f}]_\Omega^\tau + {}^{i}\left[\mathbb{J}^{\mathrm{T}}\right]_\Omega^\tau \cdot {}^{i+1}[\Delta\mathbf{u}]_\Omega^\tau \\
&\approx {}^{i}[\Delta\mathbf{f}]_\Omega^\tau + {}^{i}\left[\underbrace{\mathbb{M}\partial_{\mathbf{u}}\ddot{\mathbf{u}}}_{\mathbb{M}\Delta\ddot{\mathbf{u}}} + \underbrace{\mathbb{K}^{\mathrm{T}} + \mathbb{D}^{\mathrm{T}}\partial_{\mathbf{u}}\dot{\mathbf{u}}}_{\Delta\mathbf{f}_{int}} - \underbrace{\mathbb{F}^{\mathrm{T}}}_{\Delta\mathbf{f}_{ext}}\right]_\Omega^\tau \cdot \left[{}^{i+1}\mathbf{u} - {}^{i}\mathbf{u}\right]_\Omega^\tau,
\end{aligned}$$

where $\mathbb{K}_\Omega^{\mathrm{T}} = \partial_{\mathbf{u}}\mathbf{f}_{int}$ represents the tangent stiffness matrix, $\mathbb{D}_\Omega^{\mathrm{T}} = \partial_{\dot{\mathbf{u}}}\mathbf{f}_{int}$ is the tangent damping matrix related to the structural damping represented by the viscous part in

appropriate material model, $\mathbb{F}_{\Omega}^{\mathrm{T}} = \partial_{\mathbf{u}}\mathbf{f}_{ext}$ is the tangent operator representing the effect of the change of position of the external forces and $\mathbb{J}_{\Omega}^{\mathrm{T}}$ is the compact Jacobian tangent operator

$$ {}^{i}\left[\mathbb{J}^{\mathrm{T}}\right]_{\Omega}^{\tau} \equiv {}^{i}[\partial_{\mathbf{u}}\Delta\mathbf{f}]_{\Omega}^{\tau} = {}^{i}\left[\mathbb{M}\partial_{\mathbf{u}}\ddot{\mathbf{u}} + \mathbb{K}^{\mathrm{T}} + \mathbb{D}^{\mathrm{T}}\partial_{\mathbf{u}}\dot{\mathbf{u}} - \mathbb{F}^{\mathrm{T}}\right]_{\Omega}^{\tau}. $$

The acceleration and the velocity must be expressed through a linear approximation in finite differences. The approach above is focused on common solution by the implicit direct numerical time integration methods of equations of motion. This applies especially to the methods such as Newmark-β method, Houbolt's method, Wilson-θ method, or other implicit methods. The linearization of residual forces by tangential operators is used by the diverse numerical methods for a solution of nonlinear algebraic equations, which are generated by the mentined methods.

The situation gets easier when using one of the available explicit time integration algorithms. The difference formula is called explicit if the equation for the function at time step m involves only the derivatives at previous time steps.

It concerns primarily the central differential method and to the explicit form of the Newmark-β method. In this algorithm, the size of increment in deflections is explicitly composed, hence the deformation gradient. It is further used for a strain and stress tensor composition. The major phase in an explicit time integration algorithm of equations of motion is an integration of an internal forces in each time increment, where assembly of an expensive tangential operators coming from the linearized form of energetic functionals is not required compared to the implicit time integration algorithms.

The explicit time integration algorithm in this case uses a member of the first law of thermodynamics (conservation of energy) that expresses the *deformation power* for determination of the internal forces. It is a rate of change of the *strain energy* written as (see [133], [7])

$$ \dot{\mathcal{E}} \equiv \int_{\mathscr{B}_{\mathbf{X}}} \mathbf{P}^{T} : \mathbf{L}\, \mathrm{d}V = \int_{\mathscr{B}_{\mathbf{X}}} \mathbf{P}^{T} : \mathrm{Grad}\ \mathbf{V}\, \mathrm{d}V = \int_{\mathscr{B}_{\mathbf{X}}} \mathbf{S} : \dot{\mathbf{E}}\, \mathrm{d}V. \tag{3.13} $$

Then the increment in the strain energy $\Delta\mathcal{E}$ for a time interval $[\tau - \Delta\tau, \tau]$, where $\mathcal{E}^{\tau} = \mathcal{E}^{\tau-\Delta\tau} + \Delta\mathcal{E}$ gets the form $\Delta\mathcal{E} = \mathcal{E}^{\tau} - \mathcal{E}^{\tau-\Delta\tau} \equiv \int_{\tau-\Delta\tau}^{\tau} \left(\int_{\mathscr{B}_{\mathbf{X}}} \mathbf{S}^{\tau} : \dot{\mathbf{E}}\, \mathrm{d}V\right)\, \mathrm{d}t$. Thus the following term expresses the rules for a gaining of an internal forces in na explicit approach as time τ proceed

$$ \boxed{\mathcal{E}^{\tau} = \mathcal{E}^{\tau-\Delta\tau} + \int_{\mathscr{B}_{\mathbf{X}}} \mathbf{S}^{\tau} : \Delta\mathbf{E}\, \mathrm{d}V, \quad \mathbf{S}^{\tau} = \varrho_{\mathbf{X}}\partial_{\mathbf{E}^{\tau}}\Psi\left(\mathbf{E}^{\tau}\right)}. \tag{3.14} $$

In the case of small strain linear elastic material model considered in the thesis, the term 3.14 gets the following form:

$$ \mathcal{E}^{\tau} = \mathcal{E}^{\tau-\Delta\tau} + \int_{\mathscr{B}_{\mathbf{X}}} \boldsymbol{\sigma}^{\tau} : \Delta\boldsymbol{\varepsilon}\, \mathrm{d}V, \quad \boldsymbol{\sigma}^{\tau} = \varrho\partial_{\boldsymbol{\varepsilon}^{\tau}}\Psi\left(\boldsymbol{\varepsilon}^{\tau}\right) = \mathbb{C}_{ijkl} : \boldsymbol{\varepsilon}^{\tau}, \tag{3.15} $$

where $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$ represents corotated Cauchy's stress tensor and corotated small strain tensor, respectively. Respective tensors are composed in Eulerian coordinates as updated Lagrangian formulation suggest. The rest of the used explicit time integration algorithm is presented in chapter 3.5.

In an explicit approach, mass matrix $\mathbb{M}$ must be considered in its diagonalized form (ie. lumped mass matrix). The update of the nodal velocities and nodal displacements can then be accomplished without solving any equations provided that the mentioned mass matrix $\mathbb{M}$ is diagonal.

3.4.2 A C^0 Triangular Shell Finite Element with Corotational Coordinates

For the purpose of designing and testing a hybrid-parallel computing model of an explicit numerical solver related to the problem solution of nonlinear dynamics using FEM, an effective triangular shell finite element developed and published by T. Belytschko, H. Stolarski and N. Carpenter in 1984 (see [8]) as complement to the Belytskchko-Lin-Tsay quadrilateral shell element (see [6]) is applied with corotational coordinates for *small strain* and *large rotation* kinematics (see [58], [5]).

It is a plate element of the Reissner-Mindlin type (see [103]), which uses linear fields for rotations and transverse deflections. The element only requires a single quadrature point which reduces the number of computations, which is very attractive for non-linear analysis which is here in the main focus with respect to explicit dynamics. The accuracy is similar to the 4-node bilinear Mindlin plate and its rate of convergence is approximately of quadratic order as noted in [8].

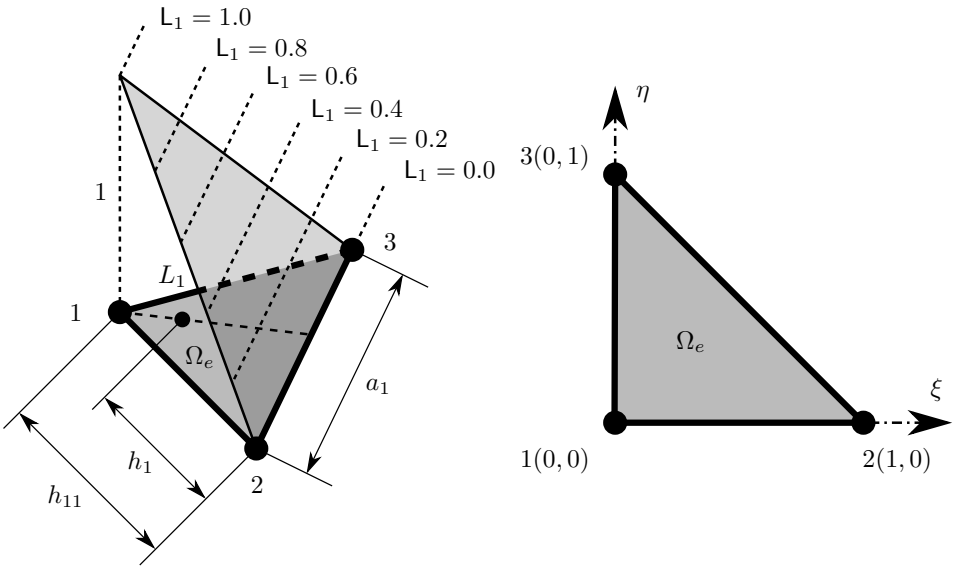


Figure 31: Area and natural coordinates of Lagrangian triangular 3-noded element.

The respective triangular finite element is also implemented, e.g. in commercial FEM software LS-DYNA or in free and open-source software project Impact. The respective element represents the Lagrangian triangular element. As a basis of this element is a standard triangular three-node isoparametric finite element with a linear approximation polynom is used (see [136]). The process of deriving the base functions of respective element is thus straightforward. The area coordinates for FE nodes are as follows:

$$\mathsf{L}_i = \frac{h_i}{h_{ii}} = \frac{\frac{1}{2}h_i a_i}{\frac{1}{2}h_{ii}a_i} = \frac{\mathsf{A}_i}{\mathsf{A}}, \quad \sum_{i=1}^{3}\mathsf{L}_i = 1, \quad \mathsf{x} = \sum_{i=1}^{3}\mathsf{x}_i\mathsf{L}_i, \quad \mathsf{y} = \sum_{i=1}^{3}\mathsf{y}_i\mathsf{L}_i. \tag{3.16}$$

The statement 3.16 is then rewritten to the following form to get inverse mapping:

$$\begin{bmatrix}1\\ \mathsf{x}\\ \mathsf{y}\end{bmatrix} = \begin{bmatrix}1 & 1 & 1\\ \mathsf{x}_1 & \mathsf{x}_2 & \mathsf{x}_3\\ \mathsf{y}_1 & \mathsf{y}_2 & \mathsf{y}_3\end{bmatrix}\cdot\begin{bmatrix}\mathsf{L}_1\\ \mathsf{L}_2\\ \mathsf{L}_3\end{bmatrix} \rightarrow \begin{bmatrix}\mathsf{L}_1\\ \mathsf{L}_2\\ \mathsf{L}_3\end{bmatrix} = \frac{1}{2\mathsf{A}}\begin{bmatrix}\mathsf{x}_2\mathsf{y}_3-\mathsf{x}_3\mathsf{y}_2 & \mathsf{y}_2-\mathsf{y}_3 & \mathsf{x}_3-\mathsf{x}_2\\ \mathsf{x}_3\mathsf{y}_1-\mathsf{x}_1\mathsf{y}_3 & \mathsf{y}_3-\mathsf{y}_1 & \mathsf{x}_1-\mathsf{x}_3\\ \mathsf{x}_1\mathsf{y}_2-\mathsf{x}_2\mathsf{y}_1 & \mathsf{y}_1-\mathsf{y}_2 & \mathsf{x}_2-\mathsf{x}_1\end{bmatrix}\cdot\begin{bmatrix}1\\ \mathsf{x}\\ \mathsf{y}\end{bmatrix}.$$

The base functions $\mathsf{N}_i \circ \boldsymbol{\xi}(\xi,\eta)$ are obtained by simple translation from $\mathsf{L}_i(\mathsf{x}\rightarrow\xi, \mathsf{y}\rightarrow\eta)$ to $\mathsf{N}_i(\boldsymbol{\xi})$ with a simple change of coordinates

$$\mathsf{L}_1(\xi,\eta)\rightarrow\mathsf{N}_1(\boldsymbol{\xi}) = 1-\xi-\eta, \quad \mathsf{L}_2(\xi,\eta)\rightarrow\mathsf{N}_2(\boldsymbol{\xi}) = \xi, \quad \mathsf{L}_3(\xi,\eta)\rightarrow\mathsf{N}_3(\boldsymbol{\xi}) = \eta.$$

Base $\mathbf{e}$ of corotated coordinate frame of triangular finite element is obtained as follows:

$$\overline{\mathbf{e}}_{\hat{x}} = \mathsf{X}_{2,1}-\mathsf{X}_{1,1}, \quad \overline{\mathbf{e}}_{\hat{y}'} = \mathsf{X}_{3,2}-\mathsf{X}_{1,2}, \quad \overline{\mathbf{e}}_{\hat{z}} = \overline{\mathbf{e}}_{\hat{x}}\times\overline{\mathbf{e}}_{\hat{y}'},$$

where $\mathsf{X}_{i,j}$ denotes j-th coordinate of coordinate vector $\mathbf{X}$ of a finite element node i in a global reference frame defined by the base $\mathbf{E}$. The respective base vectors are normalized as follows:

$$\hat{\mathbf{e}}_{\hat{z}} = \frac{\overline{\mathbf{e}}_{\hat{z}}}{\|\overline{\mathbf{e}}_{\hat{z}}\|}, \quad \hat{\mathbf{e}}_{\hat{x}} = \frac{\overline{\mathbf{e}}_{\hat{x}}}{\|\overline{\mathbf{e}}_{\hat{x}}\|}, \quad \hat{\mathbf{e}}_{\hat{y}} = \hat{\mathbf{e}}_{\hat{z}}\times\hat{\mathbf{e}}_{\hat{x}}.$$

Pursuant the obtained base of the corotated coordinate system, a transition (transformation) matrix from base $\hat{\mathbf{e}}$ to base $\mathbf{E}$ can be assembled $\mathbb{T}_e = [\hat{\mathbf{e}}_{\hat{x}}|\hat{\mathbf{e}}_{\hat{y}}|\hat{\mathbf{e}}_{\hat{z}}]$. Transition matrix is then used to transform all vectors from global coordinate system to corotated coordinate system of the respective element. This mainly applies to the position vector $\hat{\mathbf{x}}_i = \mathbb{T}_e\mathbf{X}_i$, velocity vector $\hat{\mathbf{v}}_i = \mathbb{T}_e\mathbf{V}_i$ and angular velocity vector $\hat{\boldsymbol{\theta}}_i = \mathbb{T}_e\boldsymbol{\Theta}_i$ of all element nodes i. Jacobian is defined as follows:

$$\mathbf{j}_e = \mathrm{Grad}_\xi\ \mathbf{x}_e = \partial_{\boldsymbol{\xi}}\mathbf{x} = \sum_{i=1}^{n}\mathsf{N}_{i,\xi}(\boldsymbol{\xi})\hat{\mathbf{x}}_i\otimes\hat{\mathbf{e}}_\xi,$$

thus

$$\begin{aligned}
\mathbf{j}_e &= \begin{bmatrix}\partial_\xi x_1 & \partial_\eta x_1\\ \partial_\xi x_2 & \partial_\eta x_2\end{bmatrix} = \begin{bmatrix}\hat{\mathsf{x}}_2-\hat{\mathsf{x}}_1 & \hat{\mathsf{x}}_3-\hat{\mathsf{x}}_1\\ \hat{\mathsf{y}}_2-\hat{\mathsf{y}}_1 & \hat{\mathsf{y}}_3-\hat{\mathsf{y}}_1\end{bmatrix},\\
\det\mathbf{j}_e &= 2A = (\hat{\mathsf{x}}_2-\hat{\mathsf{x}}_1)(\hat{\mathsf{y}}_3-\hat{\mathsf{y}}_1)-(\hat{\mathsf{x}}_3-\hat{\mathsf{x}}_1)(\hat{\mathsf{y}}_2-\hat{\mathsf{y}}_1),\\
\mathbf{j}_e^{-1} &= \begin{bmatrix}\partial_{x_1}\xi & \partial_{x_2}\xi\\ \partial_{x_1}\eta & \partial_{x_2}\eta\end{bmatrix} = \frac{1}{\det\mathbf{j}_e}\begin{bmatrix}\partial_\eta x_2 & -\partial_\eta x_1\\ -\partial_\xi x_2 & \partial_\xi x_1\end{bmatrix} = \frac{1}{\det\mathbf{j}_e}\begin{bmatrix}\hat{\mathsf{y}}_3-\hat{\mathsf{y}}_1 & \hat{\mathsf{x}}_1-\hat{\mathsf{x}}_3\\ \hat{\mathsf{y}}_1-\hat{\mathsf{y}}_2 & \hat{\mathsf{x}}_2-\hat{\mathsf{x}}_1\end{bmatrix}.
\end{aligned}$$

Considering a simple way of defining the local system of triangular element proposed by Belytschko et al. (see [8]), where $\hat{\mathsf{x}}_1 = \hat{\mathsf{y}}_1 = \hat{\mathsf{y}}_2 = 0$, and based on the previous relations, it is possible to express and quantify the derivatives of the element shape functions as follows:

$$\begin{aligned}\partial_{x_1}\mathsf{N}_i(\boldsymbol{\xi}) &= \partial_\xi \mathsf{N}_i \partial_{x_1}\xi + \partial_\eta \mathsf{N}_i \partial_{x_1}\eta = \frac{1}{\hat{\mathsf{x}}_3\hat{\mathsf{y}}_3}\begin{bmatrix}-\hat{\mathsf{y}}_3 & \hat{\mathsf{y}}_3 & 0\end{bmatrix}^T,\\ \partial_{x_2}\mathsf{N}_i(\boldsymbol{\xi}) &= \partial_\xi \mathsf{N}_i \partial_{x_2}\xi + \partial_\eta \mathsf{N}_i \partial_{x_2}\eta = \frac{1}{\hat{\mathsf{x}}_3\hat{\mathsf{y}}_3}\begin{bmatrix}\hat{\mathsf{x}}_3 - \hat{\mathsf{x}}_2 & -\hat{\mathsf{x}}_3 & \hat{\mathsf{x}}_2\end{bmatrix}^T.\end{aligned}$$

The nodal degrees of freedom of the respective finite element are velocities of the midplane $\hat{v}_m^x, \hat{v}_m^y, \hat{v}_m^z$ and angular velocities $\hat{\theta}_x, \hat{\theta}_y, \hat{\theta}_z$. Then the rate forms of the strain-displacement equations are then as follows:

$$\begin{gathered}\dot{\varepsilon}_x = \partial_x \hat{v}_x^m + z\partial_x\hat{\theta}_y, \quad \dot{\varepsilon}_y = \partial_y\hat{v}_y^m - z\partial_y\hat{\theta}_x,\\ 2\dot{\varepsilon}_{xy} = \partial_y\hat{v}_x^m + \partial_x\hat{v}_y^m + z\left(\partial_y\hat{v}_y - \partial_x\hat{v}_x\right), \quad 2\dot{\varepsilon}_{xz} = \partial_x\hat{v}_z^m + \hat{\theta}_y, \quad 2\dot{\varepsilon}_{yz} = \partial_y\hat{v}_z^m - \hat{\theta}_x.\end{gathered}$$

Here $\dot{\varepsilon}_{\alpha,\beta}$ denotes the rate of deformation (strain rate). Strain rates $\dot{\varepsilon}_x, \dot{\varepsilon}_y$ and $\dot{\varepsilon}_{xy}$ are in-plane strains and arise from bending and stretching of the shell element. Strains $\dot{\varepsilon}_{xz}$ and $\dot{\varepsilon}_{yz}$ denote transverse shear strains, then $\dot{\boldsymbol{\varepsilon}} = \mathbf{B}\hat{\mathbf{v}}$, where $\mathbf{B}$ and $\hat{\mathbf{v}}$ denote derivatives of the nodal interpolation functions and nodal velocities, respectively. Strain rates are partitioned into membrane and bending. The membrane contributions $\dot{\boldsymbol{\varepsilon}}^m = \mathbf{B}_m\hat{\mathbf{v}}$ are given by as follows:

$$\begin{bmatrix}\dot{\varepsilon}_x^m\\ \dot{\varepsilon}_y^m\\ 2\dot{\varepsilon}_{xy}^m\end{bmatrix} = \frac{1}{\hat{\mathsf{x}}_2\hat{\mathsf{y}}_3}\begin{bmatrix}\hat{\mathsf{y}}_3 & 0 & \hat{\mathsf{y}}_3 & 0 & 0 & 0\\ 0 & \hat{\mathsf{x}}_3 - \hat{\mathsf{x}}_2 & 0 & -\hat{\mathsf{x}}_3 & 0 & \hat{\mathsf{x}}_2\\ \hat{\mathsf{x}}_3 - \hat{\mathsf{x}}_2 & -\hat{\mathsf{y}}_3 & -\hat{\mathsf{x}}_3 & \hat{\mathsf{y}}_3 & \hat{\mathsf{x}}_2 & 0\end{bmatrix}\cdot\begin{bmatrix}\hat{\mathsf{v}}_{\hat{\mathsf{x}}_1}\\ \hat{\mathsf{v}}_{\hat{\mathsf{y}}_1}\\ \hat{\mathsf{v}}_{\hat{\mathsf{x}}_2}\\ \hat{\mathsf{v}}_{\hat{\mathsf{y}}_2}\\ \hat{\mathsf{v}}_{\hat{\mathsf{x}}_3}\\ \hat{\mathsf{v}}_{\hat{\mathsf{y}}_3}\end{bmatrix},$$

and bending contributions $\dot{\boldsymbol{\kappa}} = \mathbf{B}_b\hat{\boldsymbol{\theta}}^{def}$ are given as follows:

$$\begin{bmatrix}\dot{\kappa}_x\\ \dot{\kappa}_y\\ 2\dot{\kappa}_{xy}\end{bmatrix} = \frac{-1}{\hat{\mathsf{x}}_2\hat{\mathsf{y}}_3}\begin{bmatrix}0 & -\hat{\mathsf{y}}_3 & 0 & \hat{\mathsf{y}}_3 & 0 & 0\\ \hat{\mathsf{x}}_3 - \hat{\mathsf{x}}_2 & 0 & \hat{\mathsf{x}}_3 & 0 & -\hat{\mathsf{x}}_2 & 0\\ \hat{\mathsf{y}}_3 & \hat{\mathsf{x}}_3 - \hat{\mathsf{x}}_2 & -\hat{y}_3 & -\hat{\mathsf{x}}_3 & 0 & \hat{\mathsf{x}}_2\end{bmatrix}\cdot\begin{bmatrix}\hat{\theta}_{\hat{\mathsf{x}}_1}^{def}\\ \hat{\theta}_{\hat{\mathsf{y}}_1}^{def}\\ \hat{\theta}_{\hat{\mathsf{x}}_2}^{def}\\ \hat{\theta}_{\hat{\mathsf{y}}_2}^{def}\\ \hat{\theta}_{\hat{\mathsf{x}}_3}^{def}\\ \hat{\theta}_{\hat{\mathsf{y}}_3}^{def}\end{bmatrix},$$

Then the local element strain rates $\dot{\boldsymbol{\varepsilon}} = \dot{\boldsymbol{\varepsilon}}^m - z\dot{\boldsymbol{\kappa}}$ are then obtained by combining the above two relation

$$\begin{bmatrix}\dot{\varepsilon}_x\\ \dot{\varepsilon}_y\\ 2\dot{\varepsilon}_{xy}\end{bmatrix} = \begin{bmatrix}\dot{\varepsilon}_x^m\\ \dot{\varepsilon}_y^m\\ 2\dot{\varepsilon}_{xy}^m\end{bmatrix} - z\begin{bmatrix}\dot{\kappa}_x\\ \dot{\kappa}_y\\ 2\dot{\kappa}_{xy}\end{bmatrix}.$$

The remaining transverse shear strain rates $\hat{\boldsymbol{\varepsilon}}^s = \mathbf{B}_s \hat{\boldsymbol{\theta}}^{def}$ are given as follows:

$$\begin{bmatrix} 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{bmatrix} = \frac{1}{6\hat{\mathrm{x}}_2\hat{\mathrm{y}}_3} \begin{bmatrix} -\hat{\mathrm{y}}_3^2 & \hat{\mathrm{y}}_3\,(2\hat{\mathrm{x}}_2+\hat{\mathrm{x}}_3) & \hat{\mathrm{y}}_3^2 & \hat{\mathrm{y}}_3\,(3\hat{\mathrm{x}}_2-\hat{\mathrm{x}}_3) & 0 & \hat{\mathrm{x}}_2\hat{\mathrm{y}}_3 \\ \hat{\mathrm{y}}_3\,(\hat{\mathrm{x}}_2-2\hat{\mathrm{x}}_2) & \hat{\mathrm{x}}_2^2-\hat{\mathrm{x}}_3^2 & -\hat{\mathrm{y}}_3^2\,(\hat{\mathrm{x}}_2+\hat{\mathrm{x}}_3) & \hat{\mathrm{x}}_3\,(\hat{\mathrm{x}}_3-2\hat{\mathrm{x}}_2) & -3\hat{\mathrm{x}}_2\hat{\mathrm{y}}_3 & \hat{\mathrm{x}}_2\,(2\hat{\mathrm{x}}_3-\hat{\mathrm{x}}_2) \end{bmatrix} \cdot \begin{bmatrix} \hat{\theta}^{def}_{\hat{\mathrm{x}}_1} & \hat{\theta}^{def}_{\hat{\mathrm{y}}_1} & \hat{\theta}^{def}_{\hat{\mathrm{x}}_2} & \hat{\theta}^{def}_{\hat{\mathrm{y}}_2} & \hat{\theta}^{def}_{\hat{\mathrm{x}}_3} & \hat{\theta}^{def}_{\hat{\mathrm{y}}_3} \end{bmatrix}^T.$$

Angular velocities $\hat{\boldsymbol{\theta}}^{def} = \hat{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}^{rig}$ are the deformation components of the angular velocity $\hat{\boldsymbol{\theta}}$, which is obtained by subtracting the rigid body rotations $\hat{\boldsymbol{\theta}}^{rig}$.

The key feature here, providing numerous simplifications, is the use of corotational coordinate system in each element. The corotational coordinate system is embedded in the element so that it rotates with the element. The coordinates remain embedded in this manner throughout the deformation of the elements, so that they rotate but do not deform with the elements. This type of formulation can accurately handle any magnitude of rigid body rotations and very large extensional strains.

The nodal coordinates $\mathbf{X}^{t+\Delta t}$ of the deformed body $\mathscr{B}_t$ can be calculated from those of the undeformed state $\mathbf{X}_0$ of the body $\mathscr{B}_{t_0}$. This is achieved by adding the displacements $\mathbf{X}^{t+\Delta t} = \mathbf{X}_0 + \mathbf{u}^{t+\Delta t}$.

The calculation of the deformatoric displacements $\mathbf{u}^{def,t+\Delta t}$ is sufficient to transform the coordinates into a rotated system parallel to the element coordinate system with origin in global origin $\mathbf{u}^{def,t+\Delta t} = \mathbb{T}\mathbf{X}^{t+\Delta t} - \mathbb{T}_0\mathbf{X}_0$.

In the sense of the corotational formulation the relation between strain and nodal displacements has become non-linear, small strains in the discretized form are defined as follows:

$$\boldsymbol{\varepsilon}^{t+\Delta t} = \mathbf{B}_{lin}\mathbf{u}^{def,t+\Delta t} = \mathbf{B}_{lin}\big[\mathbb{T}(\mathbf{u}^{t+\Delta t})(\mathbf{X}_0+\mathbf{u}^{t+\Delta t}) - \mathbb{T}_0\mathbf{X}_0\big].$$

The general form of the the matrix $\mathbf{B}$ is obtained as the derivative of the strain with respect to the global nodal displacements

$$\begin{aligned} \mathbf{B} = \partial_{\mathbf{u}^{t+\Delta t}}\boldsymbol{\varepsilon}^{t+\Delta t} &= \partial_{\mathbf{u}^{t+\Delta t}}\Big[\mathbf{B}_{lin}\big(\mathbb{T}(\mathbf{u}^{t+\Delta t})(\mathbf{X}_0+\mathbf{u}^{t+\Delta t}) - \mathbb{T}_0\mathbf{X}_0\big)\Big] \\ &= \mathbf{B}_{lin}\big[\partial_{\mathbf{u}^{t+\Delta t}}\mathbb{T}(\mathbf{X}_0+\mathbf{u}^{t+\Delta t}) + \mathbb{T}(\mathbf{u}^{t+\Delta t})\big]. \end{aligned}$$

Based on the variational form of the Cauchy's equations of motion, the energy functional related to the internal nodal forces in the global system can be written in its discretized form as follows:

$$\begin{aligned} \mathbf{f}_{int}^{t+\Delta t} &= \int_{\Omega_e} \mathbf{B}^T\boldsymbol{\sigma}^{t+\Delta t}\,\mathrm{d}v = \int_{\Omega_e}\big[\partial_{\mathbf{u}^{t+\Delta t}}\mathbb{T}(\mathbf{X}_0+\mathbf{u}^{t+\Delta t}) + \mathbb{T}(\mathbf{u})\big]^T\mathbf{B}_{lin}^T\boldsymbol{\sigma}^{t+\Delta t}\,\mathrm{d}v \\ \mathbf{f}_{int}^{t+\Delta t} &= \int_{\Omega_e}\big[(\mathbf{X}_0+\mathbf{u}^{t+\Delta t})^T\partial_{\mathbf{u}^{t+\Delta t,T}}\mathbb{T}^T + \mathbb{T}^T\big]\mathbf{B}_{lin}^T\mathbf{D}\mathbf{B}_{lin}\big[\mathbb{T}(\mathbf{X}_0+\mathbf{u}^{t+\Delta t}) - \mathbb{T}_0\mathbf{X}_0\big]\,\mathrm{d}v \\ \mathbf{f}_{int}^{t+\Delta t} &= \big[(\mathbf{X}_0+\mathbf{u}^{t+\Delta t})^T\partial_{\mathbf{u}^{t+\Delta t,T}}\mathbb{T}^T + \mathbb{T}^T\big]\int_{\Omega_e}\mathbf{B}_{lin}^T\boldsymbol{\sigma}^{t+\Delta t}\,\mathrm{d}v. \end{aligned} \tag{3.17}$$

In an explicit form of FEM, where the new displacements are assembled on the basis of an explicit integration algorithm of the equations of motion from the internal, external and contact forces from the preceeding time step $t - \Delta t$, computation of the strain rates, the nodal rotations must be converted to deformation nodal rotations. The calculation of the rigid body rotation can be obtained on the basis of the Kirchhoff structure in the following formulas:

$$\begin{aligned} 2\dot{\varepsilon}_{yz} \equiv \partial_{\hat{y}}\hat{v}_z^m - \hat{\theta}_x = 0 \quad &\Rightarrow \quad \hat{\theta}_x = \partial_{\hat{y}}\hat{v}_z^m, \\ 2\dot{\varepsilon}_{xz} \equiv \partial_{\hat{x}}\hat{v}_z^m + \hat{\theta}_y = 0 \quad &\Rightarrow \quad \hat{\theta}_y = -\partial_{\hat{x}}\hat{v}_z^m, \end{aligned}$$

then the approximation has the following form:

$$\begin{aligned} \hat{\boldsymbol{\theta}}^{rig} \quad &= \quad \partial_{\hat{\mathbf{x}}}\mathbf{N}(\boldsymbol{\xi})\hat{\mathbf{v}} = \sum_{i=1}^{n} \mathsf{N}_{i,\xi}(\boldsymbol{\xi})\hat{\mathbf{v}}_i \otimes \hat{\mathbf{e}}_\xi \Rightarrow \\ &\Rightarrow \quad \begin{bmatrix} \hat{\theta}_{\hat{x}}^{rig} \\ \hat{\theta}_{\hat{y}}^{rig} \end{bmatrix} = \begin{bmatrix} \partial_{\hat{x}_1}\mathsf{N}_1(\boldsymbol{\xi}) & \partial_{\hat{x}_1}\mathsf{N}_2(\boldsymbol{\xi}) & \partial_{\hat{x}_1}\mathsf{N}_3(\boldsymbol{\xi}) \\ \partial_{\hat{x}_2}\mathsf{N}_1(\boldsymbol{\xi}) & \partial_{\hat{x}_2}\mathsf{N}_2(\boldsymbol{\xi}) & \partial_{\hat{x}_2}\mathsf{N}_3(\boldsymbol{\xi}) \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathsf{v}}_{\hat{z}_1} \\ \hat{\mathsf{v}}_{\hat{z}_2} \\ \hat{\mathsf{v}}_{\hat{z}_3} \end{bmatrix}. \end{aligned} \tag{3.18}$$

From 3.18 imply the following formulas expressing the rigid body rotations:

$$\hat{\theta}_{\hat{x}}^{rig} = \frac{1}{\hat{\mathsf{x}}_2\hat{\mathsf{y}}_3}\big((\hat{\mathsf{v}}_{\hat{z}_3} - \hat{\mathsf{v}}_{\hat{z}_1})\,\hat{\mathsf{x}}_2 - (\hat{\mathsf{v}}_{\hat{z}_2} - \hat{\mathsf{v}}_{\hat{z}_1})\,\hat{\mathsf{x}}_3\big), \quad \hat{\theta}_{\hat{y}}^{rig} = \frac{1}{\hat{\mathsf{x}}_2}\,(\hat{\mathsf{v}}_{\hat{z}_1} - \hat{\mathsf{v}}_{\hat{z}_2})\,. \tag{3.19}$$

These are constants and considered properties of the C^0 triangular element. These relations hold true for the corresponding velocity terms (see [136]). Equation 3.17 is therefore considerably simplified in terms of the explicit integration algorithm of equations of motion as follows:

$$\mathbf{f}_{int}^{t+\Delta t} = \int_{\Omega_e} \mathbf{u}^{def,t+\Delta t,T}\mathbf{B}^T\mathbf{D}\mathbf{B}\mathbf{u}^{def,t+\Delta t}\,\mathrm{d}v,$$

where $\mathbf{u}^{def,t+\Delta t} = \mathbf{u}^{t+\Delta t} - \mathbf{u}^{rig,t+\Delta t}$, and $\mathbf{B}$ is assembled in corotated coordinate system.

The LS-DYNA theoretical manual (see [39]) provides an alternative approach for obtaining formula of rigid body rotation $\hat{\theta}_x^{rig}$ which is shown in the Fig. 32. The resulting formulas for $\hat{\theta}_x^{rig}$ and $\hat{\theta}_{\hat{y}}^{rig}$ are then the same as in 3.19. This approach for obtaining $\hat{\theta}_{\hat{x}}^{rig}$ contemplates the determination of a rigid body rotation around the $\hat{x}$-axis based on the linear interpolation between the two utmost locations of element node 3

$$\begin{aligned} \hat{\theta}_{\min\ \hat{x}}^{rig} = \frac{1}{\hat{\mathsf{y}}_3}\,(\hat{\mathsf{v}}_{\hat{z}_3} - \hat{\mathsf{v}}_{\hat{z}_1})\,, \quad \hat{\theta}_{\max\ x}^{rig} = \frac{1}{\hat{\mathsf{y}}_3}\,(\hat{\mathsf{v}}_{\hat{z}_3} - \hat{\mathsf{v}}_{\hat{z}_2})\,, \\ \hat{\theta}_{\hat{x}}^{rig} = \hat{\theta}_{\min\ \hat{x}}^{rig}\left(1 - \frac{\hat{\mathsf{x}}_3}{\hat{\mathsf{x}}_2}\right) + \hat{\theta}_{\max\ \hat{x}}^{rig}\frac{\hat{\mathsf{x}}_3}{\hat{\mathsf{x}}_2}. \end{aligned}$$

The last part is the determination of the shell strains $\boldsymbol{\varepsilon}^{t+\Delta t}$, appropriate Cauchy's stresses $\boldsymbol{\sigma}^{t+\Delta t}$, updated shell thickness $h^{t+\Delta t}$ and an integration of respective internal forces

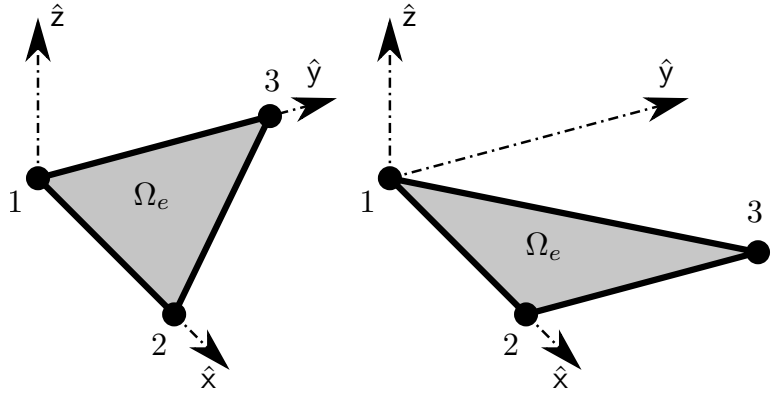


Figure 32: Element configuration for obtaining formula of rigid body rotation $\hat{\theta}_x^{rig}$.

$\boldsymbol{n}$ and $\boldsymbol{m}$. Cauchy's stresses are computed in the local (corotated) coordinate system

$$\begin{aligned}
\boldsymbol{\varepsilon}^{t+\Delta t} &= \boldsymbol{\varepsilon}^{t} + \int_t^{t+\Delta t} \dot{\boldsymbol{\varepsilon}}\left(\mathbf{u}^{def,t}\right)\,\mathrm{d}t = \boldsymbol{\varepsilon}^{t} + \Delta\boldsymbol{\varepsilon},\\
\boldsymbol{\sigma}^{t+\Delta t} &= \varrho\partial_{\boldsymbol{\varepsilon}^{t+\Delta t}}\Psi\left(\boldsymbol{\varepsilon}^{t+\Delta t}\right) = \mathbb{C}_{ijkl} : \boldsymbol{\varepsilon}^{t+\Delta t},\\
\int_t^{t+\Delta t}\left(\int_{\Omega_e} \mathrm{tr}\ \dot{\boldsymbol{\varepsilon}}\,\mathrm{d}v\right)\,\mathrm{d}t \rightarrow h^{t+\Delta t} &= h^t + \int_0^z \frac{\nu}{E}\left(\sigma_x^{t+\Delta t} + \sigma_y^{t+\Delta t}\right)\,\mathrm{d}z\\
&= h^t\left(1 + \frac{-\nu}{1-\nu}(\overline{\varepsilon}_x^{t+\Delta t} + \overline{\varepsilon}_y^{t+\Delta t})\right),
\end{aligned}$$

where the constitutive matrix $\mathbf{D}_{ij}$ related to Hooke's constitutive elasticity tensor $\mathbb{C}_{ijkl}$ for homogeneous isotropic linearly elastic material, where Poisson's ratio ν for most common materials falls into the range $(0, 0.5)$. Due to the symmetries $\mathbb{C}_{ijkl} = \mathbb{C}_{klij} = \mathbb{C}_{ijlk} = \mathbb{C}_{jilk}$, constitutive matrix $\mathbf{D}_{ij}$ has the following form:

$$\mathbb{C}_{ijkl} \rightarrow \mathbf{D}_{ij} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 & 0 & 0\\ \nu & 1 & 0 & 0 & 0\\ 0 & 0 & \dfrac{1-\nu}{2} & 0 & 0\\ 0 & 0 & 0 & \dfrac{5}{12}(1-\nu) & 0\\ 0 & 0 & 0 & 0 & \dfrac{5}{12}(1-\nu)\end{bmatrix}.$$

The internal forces $\boldsymbol{n}$ and $\boldsymbol{m}$ are the integrated from the appropriate stresses in $t+\Delta t$. Respective integration is expressed by the following formal form:

$$n_x = \int_{\frac{-h}{2}}^{\frac{h}{2}} \sigma_x\,\mathrm{d}z, \quad n_y = \int_{\frac{-h}{2}}^{\frac{h}{2}} \sigma_y\,\mathrm{d}z, \quad n_{xy} = \int_{\frac{-h}{2}}^{\frac{h}{2}} \sigma_{xy}\,\mathrm{d}z, \quad n_{xz} = h\sigma_{xz}, \quad n_{yz} = h\sigma_{yz},$$

$$m_x = -\int_{\frac{-h}{2}}^{\frac{h}{2}} z\sigma_x\,\mathrm{d}z, \quad m_y = -\int_{\frac{-h}{2}}^{\frac{h}{2}} z\sigma_y\,\mathrm{d}z, \quad m_{xy} = -\int_{\frac{-h}{2}}^{\frac{h}{2}} z\sigma_{xy}\,\mathrm{d}z.$$

Resultant element-centered forces are related to the local nodal forces through the principle of virtual power by performing one-point quadrature $\hat{\mathbf{f}} = A\mathbf{B}_m^T\hat{\mathbf{n}}_m$ and $\hat{\mathbf{m}} = A\left[\mathbf{B}_m^T\hat{\mathbf{m}} + \mathbf{B}_s^T\hat{\mathbf{n}}_s\right]$, thus

$$\begin{aligned}\left[\hat{\mathsf{f}}_{\hat{\mathsf{x}}_1} \quad \hat{\mathsf{f}}_{\hat{\mathsf{y}}_1} \quad \hat{\mathsf{f}}_{\hat{\mathsf{x}}_2} \quad \hat{\mathsf{f}}_{\hat{\mathsf{y}}_2} \quad \hat{\mathsf{f}}_{\hat{\mathsf{x}}_3} \quad \hat{\mathsf{f}}_{\hat{\mathsf{y}}_3}\right]^T &= A\mathbf{B}_m^T\left[\hat{\mathsf{n}}_{\hat{\mathsf{x}}} \quad \hat{\mathsf{n}}_{\hat{\mathsf{y}}} \quad \hat{\mathsf{n}}_{\hat{\mathsf{x}}\hat{\mathsf{y}}}\right]^T,\\ \left[\hat{\mathsf{m}}_{\hat{\mathsf{x}}_1} \quad \hat{\mathsf{m}}_{\hat{\mathsf{y}}_1} \quad \hat{\mathsf{m}}_{\hat{\mathsf{x}}_2} \quad \hat{\mathsf{m}}_{\hat{\mathsf{y}}_2} \quad \hat{\mathsf{m}}_{\hat{\mathsf{x}}_3} \quad \hat{\mathsf{m}}_{\hat{\mathsf{y}}_3}\right]^T &= A\mathbf{B}_m^T\left[\hat{\mathsf{m}}_{\hat{\mathsf{x}}} \quad \hat{\mathsf{m}}_{\hat{\mathsf{y}}} \quad \hat{\mathsf{m}}_{\hat{\mathsf{x}}\hat{\mathsf{y}}}\right]^T + A\mathbf{B}_s^T\left[\hat{\mathsf{n}}_{\hat{\mathsf{x}}\hat{\mathsf{z}}} \quad \hat{\mathsf{n}}_{\hat{\mathsf{y}}\hat{\mathsf{z}}}\right]^T.\end{aligned}$$

The remaining $\hat{\mathsf{z}}$ components of nodal forces $\hat{\mathbf{f}}_i$ are determinated by the equilibrium equations representing moment equilibrium about the local $\hat{x}$-axis equilibrium about the local $\hat{y}$-axis, and force equilibrium in the local $\hat{z}$-direction as follows:

$$\begin{aligned}\hat{\mathsf{m}}_{\hat{\mathsf{x}}_1} + \hat{\mathsf{m}}_{\hat{\mathsf{x}}_2} + \hat{\mathsf{m}}_{\hat{\mathsf{x}}_3} + \hat{\mathsf{y}}_3\hat{\mathsf{f}}_{\hat{\mathsf{z}}_3} &= 0,\\ \hat{\mathsf{m}}_{\hat{\mathsf{y}}_1} + \hat{\mathsf{m}}_{\hat{\mathsf{y}}_2} + \hat{\mathsf{m}}_{\hat{\mathsf{y}}_3} - \hat{\mathsf{x}}_3\hat{\mathsf{f}}_{\hat{\mathsf{z}}_3} - \hat{\mathsf{x}}_2\hat{\mathsf{f}}_{\hat{\mathsf{z}}_2} &= 0,\\ \hat{\mathsf{f}}_{\hat{\mathsf{z}}_1} + \hat{\mathsf{f}}_{\hat{\mathsf{z}}_2} + \hat{\mathsf{f}}_{\hat{\mathsf{z}}_3} &= 0.\end{aligned}$$

3.4.3 Numerics of Applied Contact Conditions

Among the first contributors who attempted to solve the contact phenomena using FEM were E.A. Wilson and B. Parsons (see [131], [97]). Later, the first implementation of contact conditions in numerical solvers was carried out. Among the first of them were DYNA2D and DYNA3D, using such an explicit approach for the direct integration of equations of motion, which is the main focus of this study.

In this context since the year 1990 a realistic adequate set of tools to address predictive vehicle safety design improvements has become available. The two main types of contacts are generally used for modeling vehicle safety structures. These are component contacts, such as *node-to-panel* contacts as well as *tied* contacts (see [52]).

Node-to-panel contacts are used in connection with crash analysis to ensure that Newton's third law of motion is not violated. Tied contacts are generally used to represent mechanical connections such as spot welds. Such numerics of the node-to-panel type of contact is considered here.

Frictionless normal contact constraints are imposed in the variational form of equations of motion throught the power of contact forces containing constitutive term related to equation 2.46

$$\delta\mathcal{U}_{\mathcal{P},\mathcal{C}} = \delta\left(\int_{\partial\mathscr{B}_C} \epsilon_N g(t)_N^2 \,\mathrm{d}A\right) = \int_{\partial\mathscr{B}_C} \epsilon_N g(t)_N \delta g(t)_N \,\mathrm{d}A, \tag{3.20}$$

where $\epsilon_N, \epsilon_N > 0$ denotes penalty parameter related to the constitutive parameter c_N in equation 2.46, and $g_N(t)$ denotes time dependent normal gap from equation 2.44

$$g_N(t) = \begin{cases}(\mathbf{x}_2(t) - \hat{\mathbf{x}}_1(t)) \cdot \hat{\mathbf{n}}_1 & \text{if } (\mathbf{x}_2(t) - \hat{\mathbf{x}}_1(t)) \cdot \hat{\mathbf{n}}_1 < 0\\ 0 & \text{otherwise.}\end{cases} \tag{3.21}$$

The resulting contact forces are then obtained by integrating the semidiscrete equation on the time interval $[\tau - \Delta\tau, \tau]$ as follows:

$$\mathbf{f}_{cnt}|_{\Omega_e} = \int_{\tau-\Delta\tau}^{\tau} \epsilon_N \mathbf{g}_N(t) \, \mathrm{d}\tau.$$

3.5 The Explicit Time Integration Algorithm

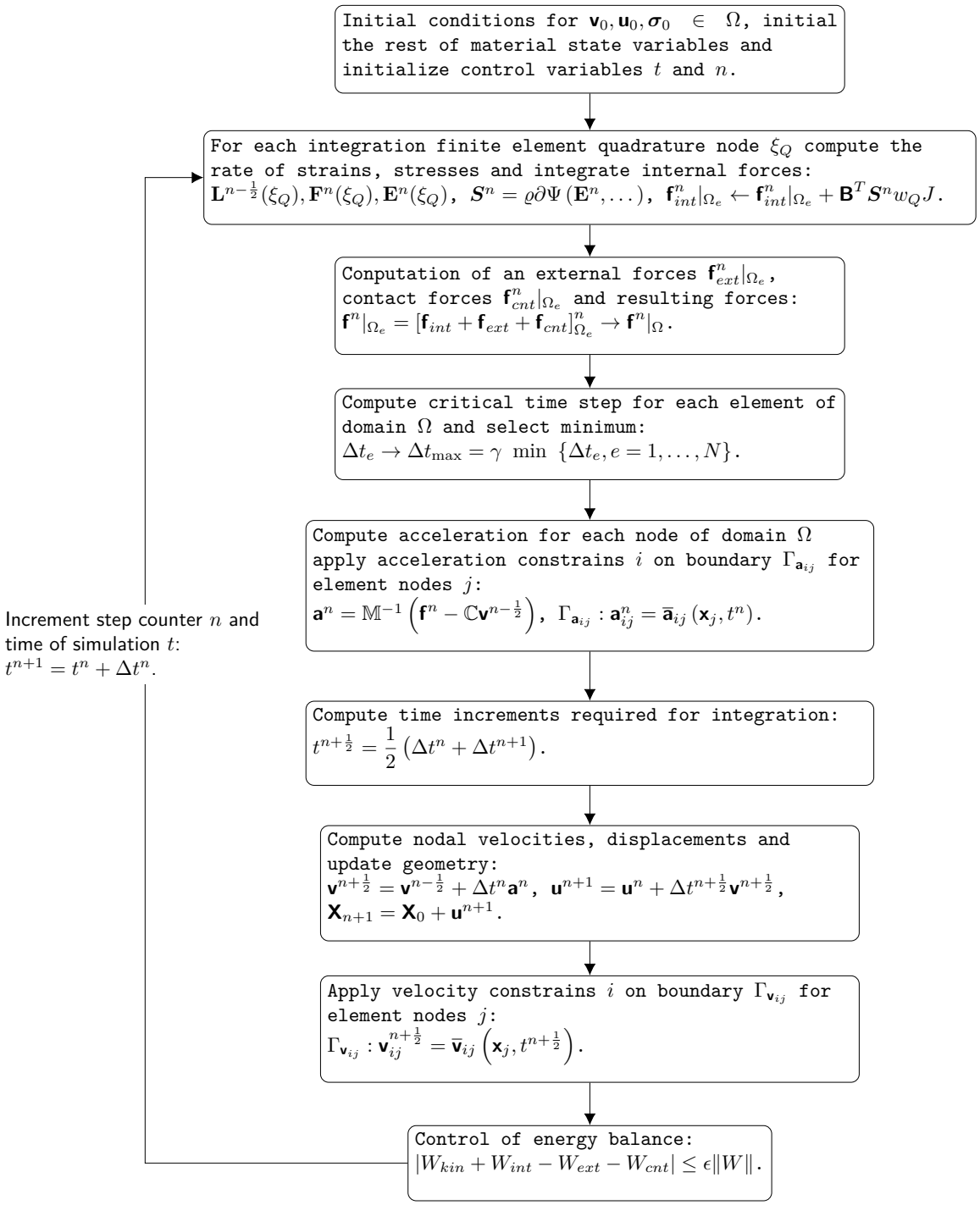


Figure 33: Flowchart of an explicit time integration algorithm.

The explicit form of the time integration algorithm has developed into a useful tool in nonlinear problems of transient dynamics. It does not require he construction of expensive tangential operators as noted in chapter 3.4.1. Thus, it allows a simple treatment of the different kind of nonlinearities. In the thesis, it involves the consideration of contacts and large rotating kinematics.

A distinct advantage is the fact that it does not require a large memory storage compared to the implicit time integration methods. On the other hand, a major disadvantage is then the need for a very small time increment, which makes the solution of a very large time domains harder. This is the reason why an explicit and implicit approach is often combined, where a fast, highly nonlinear phenomenon is solved by an explicit algorithm and consequently an implicit algorithm is used. However, this combination (explicit-implicit approach) is not considered in the thesis.

Explicit time integration algorithms state the equilibrium at time t^n and the displacement in the next step is obtained depending on the velocity and displacement of the previous step. The time integration of the discrete momentum equations does not require the solution of any equations.

In this work it is considered that the *central difference method* with variable time step as a popular method in computational mechanics. It is used for discretization velocities and accelerations in the terms of Lagrangian meshes (see [7], [83], [39]). In respective time integration algorithm it is necessary to control stable time step as the mesh deforms and the wave speed changes due to the stress. Respective algorithm is shown on Fig. 33, where the term $\mathbb{C}\mathbf{v}^{n-\frac{1}{2}}$ represents additional damping forces explicitly included in equations of motion. Such an expression would be important in the *dynamic relaxation* numerical method introduced in 1965 by A.S. Day [21], where the expected result is a steady state of a structure. In such a method accelerations and velocities are included for numerical reasons only. This even applies to mass coefficients represented by the mass $\mathbb{M}$ and also to damping coefficients represented by the damping $\mathbb{C}$, which have no physical meaning.

In selection of a mass and damping, it is necessary to ensure resulting transient characteristic of the structure to be close to the criticle damped system. Otherwise in the dissipative systems, the resulting steady state would not meet the result coming from a common implicit solution in statics of structures.

The member representing damping forces in numerical solver developed for a solution of problems in structural dynamics, is not considered. However, including the damping member in the developed solver is simply possible.

3.6 Numerical Stability

The control of numerical stability in the process of solving a set of non-linear second order ordinary differential equations by the explicit numerical integration algorithm represents a critical step due to its conditionally stable character. Possible instability then obviously leads to an exponential growth of the solution.

The large set of ordinary differential equations which are dealt and their strongly non-linear character disqualify a standard approach through the spectral analysis of the resulting sparse set of simultaneous equations to determine the critical integration time step except approach which is also used here.

For the previously mentioned reseasons, a simplified approach is used to estimate the upper bound of the integration time step value, which is based on the acoustic wave propagation process through an isotropic homogeneous elastic medium. This phenomenon is generally formulated by the wave equation. The resulting condition is generally known as Courant-Friedrichson-Lewy condition of stability (see [20]).

$$\varrho\ddot{\mathbf{u}} = \operatorname{div}\boldsymbol{\sigma}, \quad \boldsymbol{\sigma} = \lambda\left(\operatorname{tr}[\boldsymbol{\varepsilon}]\right)^2 + 2\mu\boldsymbol{\varepsilon}, \quad \varepsilon = \frac{1}{2}\left(\operatorname{grad}\mathbf{u} + (\operatorname{grad}\mathbf{u})^T\right),$$

where $\boldsymbol{\sigma}$ is Cauchy's stress, λ and μ (represents the shear modulus of elasticity) are Lamé's constants. The resulting equation is then known as a Navier equation whose compact write has the following form written in index notation (grad $f = \nabla f = \nabla_i f = \partial_i f$, div $\boldsymbol{f} = \nabla\cdot\boldsymbol{f} = \nabla_i\boldsymbol{f} = \partial_i\boldsymbol{f}$, curl $\boldsymbol{f} = \nabla\times\boldsymbol{f} = \epsilon_{ijk}\nabla_j\boldsymbol{f}_k = \epsilon_{ijk}\partial_j\boldsymbol{f}_k$, ϵ_{ijk} is Levi-Civita symbol):

$$\varrho\ddot{\mathbf{u}} = (\lambda+\mu)\partial_i(\partial_i\mathbf{u}_i) + \mu\partial_i\partial_i\mathbf{u}. \tag{3.22}$$

If the field of displacements $\mathbf{u}$ meets the Helmholtz theorem assumptions, then $\mathbf{u}$ can be decomposed into the sum of a gradient of a scalar potential ϕ and rotation of a vector potential ψ

$$\mathbf{u} = \partial_i\phi + \epsilon_{ijk}\partial_j\psi_k. \tag{3.23}$$

By the subsequent substitution of 3.23 to 3.22 it leads to the following equation

$$\begin{aligned}
\varrho\partial_{tt}\left(\partial_i\phi + \epsilon_{ijk}\partial_j\psi_k\right) &= (\lambda+\mu)\partial_i\Big(\partial_i\left(\partial_i\phi + \epsilon_{ijk}\partial_j\psi_k\right)\Big)_i + \mu\partial_i\partial_i(\partial_i\phi + \epsilon_{ijk}\partial_j\psi_k) = \\
&= (\lambda+\mu)\Big(\partial_i\Big(\partial_i(\partial_i\phi)_i\Big) + \partial_i\Big(\partial_i(\epsilon_{ijk}\partial_j\psi_k)_i\Big)\Big) + \mu\partial_i\partial_i(\partial_i\phi) + \mu\partial_i\partial_i(\epsilon_{ijk}\partial_j\psi_k) = \\
&= (\lambda+\mu)\partial_i(\partial_i\partial_i\phi) + \mu\partial_i(\partial_i\partial_i\phi) + \mu\epsilon_{ijk}\partial_j(\partial_i\partial_i\psi)_k = \\
&= \partial_i\left((\lambda+2\mu)\partial_i\partial_i\phi - \varrho\ddot{\phi}\right) + \epsilon_{ijk}\partial_j\left(\mu\partial_i\partial_i\psi - \varrho\ddot{\psi}\right)_k = 0.
\end{aligned}$$

The given equation makes sense if both its members are equal to zero

$$\partial_i\underbrace{\left((\lambda+2\mu)\partial_i\partial_i\phi - \varrho\ddot{\phi}\right)}_{=0} + \epsilon_{ijk}\partial_j\underbrace{\left(\mu\partial_i\partial_i\psi - \varrho\ddot{\psi}\right)_k}_{=0} = 0,$$

therefore we get the two following equations

$$\begin{aligned}
(\lambda+2\mu)\partial_i\partial_i\phi - \varrho\ddot{\phi} &= 0 \Rightarrow \ddot{\phi} = \frac{(\lambda+2\mu)}{\varrho}\partial_i\partial_i\phi = c_l^2\partial_i\partial_i\phi \\
\mu\partial_i\partial_i\psi - \varrho\ddot{\psi} &= 0 \Rightarrow \ddot{\psi} = \frac{\mu}{\varrho}\partial_i\partial_i\psi = c_s^2\partial_i\partial_i\psi,
\end{aligned}$$

where c_l and c_s are longitudinal (compression) and shear waves respectively. Resulting expressions are

$$c_l = \sqrt{\frac{\lambda + 2\mu}{\varrho}} \quad \text{and} \quad c_s = \sqrt{\frac{\mu}{\varrho}}.$$

Let us consider an isotropic homogeneous elastic medium for which we can state the velocity of longitudinal wave propagation for a plane stress state as follows:

$$\mu = \frac{E(1-\nu)}{2(1-\nu^2)}, \quad \lambda = \frac{E\nu}{(1-\nu^2)},$$

$$c_l = \sqrt{\frac{E}{\varrho\,(1-\nu^2)}}$$

where E is Young's modulus of elasticity, ϱ is a material density and ν is the Poisson's ratio respectively.

As the next step let us consider the length l that characterizes the observed dimension of respective matter, then from the velocity of the longitudinal wave c_l the time of wave propagation through the respective distance can be simply computed as follows:

$$t = \frac{l}{c_l}.$$

For the purpose of estimation of a stable time step, the approach from [58] is also incorporated, where detailed derivation of the resulting formulas can be found in appendix B of respective article. Estimation of the maximum stable time step is based on the calculation of the maximum eigen-frequencies of the element

$$\omega_S^2 = \frac{1}{M}\left(\frac{c_S a_1}{A} + \frac{c_S A}{4\alpha}\right), \omega_B^2 = \frac{D}{4\alpha M A}\left(R_1 + \sqrt{R_1^2 - 4\left(1-\nu^2\right)R_2^2}\right), \omega_M^2 = \frac{12\alpha\omega_B^2}{h^2},$$

The parameters used in calculation of eigen-frequencies are defined as follows:

$$\begin{aligned}
G &= \frac{E}{2\,(1+\nu)} \\
c_S &= \kappa_S G h \\
R_1 &= y_{24}^2 + y_{31}^2 + x_{24}^2 + x_{31}^2, R_2 = y_{31}x_{24} - y_{24}x_{31} = 2A, R_3 = \sqrt{R_1^2 - 4R_2^2} \\
a_1 &= \frac{1}{4}\varrho A h \\
D &= \frac{E h^3}{12\,(1-\nu^2)} \\
\alpha &= \frac{1}{12}\left(h^2 + A\right) \\
x_{ij} &= x_i - x_j, y_{ij} = y_i - y_j,
\end{aligned}$$

where h is element thickness, A is element area and κ_S is shear correction factor. Then the final value of the eigen-frequency for selected element is obtained according to the following formula

$$\omega = \min\{\omega_S, \omega_B, \omega_M\}, \quad \omega_{\max} = \max\{\omega_e, e = 1 \dots N\}.$$

To determine the value of the maximum (critical) time integration step $\Delta t_{\max}$ it is necessary to select the element with the smallest characteristic dimension $l_{\min}$ or maximu eigen-frequency $\omega_{\max}$, then

$$\Delta t \leq \Delta t_{\max} = \gamma \frac{l_{\min}}{c_l} = \gamma \frac{2}{\omega_{\max}},$$

where γ denotes a safety factor, typically with value from interval $[0.7, 0.9]$ and $\omega_{\max}$ denotes the highest eigen-frequency $\omega_{\max}$.

In numerical model with various types of finite elements the critical time step Δt is computed separately for each element type. The value of Δt is calculated either based on the longitudinal wave propagation or derived eigen-frequency of the finite element as in the previous case for a C^0 triangular finite element. Subsequently the smallest value of Δt is selected, then $\Delta t_{\max} = \gamma \min\{\Delta t_e, e = 1 \dots N\}$. The calculation of a critical time step $\Delta t_{\max}$ is repeated during the time integration process to ensure numerical stability of respective dynamic simulation.

In connection with the above, M.N. Neal and T. Belytschko in [89] proposed an additional stability control mechanism for nonlinear problems based on an energy balance check. This concerns primarily to the problems with a dissipative character. Since material model used does not provide such characteristic, thus a simple energy balance control is additionaly used only.

$$\begin{aligned}
W_{kin}^{n+\frac{1}{2}} &= \frac{1}{2}\left(\mathbf{v}^{T,n+\frac{1}{2}}\mathbb{M}\mathbf{v}^{n+\frac{1}{2}}\right) \\
W_{kin}^{n} &= \frac{1}{2}\left(W_{kin}^{n+\frac{1}{2}} + W_{kin}^{n-\frac{1}{2}}\right) \\
W_{ext}^{n} &= W_{ext}^{n-1} + \frac{\Delta t_{\max}^{n}}{2}\mathbf{v}^{T,n+\frac{1}{2}}\left(\mathbf{f}_{ext}^{n} - \mathbf{f}_{ext}^{n-1}\right) \\
W_{int}^{n} &= \int_{\Omega_e} \boldsymbol{\sigma}^n : \boldsymbol{\varepsilon}^n \, \mathrm{d}V \\
W_{cnt}^{n} &= W_{cnt}^{n-1} + \frac{\Delta t_{\max}^{n}}{2}\mathbf{v}^{T,n+\frac{1}{2}}\left(\mathbf{f}_{cnt}^{n} - \mathbf{f}_{cnt}^{n-1}\right),
\end{aligned}$$

where W_{kin}^n is the current kinetic energy of the system, W_{ext}^n is a current external work performed on the system, W_{int}^n is the current internal energy for an element and W_{cnt}^n is the current work of contact forces, respectively. Then an energy balance control has the form

$$|W_{kin} + W_{int} - W_{ext} - W_{cnt}| \leq \epsilon \|W\|,$$

where and $\|W\|$ is a measure of the total energy of the system and ϵ is a suitable tolerance.

3.7 Summary of Chapter

In this chapter the numerical model which is being worked with is introduced. It also inclues the rough theory of variational calculus, which is critical for the FEM based application to a numerical solution of problems in the field of structural dynamics. The approach to the model's numerics contains a certain amount of generality in some of formulations used.

The introduced effective one-point quadrature C^0 triangular shell finite element of Reissner-Mindlin type and the algorithm for explicit numerical integration of equations of motion is implemented in a numerical hybrid-parallel testing solver, which will be further presented in a later chapters.

Chapter 4

Searching in the Euclidean Space

For the purpose of finding a solution to many problems within the scope of computer science, the respective problem can often be transformed into the task of searching in an appropriate mathematical space. It often leads to a searching in a trees or graphs that can be simply assembled and searched due to the discrete character of the data whose bindings they represent. If these data come from the space of real numbers $\mathbb{R}$, then such the problem of serching is somewhat more complicated. One of the application areas in the CM that requires an effective way of searching in such a space of real numbers is the CCM.

Effective contact implementation requires algorithmic strategy dependent on a complex knowledge from both an analysis of the relevant mathematical-physical structure of a problem and an algorithmic maturity of the FEA software developer, as is often the case in CM.

Due to the considerated generality of a model of contact, the algorithm used must be applied to all the FE nodes of the FE model, which is the most computationally demanding process. Due to the rapid improvement of modern computer technology, one can today apply the tools of computational mechanics to simulate contact/impact mechanisms numerically. Numerical simulation of the Euro NCAP frontal impact of a car with an initial speed of 64 km/h against a deformable barrier, human joints or implantation of teeth in biomechanics are examples of the application of non-linear boundary conditions. More than any of the areas of CM, contact problem is the most closely related to the field of theoretical computer science through the branch of computational geometry.

The solution to such a problem originates from the philosophy of modern database software systems that currently allow us to collect huge amounts of data such as images, music or videos (eg social networks as Facebook, Twitter, atc.), personal data in banking systems, data from government administration, medical or astronomical data (e.g. SETI, abbreviation of the Search for Extra-Terrestrial Intelligence), data from other enterprise software systems (e.g. PDM, abbreviation of the Product Data Management) or for graphical (gaming) engines with the focus on the organization of geometrical data structures.

As a critical task of managing any huge datasets, we can then designate a search of

a set of such data that meets our specific requirements. Here, it is important to realize the fact that such a dataset may contain billions of items that incessantly grows (e.g. social networks or banking software systems, etc.). Thus, data addition and removal is also inherently associated with this issue.

A naive way to perform contact detection is an exhaustive check of each body against all the others without taking advantage of available information about the spatial distribution of the bodies. Such a computational procedure has very expensive time complexity $O(n^2)$, where n denotes items in dataset. For the 10^6 of finite element nodes it would mean 10^{12} of contact checking operations per each integration time step Δt in the worst case, which is unuseful.

Here we get to the requirement that any action that we would perform over such a data structure would be done in logarithmic $O(\log n)$ or polynomial $O(P(n))$, preferably in linear $O(n)$ or constant $O(1)$ time complexity. The last type of time complexity of algorithm is practically inaccessible for such search problems under the consideration, and the penultimate one only with a number of restrictions, those however, limit the generality of the algorithm. The search algorithm used here is a logarithmic type of time complexity.

The solution to the problem of contact of solids then theoretically falls within the area of a so-called *nearest neighbor* (NN) search. The core of such an algorith is here defined as a collection of n objects (FE nodes) that build a data structure, which provides objects (FE nodes, FEs, etc.) in the time as fast as possible based on the NN *query*.

4.1 Nearest Neighbor Searching

The NN problem is of major importance in a number application areas, including data compression, databases and data mining, information retrieval, searching image datasets, machine learning, pattern recognition, statistics, and also data analysis.

Given a set S of points in a high-dimensional space, construct a data structure which given any query point q finds the point in S closest to q. Generally the dimensionality of the points is usually large as well. The NN problem is an example of a large class of *proximity problems*. Problems whose definitions involve the notion of distance between the input points. Many of these problems were among the first investigated in the branch of computational geometry mentioned before.

The NN problem can be solved with $O(\log n)$ time per query, using only $O(n)$ storage. However, the growing dimension of the problem leads to growth of algorithm inefficiency. More specifically, their space or time requirements grow *exponentially* in the dimension. The NN problem has a solution with $O(d^{O(1)} \log n)$ query time using $n^{O(d)}$ storage space.

The lack of success in removing the exponential dependence on the dimension d led to conjecture that no efficient solutions exists for these problems when the dimension is sufficiently large. Removing the exponential dependence on the problem dimension resulted in its approximation, which, at the end, is the best explanation of the core of NN search algorithm. The algorithm is allowed to report any point within distance $(1 + \epsilon)$ times the distance from q to point p.

The subset of the NN searching problem is the range-searching problem, which is one of the central problems in the computational geometry. Let S be a set of n points in $\mathbb{R}^d$, and let $\mathbf{R}$ (ranges) be a family of subsets of $\mathbb{R}^d$. The target is to preprocess S into a data structure, so that for a query range γ, the points in $S \cap \gamma$ can be reported or counted efficiently. Range counting and range reporting are just two instances of range-searching queries.

For each point $p \in S$, we assign a weight $w(p) \in \mathbf{S}$, where $(\mathbf{S}, \oplus)$ is a commutative semigroup, $w : S \to \mathbf{S}$. For any subset $S' \subseteq S$, let $w(S') = \sum_{p \in S'} w(S)$. Addition is taken over the semigroup $\mathbf{S}$. For a query range $\gamma \in \mathbf{R}$, let compute $w(S \cap \gamma)$.

Geometric range-searching data structures are constructed by subdividing space into several regions with some properties and recursively constructing a data structure for each such region. Range queries are answered with such a data structure by performing a *depth-first search* through the resulting recursive space partition. Most of the range-searching data structures are based on the general scheme, where for the given a point set S the structure precomputes a family $\mathbf{F} = \mathbf{F}(S)$ of canonical subsets of S and store the weight $w(C) = \sum_{p \in C} w(p)$ of each canonical subset $C \in \mathbf{F}$. Then for a query range γ, the query procedure determines a partition $C_\gamma = C(S, \gamma) \subseteq \mathbf{F}$ of $S \cap \gamma$ and adds the weights of the subsets in C_γ to compute $w(S \cap \gamma)$. It is a *decomposition scheme* of a such data structures. Each canonical subset $C = \{p_i | i \in \mathbf{I}\} \in \mathbf{F}$, where $\mathbf{I} \subseteq \{1, \ldots, n\}$, corresponds to the generator $\sum_{i \in \mathbf{I}} x_i$. Computation of C_γ and storage of the weights of canonical subsets depends on the model of computation and on the specific range-searching problem.

In practice, linear-size data structures are predominantly used for these purposes, including mainly B-trees and their variants for answering one-dimensional range queries. For high-dimensional range queries, recursive partitioning into the specific spaces is used. As a typical example of that spaces in a plane are rectangles, which are used to build a tree. Example of some the simplest data structures are *Quadtree* (2D mapping) or *Octree* (3D mapping), which are shown in the Fig. 41.

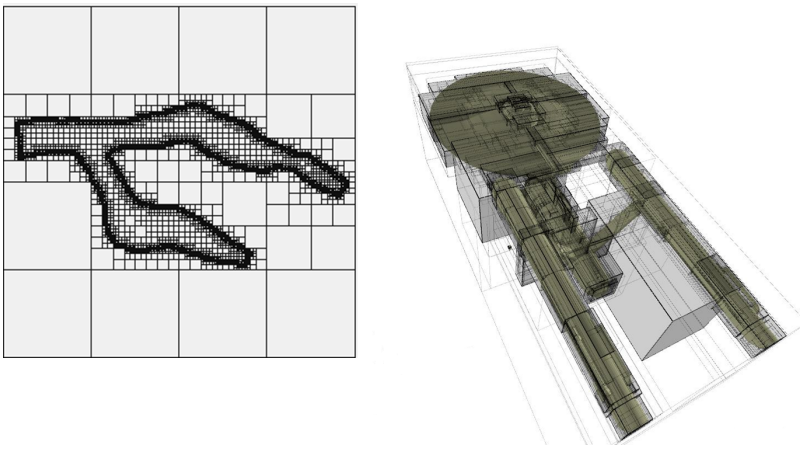


Figure 41: Examples of the Quadtree and Octree map.

Here we are getting to the data structure that is primarily used in this work and it is

described later. These are *kd*-trees that oppose the quadtree. It partitions the enclosing rectangle into two rectangles drawing a horizontal or vertical line and partitioning each of the two rectangles independently.

The performance of an appropriate data structure is measured by the time spent in answering a query, called the *query time*, by the size of the data structure, and by the time constructed in the data structure, called the *preprocessing time*. The respective data structure is usually constructed only once. However, the situation changes dramatically when the entire data changes dynamically. This behavior is typical for the numerical solution using FEM in the case of an analysis of geometrically nonlinear behavior of a solid structure, where contact can take a place in a completely general form.

4.2 Range Searching in the d-dimensional Space Using the kd-tree Data Structure

The first such data structure, called *kd*-trees (*d*-dimensional tree or *dd*-tree) was introduced by J. Bentley from Stanford University CA in 1975 [9] as a powerful extension of one-dimensional trees. Since then, many other multidimensional data structures have been developed. It mainly includes R-trees and its various mutations used preferably in spatial databases (see [80]).

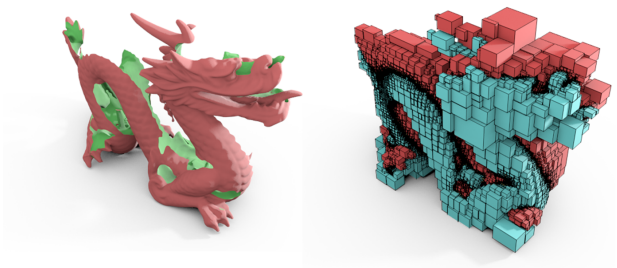


Figure 42: Example of the *kd*-tree map.

The letter k in the name of data structure denotes the dimensionality of the space being represented. The *kd*-tree is a binary tree where the underlying space is partitioned on the basis of the value of just one attribute at each level of the tree instead of on the basis of the values of all d attributes, thereby making d tests at each level, as is the case for the *quadtree*. The distinction from the *quadtree* is that, in the *kd*-tree, only one attribute value is tested when determining the direction in which a branch is to be made.

Theoretical description

This structure is theoretically well described by K. Mehlhorn in [84]. At every level of a *kd*-tree we split the set according to one of the coordinates, where we go through in a cyclic order. Let $U_i, 0 \leq i < d$, be an ordered set and let $U = \{U_0 \times \cdots \times U_{d-1}\}$. An

element $p_x = \{x_0, \ldots, x_{d-1}\} \in U$ is also called point (in geometry) or record (in database). Components x_i are called coordinates or attributes.

The region searching problem is specified by a set $\Gamma \subseteq 2^U$ of regions in U. Four types of region searching are usually used. Here we use the orthogonal range querie, where Γ is the set of hypercubes (in general) in U

$$\Gamma = \{R; R = [\ell_0, h_0] \times [\ell_1, h_1] \times \cdots \times [\ell_{d-1}, h_{d-1}] \text{ where } \ell_i, h_i \in U_i \text{ and } \ell_i \leq h_i\}.$$

Definition 4.2.1. Let $S \subseteq \{U_0 \times \cdots \times U_{d-1}\}$, $|S| = n$. A kd-tree for S (starting at coordinate i) is defined as follows

1. If $d = n = 1$ then it consists of a single leaf labelled by the unique element $x \in S$.
2. If $d > 1$ or $n > 1$ then it consists of a root labelled by some element $d_i \in U_i$ and three subtrees $T_<$, $T_=$ and $T_>$. Here $T_<$ is a kd-tree starting at coordinate $(i+1)$ mod d for set $S_< = \{p_x \in S; p_x = (x_0, \ldots, x_{d-1})$ and $x_i < d_i\}$, $T_>$ is a kd-tree starting at coordinate $(i+1)$ mod d for set $S_> = \{p_x \in S; p_x = (x_0, \ldots, x_{d-1})$ and $x_i > d_i\}$ and $T_=$ is a $(d-1)d$-tree starting at coordinate i mod $(d-1)$ for set $S_= = \{p_x \in S; p_x = (x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{d-1}); p_x = (x_0, \ldots, x_{i-1}, d_i, x_{i+1}, \ldots, x_d) \in S\}$.

Definition 4.2.2.

(i) Let T be a kd-tree and let v be a node of T. Then $S(v)$ is a set of leaves in the subtree with root v_r, $d(v)$ is the depth of node v, and $sd(v)$, the number of pointers $ptr_{<-}$ and $ptr_{>-}$ on the path from the root v_r to v, is the strong depth of v. Node x is the proper son of node v if it is son via a $ptr_{<-}$ or $ptr_{>-}$ pointer.

(ii) A kd-tree is ideal if $|S(x)| \leq \dfrac{|S(v)|}{2}$ for every node v and all proper sons x of v.

Ideal kd-tree are a generalization of perfectly balanced $1d$-trees.

Lemma 4.2.3. Let T is an ideal kd-tree for set S, $|S| = n$.

(i) $d(v) \leq d + \log n$ for every node v of T.

(ii) $sd(v) \leq \log n$ for every node v of T.

Proof. (i) follows from (ii) and the fact that at most d $ptr_{=-}$ can be on the path to any node v. Part (ii) is immediate from the definition of ideal tree. □

Theorem 4.2.4. Let $S \subseteq U = \{U_0 \times \cdots \times U_{d-1}\}$, $|S| = n$.

(i) An exact match query in an ideal kd-tree for S takes time complexity $O(d + \log n)$.

(ii) An ideal kd-tree for S can be constructed in time complexity $O(n(d + \log n))$.

Proof.

(i) Immediate from lemma 4.2.3/(i).

(ii) We describe a procedure which constructs ideal kd-tree in time complexity $O(n(d + \log n))$. Let $S_0 = \{x_0; (x_0 \ldots, x_{d-1}) \in S\}$ is the multi-set of 0-th coordinates of S. We use the linear time median algorithm to find the median d_0 of S_0. d_0 will be the label of the root. Then clearly $|S_<| \leq \frac{|S|}{2}$ and $|S_>| > \frac{|S|}{2}$ where $|S_<| = \{p_x \in S; x_0 < d_0\}$ and $|S_>| = \{p_x \in S; x_0 > d_0\}$. We use the same algorithm recursively to construct kd-tree for $S_<$ and $S_>$ (starting at coordinate 1) and a $(d-1)d$-tree for $S_=$. This algorithm will clearly construct an ideal kd-tree T for S. The bound on the running time can be seen as follows. In every node v of T we spend $O(|S|)$ steps to compute the median of a set of size $|S(v)|$. Furthermore, $S(v) \cap S(w) \neq \emptyset$ if v and w are nodes of the same depth and hence $\sum_{d(v)=l} |S(v)| \leq n$ for every l, $0 \leq l \leq d + \log n$. Thus the running time is bounded by

$$\sum_{v \in T} O(|S(v)|) = O\left(\sum_{0 \leq l \leq d + \log n} \left(\sum_{d(v)=l} |S(v)| \right) \right) = O(n(d + \log n)).$$

□

kd-tree Algorithms for Building and Searching

For the purpose of contact detection, the two algorithms are considered to work with the *kd*-tree data structure. It consists of its assembly and the subsequent usage to obtain a set of nodes falling within the respective searching range query γ of the examined node belonging to that appropriate finite element.

The first part consists of an assembly of the *kd*-tree map T from the set of nodes S_p, which in its general form for the d-dimensions represents the following pseudocode.

Algorithm 1: Pseudocode of a building the *kd*-tree map.

Input : S_p; $level \in \{0, \ldots, d-1\}, level, d \in \mathbb{N}$ *is used to partition* S_p
Output: $v_{p_x} \in T$

Function `build_kd_tree`$(S_p, level)$
  $S_< = S_> = \emptyset$;
  for $p_{x,i}$ **in** $p_{x,i} = (x_{i,0}, \ldots, x_{i,d-1}) \in S_p$ **do**
    $\mu = \sum_{p_{x,i} \in S_p} \frac{x_{i,k}}{|S_p|}, k \in \{0, \ldots, d-1\}$;
    if $x_{i,0} \leq \mu$ **then**
      $S_< \cup \{p_{x,i}\}$;
    else
      $S_> \cup \{p_{x,i}\}$;
    end
  end
  Make $v_{ptr_{<-}}$ *left the left child of* v*, and make* $v_{ptr_{>-}}$ *right the right child of* v.;
  $v_{ptr_{<-}} \leftarrow$ `build_kd_tree`$(S_<, (level+1) \bmod d)$;
  $v_{ptr_{>-}} \leftarrow$ `build_kd_tree`$(S_>, (level+1) \bmod d)$;
  return v_{p_x};

Since the overall contact detection algorithm considers the topology mapping of the discretized model that takes place at each integration time step, deletion of nodes algorithm and tree balancing algorithm are not needed. The only other one algorithm is the algorithm of an application of the range-searching query γ to the *kd*-tree map. In the algorithm we traverse the *kd*-tree, but visit only nodes whose region is intersected by the query rectangle. When a region is fully contained in the query rectangle, we can report all the points stored in its subtree. When the traversal reaches a leaf, we have to check whether the point stored at the leaf is contained in the query region and, if so, report it (see [10]). Range-searching in a *kd*-tree map represents the following last pseudocode.

Algorithm 2: Pseudocode of a range-searching in the *kd*-tree map.

Input : $v \in T$ *root node*; $\gamma \in \mathbf{R}$ *range-searching query region*
Output: $C_\gamma \subseteq \mathbf{F}$ *subset of nodes meeting the query* γ*, which means points at leaves below* $v \subseteq (\gamma \cap S_p)$

1 **Function** `search_kd_tree`(v, γ)
2 **if** *v is a leaf* **then**
3 *Report the point* $p_x \in v$ *if* $v \subseteq \gamma$;
4 **return** $C_\gamma \cup \{p_x\} \in S_p$;
5 **else if** $reg(v_{ptr_{<-}}) \subseteq \gamma$ **then**
6 *Reports all the points from the subtree rooted at leaves* $v_{ptr_{<-}}$;
7 **return** $C_\gamma \cup (T_<(v_{ptr_{<-}}) \rightarrow \{p_{x,i}\} \in S_p)$;
8 **else if** $(reg(v_{ptr_{<-}}) \cap \gamma) \neq \emptyset$ **then**
9 **return** $C_\gamma \cup$ `search_kd_tree`$(v_{ptr_{<-}}, \gamma) \rightarrow \{p_{x,i}\} \in S_p$;
10 **else if** $reg(v_{ptr_{>-}}) \subseteq \gamma$ **then**
11 *Reports all the points from the subtree rooted at leaves* $v_{ptr_{>-}}$;
12 **return** $C_\gamma \cup (T_>(v_{ptr_{>-}}) \rightarrow \{p_{x,i}\} \in S_p)$;
13 **else if** $(reg(v_{ptr_{>-}}) \cap \gamma) \neq \emptyset$ **then**
14 **return** $C_\gamma \cup$ `search_kd_tree`$(v_{ptr_{>-}}, \gamma) \rightarrow \{p_{x,i}\} \in S_p$;
15 **return** $C_\gamma \cup \emptyset$;

4.3 Summary of Chapter

In this chapter an algorithm for consideration of non-linear boundary conditions represented by the contact phenomena is introduced.

Compared with older algorithms for a solution of contact problem, the proposed algorithm uses topology mapping based on the *kd*-tree data structure applied in database software systems or graphical engines for quick processing of the range-searching query over the *d*-dimensional data. The developed contact detection algorithm is useful not only in such problems under the consideration but also in many other application areas that involve contact detection between parts of the system that is simulated such as path planning in robotics, in molecular dynamics or in phenomenas in astrophysics, etc.

Subsequent post-processing represented by the computation and application of induced contact forces within the explicit integration algorithm meets the standard approach usually applied in a number of commercial software packages for the numerical simulation of crash tests by FEA in the automotive and aviation industries, and also in traffic structure engineering for the design of a road restraint systems.

Chapter 5

Analysis of the Macro Entity Interaction Multigraph

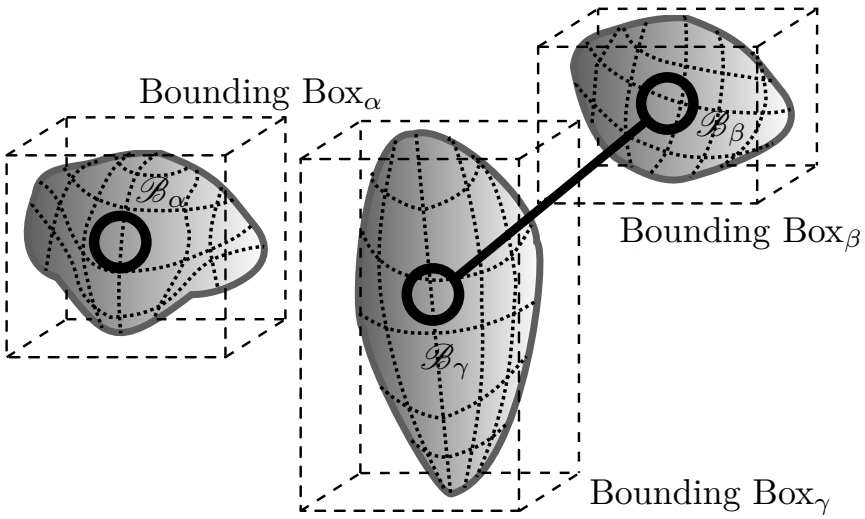


Figure 51: The Macro Entity Interaction Multigraph (macro body contact interaction).

The proposed algorithm for the purpose of hybrid-parallel computational processing strongly relates to the analysis of the motion the individual macro elements in a space representing the structures interacting with each other through the contact forces.

Nonlinear dynamics is often externally demonstrated by the chaotic behavior providing the so-called *strange attractors*. As the source of chaotic behaviour, is here, provided by mutually interacting deformable bodies, which generate special types of combinatorial sequences. Such a combinatorial sequence can be defined by so-called *unoriented multigraph*. The topology of such a multigraph can be defined by the spatial distribution of the interacting bodies. The original spatial multigraph can be isomorfly mapped into a arbitrary planar form. The nodes in the multi-graph represent all the contained bodies

(macro entities) and the graph edges represent contact interaction between the bodies. For the purpose of further analysis of such a problem for parallel computing, the term *Macro Entity Interaction Multigraph* (MEIM) is introduced. The related study of multigraphs has not been treated as extensively as simple graphs in the literature.

The MEIM assembly is performed through the range searching queries to the kd-tree data structure used for mapping the spatial data as presented in chapter 4. The MEIM analysis itself is further based on the *depth-first search* (DFS) algorithm to find *connected subgraphs* representing individual clusters of touching bodies.

A technique of DFS has been widely used for finding solutions to problems in combinatorial theory as well as in the other fields especially related to computer science. The DFS algorithm has been known since the nineteenth century as a technique for threading mazes (see [79]). Later, e.g. the paper of Hopcroft and Tarjan (see [47]) recognized DFS as a powerful technique for solving various graph problems.

The assembly and analysis of the MEIM is only required while performing a numerical simulation in the scope of a computer network where it is no longer possible to take advantage of the shared memory address space. Another problem is the time latency caused by remote network communication, which is considerably longer than the same one within one workstation. Therefore, it is necessary to limit the data flows to the minimum and to ensure the maximum computational utilization of individual workstations within the computer network. A similar approach uses various methods of domain decomposition such as the FETI based methods for solving large numerical models. However, the proposed approach does not require additional expensive algorithms for domain decomposition, and it is also designed to address impact problems within the framework of explicit dynamics. As a result, the entire computational process is not so error prone compared to the computational process requiring the utilization of an additional complex algorithms.

5.1 The Graph Theory

For the purpose of the applied algorithm, it is first necessary to define the relational structure representing the MEIM. The general definition of the unoriented graph as follows (see [13])

Definition 5.1.1. The (general unoriented) *graph* G is an ordered triple of disjoint sets $G = (V, E, \psi)$ consisting of a nonempty set V of vertices, a set E, disjoint from V, of edges, and an incidence function ψ_G that associates with each edge of G an unordered pair of vertices of G. If ϵ is an edge and u and v are vertices such that $\psi_G(e) = (u, v)$, the e is said to *join* u and v. The vertices u and v are called the *ends* of e. The sum of edges is denoted $\sum_{e \in G} e = \epsilon$ and sum of vertices is denoted $\sum_{v \in G} v = \nu$.

Definition 5.1.2. To any graph G corresponds a $\nu \times \epsilon$ matrix called the incidence matrix of G. The incidence matrix of G is the matrix $\mathbf{M} = [m_{ij}]$, where $m_{ij} : m_{ij} \in \mathbb{N}$ is the number of times that v_i and e_j are incident.

The DFS algorithm requires to have a graph in the form of a so-called *adjacency matrix* (map) as a very basic and very common graph representation. It is always a symmetric square matrix with entries 0 and 1, with 0s on the main diagonal.

Definition 5.1.3. Let $G = (V, E, \psi)$ be a graph with ν vertices. Denote the vertices by $v_1, v_2, \ldots, v_n$. The *adjacency matrix* of G, with respect to the chosen vertex numbering, is an $\nu \times \nu$ matrix $\mathbf{A} = [a_{ij}]$ defined by the following rule:

$$a_{ij} = \begin{cases} 1 & \text{if } \psi_G(e_k) = (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Since the case of MEIM structure relates to the multigraph, so the appropriate definition of multigraph is as follows:

Definition 5.1.4. A *multigraph* is a set of vertices V, a set of edges E, and function $f : E \to \{\psi_G(e) = (u, v) \in V \wedge u \neq v\}$. If $e_1, e_2 \in E$ are such that $\psi_G(e_1) = \psi_G(e_2)$, then e_1 and e_2 are parallel (multiple) edges.

A multigraph (multiple or multivariate graph) consists of a set of bodies (vertices), and a collection of contact relations (edges) that specify how bodies contact each other. Multigraph data structures can be observed directly and are common in contexts where several edges can be mapped on the same vertex pair. From the nature of the problem being dealt with, where the edges between the nodes represent the contact relationship between the individual bodies in the 3D space, it is suitable to map such multigraph into the 2D plane. It is related to the isomorphic mapping of a graph as follows:

Definition 5.1.5. Graphs G_{3D} and G_{2D} are *isomorphic* $G_{3D} \cong G_{2D}$ if there is a one-one correspondence between the vertices of G_{3D} and those of G_{2D} such that the number of edges joining any two vertices of G_{3D} is equal to the number of edges joining the corresponding vertices of G_{2D}. The graphs G_{3D} and G_{2D} are isomorphic if there exists a *bijection* $f : V_{3D} \to V_{2D}$ and $g : E_{3D} \to E_{2D}$ such that $\psi_{G_{3D}}(e) = (u, v)$ if and only if $\psi_{G_{2D}} \circ g(e) = (f(u), f(v))$. Such a pair (f, g) of mappings is called an *isomorphism* between G_{3D} and G_{2D}. For all $v_1, v_2 \in V_{3D}$, v_1 and v_2 are adjacent in G_{3D} if and only if $f(v_1)$ and $f(v_2)$ are adjacent in G_{2D}.

Then the isomorfismus between the spatial and planar multigraph can be defined as follows:

Definition 5.1.6. if β is a bijection in terms of definition 5.1.5, then β is called an *isomorphism* between G_{3D} and G_{2D}, thus $\beta : G_{3D} \to G_{2D}$.

The MEIM needs to be further simplified to a so-called *simple* graph, where multiple contact between individual macro elements represented by the multiple edges connecting the same pairs of vertices in MEIM. Such multiple edges are simplified to the one edge only. By deleting from multigraph all loops and, for every pair of adjacent vertices, all but

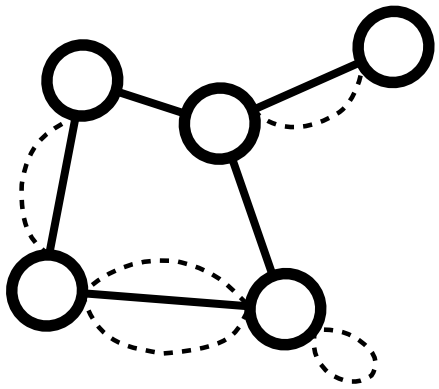


Figure 52: Example of underlying simple graph of multigraph.

one link joining them, then a simple spanning subgraph is obtained. It is called *underlying simple graph*. It is required for the purpose of domain decomposition.

For the further ability to solve numerical models on computer networks, the definition of the *subgraph* (underlying simple graph) is important with respect to the domain decomposition in terms of MEIM. The subgraph definition is as follows:

Definition 5.1.7. Let G and G' be a graphs. Graph G' is a *subgraph* of G (written $G' \subseteq G$) if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$.

In the context of MEIM analysis, the definition of a subgraph can be interpreted so that all the subgraphs from one multigraph arise based on the finding of such graphs that do not share any node and edge. Thus, all the subgraphs must satisfy the so-called connectedness. Thus, the subgraphs must satisfy the so-called *connectedness*, which says that a graph G is *connected* if for any two vertices $u, v \in V(G)$, G contains a path from u to v.

Subsequent definition leads to the importatnt multigraph property, namely to the definition of the *degree* of a vertex. This property plays an important role primarily because of the reason of finding the separate macro models that are not in contact interaction with the others. Respective definition is as follows:

Definition 5.1.8. The *degree* $\deg_G v$ of a vertex $v \in V$ is the number of edges of G incident with v, each loop counting as two edges, $\deg_G v, \deg_G v = |\{e; e \in E \wedge v \in e\}|$.

Theorem 5.1.9. $\sum_{v \in V} \deg_G v = 2\epsilon$.

Proof. Consider the incidence matrix $\mathbf{M}$. The sum of the entries in the row corresponding to vertex v is precisely $\deg_G v$, and therefore $\sum_{v \in V} \deg_G v$ is just the sum of all entries in $\mathbf{M}$. But this sum is also 2ϵ, since each of the ϵ column sums of $\mathbf{M}$ is 2. □

As mentioned before, the DFS algorithm is applied to find connected subgraphs. The DFS algorithm can be considered in both the recursive and non-recursive form, respectively. The non-recursive form of the algorithm is safer with respect to overflow of application

stack. Thus, the use of a data structure to simulate a an application stack is expected. Such a data structure uses important functions, namely the function `PUSH` for adding new element and the `POP` function for obtaining current element, respectively.

Thus, the required algorithm for the purpose of domain decomposition is represented primarily by the DFT algorithm for finding connected subgraphs. The non-recursive form of the DFS algorithm represents the pseudocode 3.

Algorithm 3: Pseudocode of a non-recursive form of the DFS algorithm.

Input: $V(G)$ *vertices*; ADJ *adjacency map*; π *map of connected simple subgraphs* G_i; $i \in \mathbb{N}\backslash\{0\}$ *subgraph id generator*

```
1  Subroutine dfs_non_recursive(V(G), ADJ, π)
2      vstack ← ∅;
3      for v ← ∀v ∈ V(G), i ← i + 1 do
4          PUSH(vstack, v);
5          while vstack ≠ ∅ do
6              v_t ← POP(vstack);
7              if (try_color(v_t, π, i) ∩{False}) ≠ ∅ then
8                  continue;
9              end
10             for v_adj ← ∀v_adj ∈ ADJ[v_t] ⊆ V(G) do
11                 PUSH(vstack, v_adj);
12             end
13         end
14     end
```

Algorithm 4: Pseudocode of function for coloring explored graph nodes.

Input : $v \in V(G)$ *vertex*; π *map of connected simple subgraphs* G_i; CLR *map of explored colored vertices*; $i \in \mathbb{N}\backslash\{0\}$ *subgraph id generator*

Output: $\{True\}$ or $\{False\}$ *Boolean value indicating whether vertex has been explored*

```
1  Function try_color(v, π, CLR, i)
2      if ({v} ∩ CLR) = ∅ then
3          return {False};
4      end
5      if ({i} ∩ π) ≠ ∅ then
6          π[i] ← ∅;
7      end
8      π ← π[i] ∪ {v};
9      CLR ∪ {v};
10     return {True};
```

5.2 The Data Distribution Algorithm

The algorithm for the data distribution within a computer network starts primarily from the basic DFS analysis of the MEIM. Based on the DFS algorithm, all connected subgraphs are obtained, even if they represent isolated nodes only. As an input to the analysis is therefore a map of connected subgraphs π. It contains a set of nodes belonging to each subgraph. For the best understanding of the algorithm, it is appropriate to simulate it on a simple model example. Let us therefore consider the MEIM relationship shown in the Fig. 53.

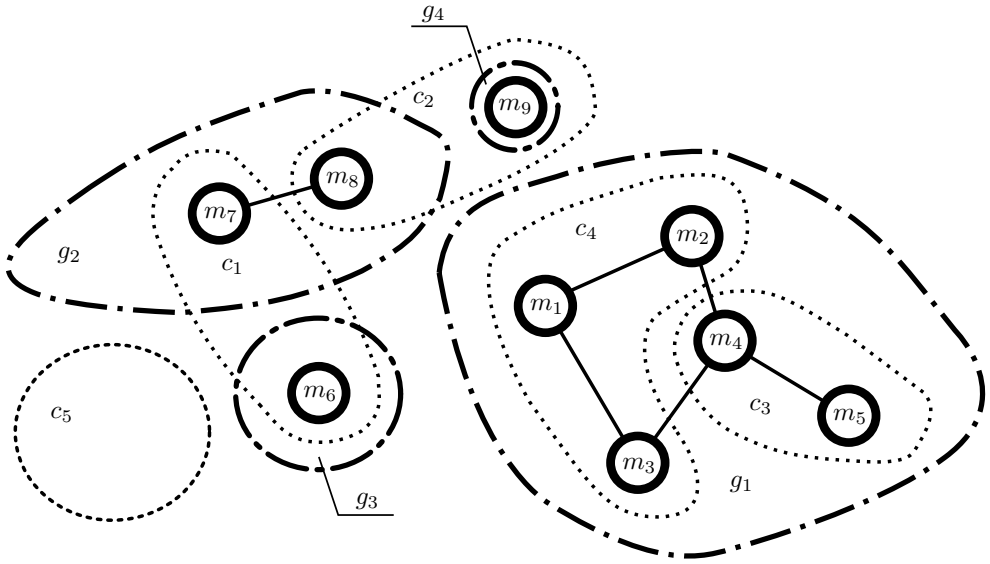


Figure 53: The MEIM model example, where $c_i \in C, m_j \in M$ and $g_k \in G$.

Definition 5.2.1. Let π and S be a two sets. Mapping f from the set π to the set S is relationship $f \subseteq \pi \times S$. Mapping must satisfy the condition, where for each element $p_i \in \pi$ there exists just the one element $s_i \in S$ such that $(p_i, s_i) \in f$, thus $f : \pi \to S$.

Based on the definition 5.2.1 of a given mapping f, it is possible to assemble the contents of the respective sets π and S (see Tab. 51) shown as an arranged pair relationship ((graph,vertex) and (vertex,graph)), respectively. The set π is a source set coming from the DFS analysis.

The MEIM shown in the Fig. 53 represents the *binary relations*. Binary relations are generally defined as follows:

Definition 5.2.2. Given set X and Y, a relation from X to Y is a subset R of $X \times Y$. When $(x, y) \in R$, we say that x is related to y(by R) and write xRy. Similarly, $(x, y) \notin R$ is denoted by $x \not R y$.

Table 51: Result of mapping $f : \pi \to S$ for model example.

Set	Content
π	(g_1, m_1), (g_1, m_2), (g_1, m_3), (g_1, m_4), (g_1, m_5), (g_1, m_9), (g_2, m_7), (g_2, m_8), (g_3, m_6), (g_4, m_9)
S	(m_1, g_1), (m_1, g_2), (m_1, g_3), (m_1, g_4), (m_1, g_5), (m_1, g_9), (m_2, g_7), (m_2, g_8), (m_3, g_6), (m_9, g_4)

From the definition 5.2.2, the relations of the sets can be generally defined. Such relations then closely refer to the MEIM. Thus C, M and G be a sets representing computers, model macro entity and simple connected graphs, respectively. Let R be a relation from C to M and let S be a relation from M to G. That is R is a subset of $C \times M$ and S is a subset of $M \times G$. Then R and S give rise to relation from C to G denoted by $R \circ S$ and defined by $c\,(R \circ S)\,g$ if for some $m \in M$ we have cRm and mSg. Thus $R \circ S = \{(c, g)\,|\exists m \in M$ for which $(c, m) \in R$ and $(m, g) \in S\}$. The relation $R \circ S$ is *composition* of R and S (denoted RS).

In connection with the example shown in the Fig. 53, the content of the set C is $c_i \in C, i \in \{1, \ldots, 5\} \in \mathbb{N}$, M is $m_j \in M, j \in \{1, \ldots, 8\} \in \mathbb{N}$ and G is $g_k \in G, k \in \{1, \ldots, 3\} \in \mathbb{N}$, respectively. The given relations of sets C, M and G are then appropriately represented in the Fig. 54. It is need to note, that there do not exist the *binary relations* $C \times C$, $M \times M$ and $G \times G$.

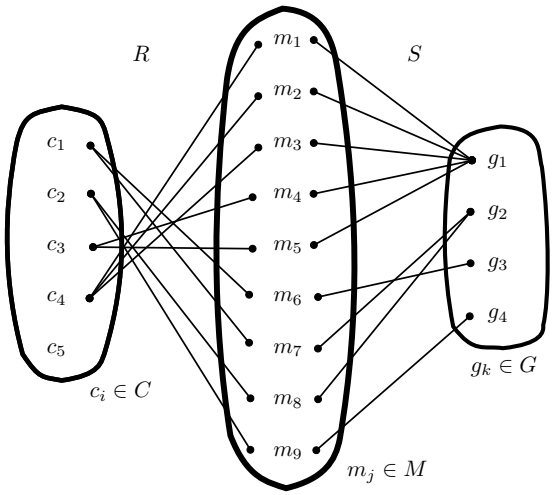


Figure 54: Relations of sets C, M and G.

Based on relations shown in the Fig. 54, the following general theorem could be formulated.

Theorem 5.2.3. Let A, B, C and D be sets. Suppose R is a relation from A to B, S is a relation from B to C, and T is a relation from C to D. Then $(R \circ S) \circ T = R \circ (S \circ T)$.

For the purpose of the data distribution algorithm, it is appropriate to represent the respective relations using the matrices. In the case of the example shown in the Fig. 53 and Fig. 54, the given matrixes have the following form:

$$\mathsf{M}_R = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathsf{M}_S^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

where relations R and S are represented by a zero-one matrixes M_R and M_S, respectively. Each of the entries within the respective matrixes is either 0 or 1. The rows of matrixes M_R and M_S are labeled by the elements of C and M, respectively. The columns of matrixes M_R and M_S are labeled by the elements of M and G, respectively.

From those represented relations, the information required for the given data distribution algorithm could be obtained. It involves finding a sequence of numbers reprezenting the occupancy of the given workstations (computers) by the number of amount of model macro entities within each connected simple graph. Such a sequence then also represents a new relation. As such the new relation T is obtained by the multiplication of matrixes M_R and M_S, $\mathsf{M}_T = \mathsf{M}_{R \circ S} = \mathsf{M}_R \odot \mathsf{M}_R$. Thus, with respect to the example which is solved here, the matrix M_T has the following form:

$$\mathsf{M}_T = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Matrix M_T represents relation $T = R \circ S$. It is the relation from set C to set G. Now, the matrix contains valuable information about the number of a model's macro entities that the appropriate relational pair $(c, g) \in T$ contains. Such a relation T can be drawn as a forest, which is a set of $k > 0$ disjoint trees, where the appropriate root of each tree represents separate interaction graph $g_k \in G$. The forest representing relation T of the solved example is shown in the Fig 55, where the dotted line represents a single workstation $c_i \in C$ containing multiple disjoint simple connected interaction graphs $g_k \in G$.

The dashed line in the Fig. 55 represents the direction for data transfer in the first phase. Due to the workloads of individual workstations within the computer network, it is necessary to sort them according to the amount of models's macro entities they currently contain. Thus the respective criterion is the maximum value of a tree edge. Individual trees can also be understood as a directed graph with the rated edges where the maximum

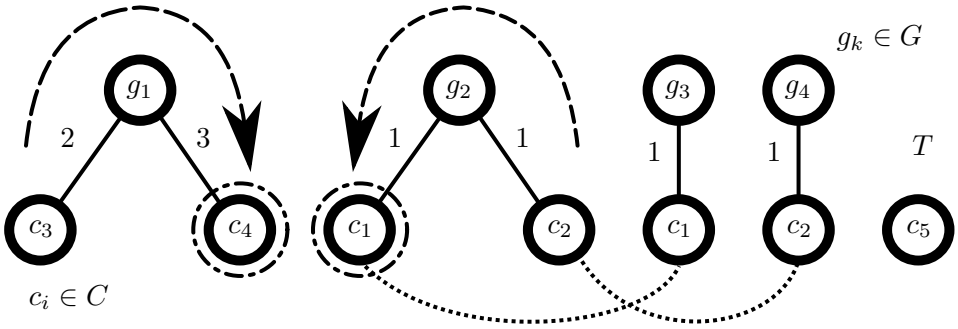


Figure 55: Forest (the set of disjoint trees) from solved example.

value from all graph edges represents the direction from g to c and the rest of the other edges in a graph the direction from c to g. The second phase of the algorithm represents a uniform distribution, as far as possible, of the remaining graph data across individual workstations. The data distribution within the example being solved then represents the Tab. 52. Here the individual graphs $g_k \in G$ represent the relational sets composed from the elements m_j of the set M.

Table 52: Phases of algorithm for a model's macro entities fluctuation in a scope of the single data transfer within the computer network (model example).

Algorithm phase	**Workstation**	**Workstations content**
0 (initial state)	c_1	$\{m_7\} \in g_2, \{m_6\} \in g_3$
	c_2	$\{m_8\} \in g_2, \{m_9\} \in g_4$
	c_3	$\{m_4, m_5\} \in g_1$
	c_4	$\{m_1, m_2, m_3\} \in g_1$
	c_5	$\emptyset$
1	c_1	$\{m_7, m_8\} \in g_2, \{m_6\} \in g_3$
	c_2	$\{m_9\} \in g_4$
	c_3	$\emptyset$
	c_4	$\{m_1, m_2, m_3, m_4, m_5\} \in g_1$
	c_5	$\emptyset$
2	c_1	$\{m_7, m_8\} \in g_2$
	c_2	$\{m_9\} \in g_4$
	c_3	$\{m_6\} \in g_3$
	c_4	$\{m_1, m_2, m_3, m_4, m_5\} \in g_1$
	c_5	$\emptyset$

5.3 Summary of Chapter

In this chapter, important algorithms for domain decomposition of a numerical model for the purpose of numerical simulation in terms of a hybrid-parallel computer solver running in the scope of computer network were introduced. For the mentioned subject, algorithms from graph theory had to be applied. These concern both the search problem in an Euclidean space and also to a search in general graphs containing discrete values. In the end, the heuristics for specific data exchange within a computer network were introduced. The data distribution algorithm was presented on a specific example simulating the specific state of the MEIM. The newly invented term MEIM will be further used in subsequent chapters dedicated to the implementation of already presented approaches.

Chapter 6

Massive Parallel Computing

Advent of multi-core and multiprocessor systems into personal computers in the aftermath of the millennium has changed many approaches in software engineering.

For decades, the vast majority of software applications contained algorithms used for sequential execution of instructions only. Multi-process and asynchronous approaches in programming were generally considered primarily by system engineers for designing operating systems or special applications focused on network resource utilization. To a large extent, however, it was related to a pseudo concurrency (or parallelism) of applications, as the instructions were performed sequentially on a single processor core thread. The impression of concurrency is caused by the rapid alternation of running of the different processes in successive time. For such reason, the division of processor time did not provide any advantage in terms of processing more data, but vice versa. Thus, a large number of software applications for scientific-technical computations have been focused on what they solved rather than how they solved it. Insomuch as it relied heavily on the progressive growth of single CPU performance, there was almost no effort to think about the issues of parallelism and appropriate algorithmization. This was largely due to the purchase price of such computer technology, the limited amount and complexity of software technologies focused on handling such computer systems, and the lack of qualified professionals on the labor market or in academia.

Nowadays, multi-core and multiprocessor computing systems or access to computer networks are an absolute must for everyday life. Therefore, it is an attempt to maximize the usage of such computer performance through applications that are capable of doing so. The reasons are primarily of economic nature, this concerns an increase of a labor productivity and progress in advancement of a developing new technologies for commercial usage.

In conditions of computational mechanics, this concerns mainly analytical software systems using the FEM, finite difference method, finite volume method or their combination to a numerical solution of a set of integral-differential equations often with a nonlinear character. Since many of these software systems have been developed over the decades since the 1960's, thus they contain a large amount of program code with a considerable

amount of tuned functionality. The implementation of new approaches related to the parallelization of respective computations requires considerable expert analysis of existing and new algorithms. The same applies to the analysis of suitable hardware and software technologies. Here, it is also necessary to realize the fact, software and hardware technologies are closely interlinked, so the algorithms used must respect all of the technological constraints in order to optimize their effective usage. The parallel approach has not always been an effective way in effort to improve the performance of affected operations, so the process of searching for opportunities for parallelization is equally important.

At this time, the technologies of multi-core CPU processors, programmable graphical multi-core GPU processors, as well as their combinations, are commonly available on the market. Also due to the affordability of solid computer assemblies, the next level of parallelism is an interconnection of such parallel machines (CPU+GPU) in a generally heterogeneous computer cluster through the computer network. All of the mentioned opportunities and limitations then create huge demands on the complex expert maturity of the responsible person. The situation gets easier if the expertise can be distributed through a team of experts, but it is often not the case, mainly due to the lack of experts on the labor market (situation in the Czech Republic at the time of composing the thesis).

The description of the FEXP solver software architecture and the technologies used are included in the following chapters.

6.1 Theoretical Performance Analysis

In the context of data parallelization, which here is the main focus, it is necessary to mention *Amdahl's law* as a theoretical estimation of application performance. In Amdahl's law, computational workload W is fixed while the number of processors that can work on W can be increased. The execution time of the given work by N processors is then

$$\tau_N = \frac{W_1}{v_1} + \frac{W_N}{v_N}, \quad \$ = \frac{\tau_1}{\tau_N},$$

then it leads to the following final form

$$\$_A = \frac{W}{\xi W + (1-\xi)\, W N^{-1}} \Rightarrow N\big(1 + \xi\,(N-1)\,\big)^{-1}, \tag{6.1}$$

where $\$_A \in \mathbb{R}$ denotes speedup by Amdahl, τ_N denotes execution time given by work of N processors, v_i is the rate of execution of i-th processor $i \in \mathbb{N}^{[1,N]}$, ξW is the portion of work belonging to the sequential running part of a code $W_i = 0$ for $\forall i \in \mathbb{N}^{[1,N]} \backslash \{1, N\}$ and $\xi \in \mathbb{R}^{[0,1]}$ is the fraction ratio, respectively.

Another law dealing with the estimation of speed up is *Gustafson's law*, which says that an increase in problem size for large machines can retain scalability with respect to the number of processors. The execution workload of the whole task before the improvement of the resources of the system is denoted W, and execution workload after the improvement

of the resources is denoted $W(N)$, thus the theoretical Gustafson's speedup $\$_G$ has the following form

$$\$_G = \frac{W(N)}{W} = \frac{\xi W + (1-\xi)NW}{\xi W + (1-\xi)W} \Rightarrow \xi + (1-\xi)N. \tag{6.2}$$

Respective law says that the true parallel power of a large multiprocessor system is only achievable when a large parallel problem is applied.

As the last theoretical model of performance evaluation, let us mention *Su-Ni's law*. This is referred to as a memory bound model. It turns out that when the speedup is computed by the problem size limited by the available memory in n-processor system, it leads to a generalization of Amdahl's and Gustafson's law.

6.2 Distributed and Cloud Computing

Here it is needed to start with the simplest domain unit called thread, used for concurrency or data parallelism regardless of whether it is a thread running on a CPU, GPU, or some its hybrid mutation in terms of the system virtualization.

With the parallelization on OS process level by threads is closely related the process-level parallelization. In such a case, it is a much more complicated approach, primarily due to a separate memory space compared to the shared memory space between the threads in one process. Process synchronization and general data exchange between processes is thus an extensively more challenging task.

The standard relationship between the thread and OS process is many-to-one. However, other types of arrangements exist so that e.g. one-to-many and many-to-many are also valid and currently being investigated. It concerns the clustering for massive parallelism. Such an approach requires a special type of OS so-called *the cloud distributed operating system*. The TRIX research operating system at MIT's Laboratory for Computer Science is one such type of OS. Single user activity represented by a thread can be performed on multiple domains, where the system scheduler can migrate threads between CPUs in order to keep all processors busy. This system represents many-to-many relationship.

The cloud solution represents the highest degree of virtualization when it groups multiple computer clusters under one solid model. It is in relation to computer cluster architecture, where a collection of interconnected stand-alone computers can work together collectively and cooperatively as a single integrated computing resource pool as a hybrid system for massive paralellization. In general, it is a dynamically changing system. These node machines are interconnected through hierarchical construction using a SAN (abbreviation of the Storage Area Network, e.g. Myrinet), LAN (e.g., Ethernet), or WAN (abbreviation of the Wide Area Network). Contained cluster is connected to the Internet via a virtual private network (VPN) gateway, where the gateway IP address locates the cluster. All resources of a server node are managed by their own OS. Thus, most clusters have multiple system images as a result of having many autonomous nodes under different OS control. Cluster designers desire a cluster OS or some middleware to support SSI (shortcut of the

Single System Image) at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes. An SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource. SSI makes the cluster appear like a single machine to the user. A cluster with multiple system images is nothing but a collection of independent computers. This relates to cloud computing defined by IBM as follows: *"A cloud is a pool of virtualized computer resources. A cloud can host a variety of different workloads, including batch-style backend jobs and interactive and user-facing applications."*

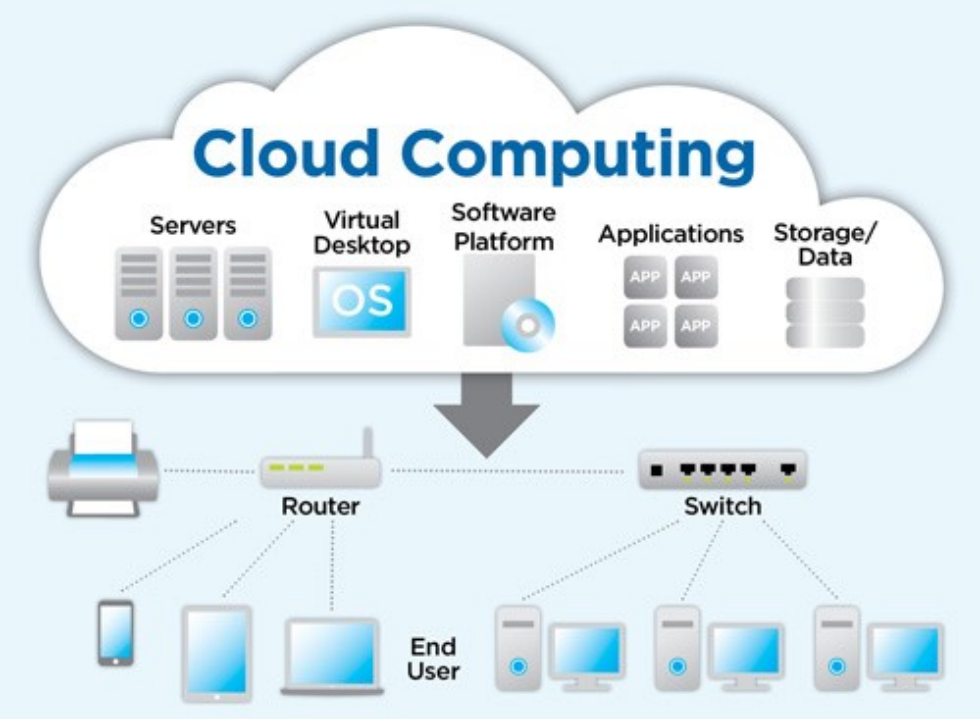


Figure 61: Cloud computing.

Overall, it leads to the idea of using computing resources in a way other than usual in the past, so that working with large data sets in future will mean sending the computations to the data in large data centers provisioning of software, hardware and data as a service, rather than copying the data to the workstations. Cloud and underlying computer clusters are related to a number of technologies, notably the already mentioned systems for the virtualization of computing resources and the SOA (abbreviation of the Service Oriented Architecture) paradigm for accessing data by users (often using the SOAP, abbreviation of the Simple Object Access Protocol).

The later presented solver model is focused on the utilization of heterogeneous clusters interconnected by high-speed network links over selected resource sites. It is also so-called *Computational Grids* (CG). It handles the grid nodes represented by the single workstations interkonected within the LAN. The communication between grid nodes is then secured by a low-level TCP/IP connection protocol (abbreviation of the Transmission

Control Protocol/Internet Protocol). The individual nodes of CG can then use multicore processor services for local data parallelization within the explicit integration of equations of motion. The solution also necessarily includes a solution for network resource workload balancing that can be dynamically changed during the computations. It concerns, in particular, the number of workstations connected to the CG. Such an applied method solves the virtualization of the CG without the use of the common way of virtualization through the distributed OS or without the use of other high-level middleware.

6.2.1 Utilization of CPU Cores

Considered here is parallelization represented by running threads distributed to the physical cores of the respective processor. The considered technology refers to the architecture of superscalar or multi-core processors, where program instructions are lunched by two or more execution pipelines, making it possible for two instructions to be in the execution stage at the same time (for more see [54]). Parallelization using SIMD (shortcut of the Single Instruction Multiple Data) technology, represented primarily by GPGPU, can be additionally implemented for specific parts of computations.

For the purposes of data parallelismus on a local machine the approach to the utilization of CPU threads based on the number of physical CPU cores was chosen. CPUs with Hyper-threading (Intel technology) support cause an increase in the number of available hardware threads. It means virtual division of each CPU core to two virtual cores. This does not lead to an increase of performance in parallel computations.

An important note then relates to the way threads are made. Some libraries are capable of creating threads without OS awareness of their existence. This involves division into the *user-level* and *kernel-level* types of CPU threads.

In the case of user-level threads, creating a thread switching between threads and synchronization between threads can be done without OS kernel intervention. Thread switching is not much more expensive than a procedure call. This applies to the OS that does not support threads. In a pure user-level thread strategy, a multithreaded application cannot take advantage of multiprocessing.

On the other hand, strategy based on the kernel-level thread facility, all of the work of thread management is done by kernel. It means no thread management code in the application level of OS. Windows OS is an example of this approach. The OS kernel maintains context information for the process as a whole and for individual threads within the process. This approach has one important benefit, namely the fact that the OS kernel can simultaneously schedule multiple threads from the same process on multiple processors (CPU cores). Thus, an application can benefit from true parallelism versus user-level thread strategy. Such an approach is used in the thesis for data parallelization on an application level. A disadvantage over the user-level thread strategy is the transfer of control from one thread to another within the same process, which requires an expensive mode switch to the kernel. A higher number of threads used than accessible physical cores would cause performance decreasement.

All modern OSs create an effective abstract layer between user application and underlying powerful harware. This applies primarily to UNIX like OSs and Windows OS (current version 10). In the case of UNIX like OS, this concerns the functionality provided by the POSIX programming interface. For the Windows OS, the situation is somewhat more specific due to a non-open source character compared to Linux, inasmuch as Windows OS is specifically developed by one vendor.

Windows OS for a native accessing to its functionality provides Win32 API represented by the dynamically linked *ntdll.dll* library (see Fig. 62). Since the Win32 API is also referred to as a 64-bit platform, the number 32 in Win32 API is somewhat misleading. Such a label is used for historical reasons, primarily due to the old Windows 95 OS, which meant a breakthrough event in the development of operating systems. For such a reason, all references to Win32 API in the text of thesis will include functionality for both 32 and 64-bit platforms.

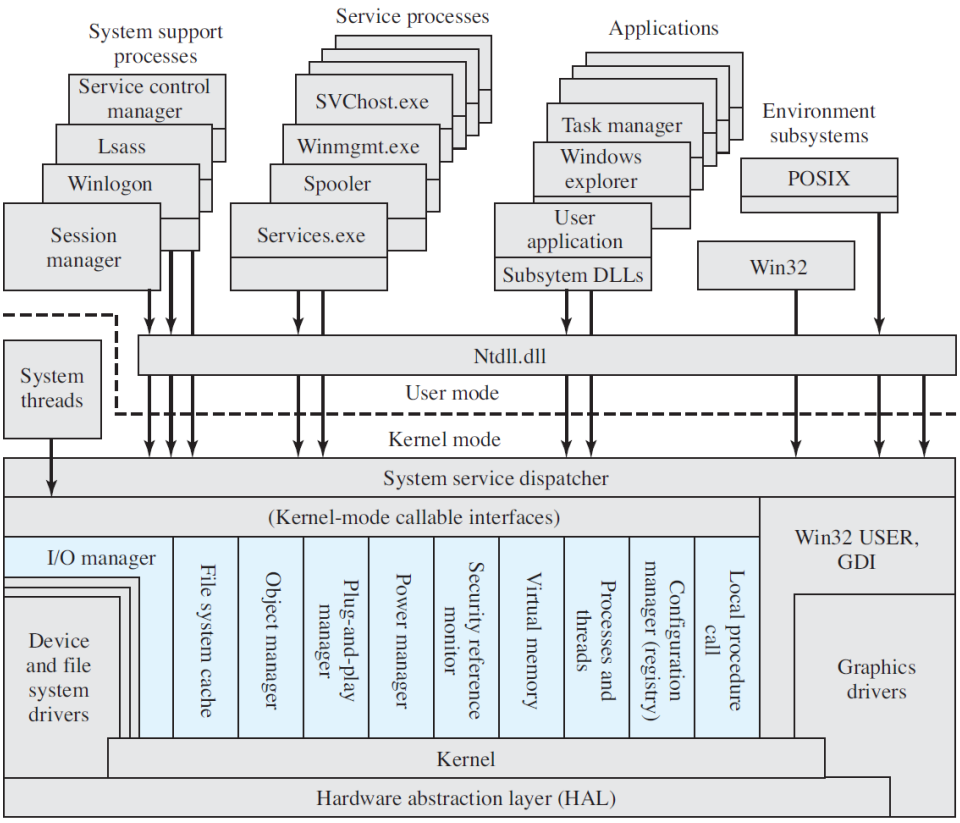


Figure 62: Windows OS (Vista) software architecture.

In a view of the recent extension of a native C++ (since version 11, for ISO/IEC 14882:2011 see https://www.iso.org/standard/50372.html) programming language to the ability to run asynchronous and parallel operations, it is no longer necessary to target multicore computers using OS facilities (Win32 API on Windows or pthreads on UNIX like systems) or special third-party libraries like OpenMP and MPI.

The code can be transferred between the different platforms without any probems thanks to the portability a native C++ programming language. The current C++ ISO/IEC 14882:2017 standard (so called C++ 17) can be purchased and subsequuently downloaded from the web of *International Organization for Standardization* (https://www.iso.org/standard/68564.html).

A large part of the new libraries was defined in the document *C++ Standards Committee's Library Technical Report* ISO/IEC TR 19768, C++ Library Extensions (TR1, published in 2005) as shown in the Fig. 63.

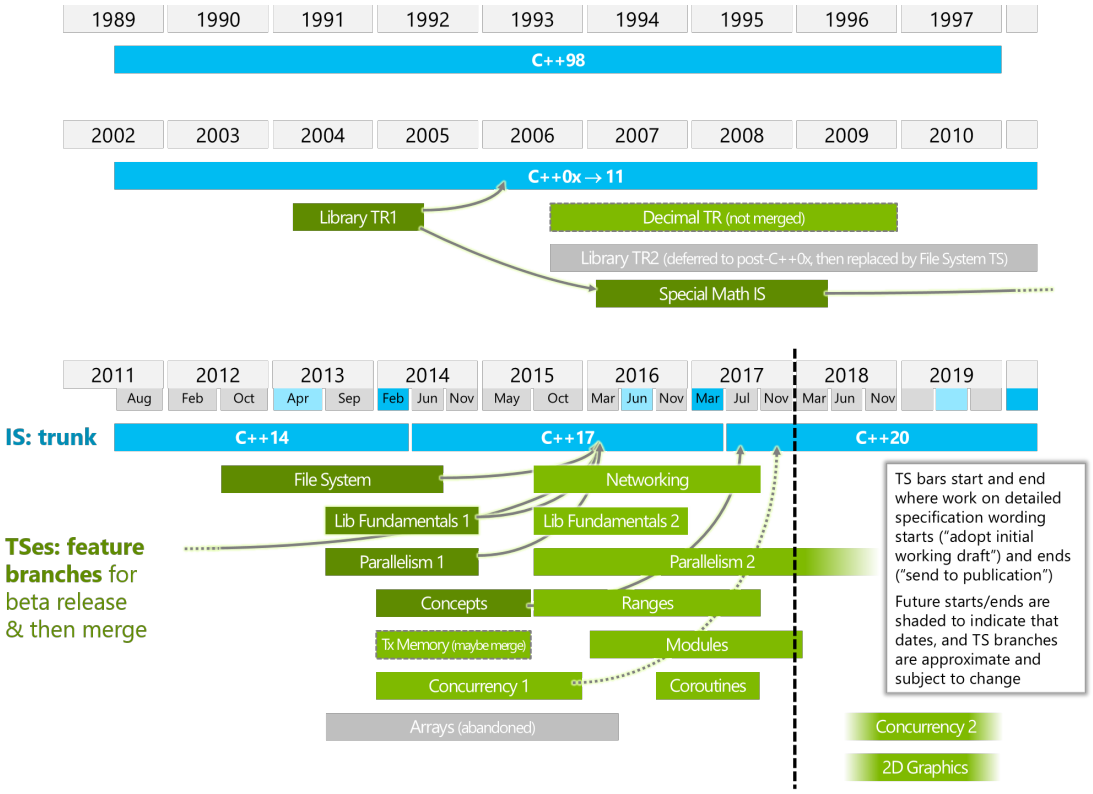


Figure 63: C++ revision timeline.

The previouly mentioned functionality has already been supported by the *Boost* library (see http://www.boost.org/) as a source for the eventual standardization for extensions of C++ programming language. Ten Boost libraries are included in the C++ Standards Committee's Library Technical Report (TR1) and in the new C++11 standard including support of thread and related libraries for thread synchronization. Many new features of C++ 11, including of some features from C++ versions 14 and 17, are used in the FEXP source codes.

6.2.2 Network Based Parellel Computing

In this section it is necessary to restrict the view to the smaller LAN networks with regard to the deployed networking solution in terms of FEXP solver. The main reason for the use of such a solution is the frequent situation in engineering design studios especially of a small and middle extend companies. Such a small LAN network is simply described in the Fig. 64 as a small amount of workstations connected to the server.

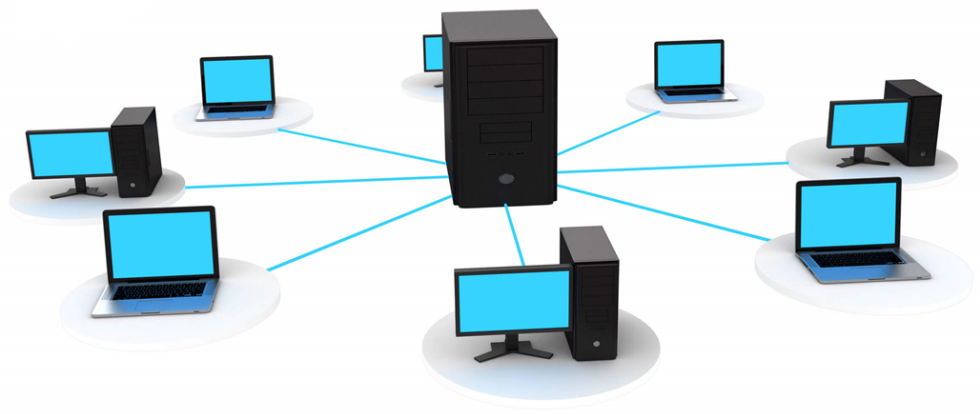


Figure 64: Local Area Network (LAN).

With regard to the use of the proposed network solution and its future expansion, not only for the purposes of explicit dynamics, it is also necessary to consider groups of LANs. Achieving such a goal can then be best done through appropriate middleware like in the case of the Cloud solution, that would take care of the virtualization.

One of the interesting opportunities is approach of the open-source project SoftEther VPN (see https://www.softether.org/, University of Tsukuba), which can provide virtualization of Ethernet devices. This software virtualizes Ethernet devices in order to realize a flexible *virtual private network* (VPN) for both remote-access VPN and site-to-site VPN through the Virtual Network Adapter program as a software-emulated traditional Ethernet network adapter. Communication between individual workstations from different LANs is carried out through the VPN communication tunnel, which means mutual communication through the IP routing enabled VPN. When a computer from one LAN attempts to communicate with a host on another LAN it will automatically do so through the VPN. A detailed description of the implementation details and provided possible network configurations for building such groups using the SoftEther VPN software can be found on web https://www.softether.org/4-docs/1-manual. Some examples of building VPN networks considered in an open-source free cross-platform multi-protocol VPN program SoftEther VPN are shown in the Fig. 65.

For the network data interchange between individual worstations in the scope of respective LAN, the SOA-based paradigm high-level technologies such as Microsoft .NET/WCF (abbreviation of the Windows Communication Foundation) or JEE (abbreviation of the Java Enterprise Edition) and similar could be used. For the cross-platform applicability,

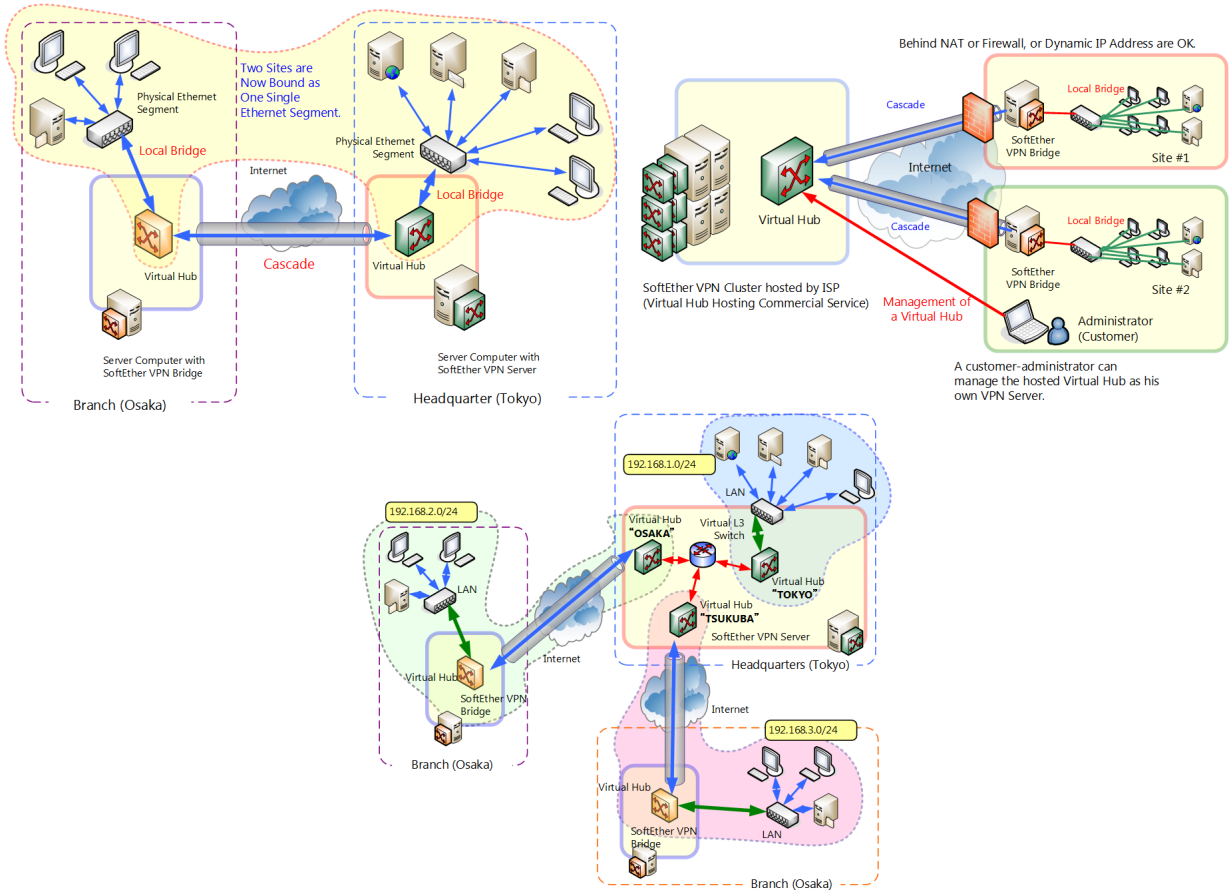


Figure 65: Examples of possible VPN configurations (SoftEther VPN).

such technologies use HTTP (abbreviation of the Hypertext Transfer Protocol), XML (abbreviation of the eXtensible Markup Language), SOAP, and UDDI (abbreviation of the Universal Description, Discovery, and Integration) high-level communication protocols.

All of the mentioned technologies are then built on the shared core, which is based on the lower-level protocols of the transport layer, network layer and the link layer respectively. For the purpose of the network solution within the FEXP solver, the TCP protocol of transport layer is the most important. The rest of underlying low-level protocols is implicitly included, such as IP protocol of network layer (IPv4 or IPv6). The process of data transport across individual network layers is shown in the Fig. 66.

Network communication within the FEXP solver is performed over the opened low-level network socket by TCP transport protocol against to delivery unsafe UDP protocol. The Windows Sockets from Win32 API library (is not strictly part of it) are used to create communication. They are almost the same as, and interoperable with, Berkeley Sockets, and de facto industry standard. It extends the Berkeley Sockets API into the Windows environment. It then allows it to exploit higher-level protocols and applications, all of which provide different, and higher-level, models for standard, interoperable, networked

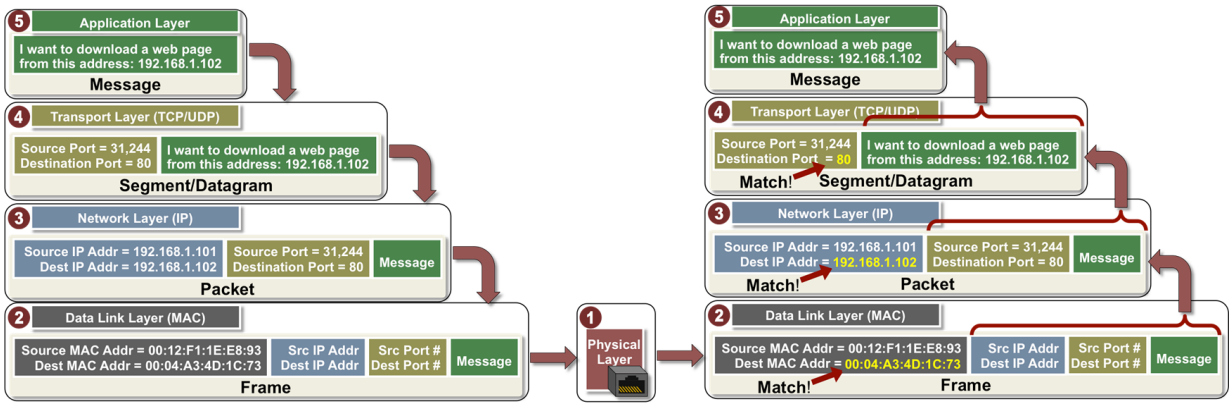


Figure 66: Example of sending and receiving data across TCP/IP network.

interprocess communication.

With respect to future revisions of the C++ programming language, socket implementation within the FEXP solver will be complemented by the portable form of network communication using the standard libraries. Currently, this possibility supports the *Boost.Asio* (see http://www.boost.org/doc/libs/1_66_0/doc/html/boost_asio.html) library, which would now require to be extra added to the FEXP project binaries as a third-party library. It is a cross-platform C++ library for network and low-level I/O programming that provides developers with a consistent asynchronous model. This library is expected to become a part of C++ standard libraries, as in the case of multithreaded programming support, as mentioned earlier.

Specific implementation details of a network solution within the FEXP solver are presented in the following chapters dealing with the architecture of the program.

6.3 Introduction of the Hybrid Parallel Testing Solver FEXP

Since the implementation of the discussed algorithms and technologies in existing open-source projects mentioned earlier would require considerable time for their own analysis, a hybrid-parallel numerical testing solver called FEXP (abbreviation of the Finite [E]lement [E]XPlicit solver) was composed. The solver has a testing purpose, which means testing the effectiveness of both the algorithms and the technologies used.

This thesis is focused on the usage of parallelization for numerical computations in nonlinear dynamics, when an explicit approach to the direct time integration of equations of motion was chosen as a natural source of opportunity to an effective usage in paralelization of a numerical computations. The physics of the considered models and numerical algorithms were presented in the previous chapters.

The FEXP solver for parallelization of computations combines the possibility of using multi-core processors (CPU) with the possibility of parallelization in computer heterogen-

eous cluster interconnected in a LAN. With respect to generality, availability and support of a wide range of software technologies and software modularity, the programming language C++ in current version 14 was chosen as the main programming language for the composition of FEXP solver. As a development environment, the Windows OS was chosen, primarily due to the availability of advanced development tools required to develop such applications. Thus for the project management and code compilation, IDE Microsoft Visual Studio 2015/2017 Community was used. For the purpose of easier user control over the FEXP solver, the windows based application FEXP Solver Manager was designed and programmed using Microsoft .NET/C# programming language requiring .NET Framework 4.6 (C# 6.0) or later versions.

6.3.1 Applied Software Architecture

A well-chosen software architecture is especially important because of the possible later seamless expansion of both new functionality and new technologies. The current trend is to compose software using installable libraries (plugins) that modify or complement functionality of a more generally designed software core, which provides specific interface. This type of customization is provided by enterprise systems, e.g. products of SAP company (see https://www.sap.com/products.html) or PDM (abbreviation of the Product Data Management, see Smap3D http://www.smap3d.com/PDM/en/pdm-cad-concept.html) and ERP (abbreviation of the Enterprise Resource Planning) software solutions from the other vendors.

The good modularity of the program primarily enable good maintance of the respective software project, which is the critical part in terms of smooth further development and subsequent profit of the respective software company and their custommers, who can quickly use stable and flexible software for their business. In this context, a great deal of contained functionality has been designed modularly using a template approach as a modern object oriented programming paradigm. A similar approach is applied in the Microsoft ATL library (abbreviation of the Active Template Library) to simplify programming of COM (abbreviation of the Component Object Model) objects, ActiveX members and Win32 applications.

6.3.2 The Parallel Hybrid Model

For the purpose of paralelization of numerical computations, a combination of a local parallelization on a single workstation with the possibility of interconnecting multiple such workstations within a computer network (LAN) was chosen. In the case of a local workstation within the LAN, it concerns the use of multicore CPUs and the later possible usage of GPGPU technology Nvidia CUDA or OpenCL refering to the use of Graphics Processing Units (GPU) for some parts of the computations.

For the mentioned reasons, the FEXP solver is divided into two main parts from an external viewpoint. To the parts refering to the functionality running on a single worksta-

tion and the part of the networking extension. This model is illustratively shown in the Figure 67.

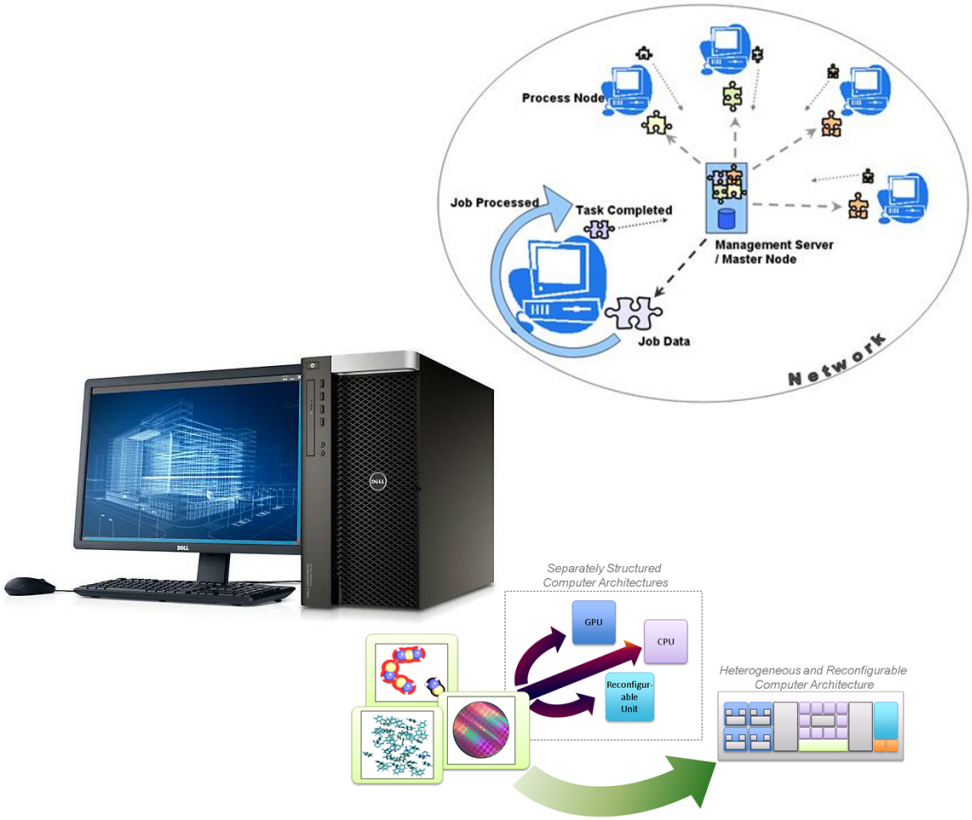


Figure 67: Simplified representation of designed parallel model of the FEXP solver.

From the Fig. 67, it is noticeable that the major part of all computations is done on the single workstation. The server computer then works primarily as a final assembler and distributor of a overall work. From this perspective, the FEXP model is comparable to the massive parallel processing model used in the LS-DYNA under the ANSYS LS-DYNA Parallel license (see https://www.ansys.com/products/platform/ansys-high-performance-computing). However, this approach requires third-party MPI software support be correctly installed (see https://www.ibm.com/us-en/marketplace/spectrum-mpi or https://software.intel.com/en-us/articles/intel-mpi-library-documentation/).

The parallel model of FEXP is designed somewhat more generally in this case, initially with greater emphasis on efficient utilization for explicit impact dynamics. Unlike LS-DYNA, it does not have to use the demanding methods of domain decomposition. The critical part of the LS-DYNA parallel model is based on a domain decomposition (decomposition of meshes) using METIS (Serial Graph Partitioning and Fill-reducing Matrix Ordering, see http://glaros.dtc.umn.edu/gkhome/metis/metis/overview) similarly as a free open-source finite element program SIFEL that uses the open source library HPMESHDECOMP based on METIS (see http://mech.fsv.cvut.cz/~sifel/TOOLS/mes

hdecomp.html). In the case of program SIFEL, the FETI method is then applied on decomposed model.

The parallel FEXP model is based on a physically logical decomposition according to the specific nature of the addressed impact problem. This approach is applicable primarily to the simulation of the impact tasks of a wider group of separate entities interacting with each other through contact forces. As an example in this context, collision of many vehicles on the highway could be mentioned. Throughout the addresed models, the critical part of an effective contact detection must be considered. This of course also applies to the so-called *self-contact* problem. For such a reason, an algorithm for the so-called *nearest neighbor searching* is applied.

Since the parallel hybrid model is based on the interaction of distinct entities, so for the purpose of computation on the computer network, it was necessary to design a heuristic approach for computer network balancing based on an analysis of the *macro entity interaction multigraph* (MEIM). An analysis of the MEIM provides a scope for future optimization, both in terms of the quantity and character of the data being considered, as well as different types of heuristics such as their popular subset of the so-called *metaheuristics*, often used in *soft computing*. The applied algorithm for the MEIM analysis will then be further presented in subsequent chapters as well as the algorithm for self-contact and also for macro body contact detection.

6.4 Description of the FEXP Parts

In this chapter, the individual parts of the FEXP solver are presented in a more detailed form. Since the total number of source code lines is currently somewhere around the number 20,000, only some snippets of code parts are directly listed here. An overal description of the individual parts of the source code can be found in the attachment of the thesis, in the documentation generated by the program Doxygen (see http://www.stack.nl/~dimitri/doxygen/) usually used in software engineering for the creation of program documentation.

For the purpose of program describtion, the specific parts were selected in such a way which enables an unfamiliar developer to easily penetrate to the logic of program assembly. It is primarily due to the needs of further development that could be expected like it is suggested in the final conclusion of the thesis.

The FEXP solver architecture is presented in the Fig. 68 in its simplified form. It schematically describes main thematic blocs contained in the program. As shown the FEXP solver is modularly composed from the separate parts meeting the required functionality explicitly defined by a specific interface for each such part. For better understanding of the FEXP solver software architecture, a rough explanation of an individual parts then follows.

- *Common and Concurency* libraries include both the functionality generally used throughout the FEXP solver, as well as functionality related to the work with threads and asynchronous operations. It also includes resources for safe management with

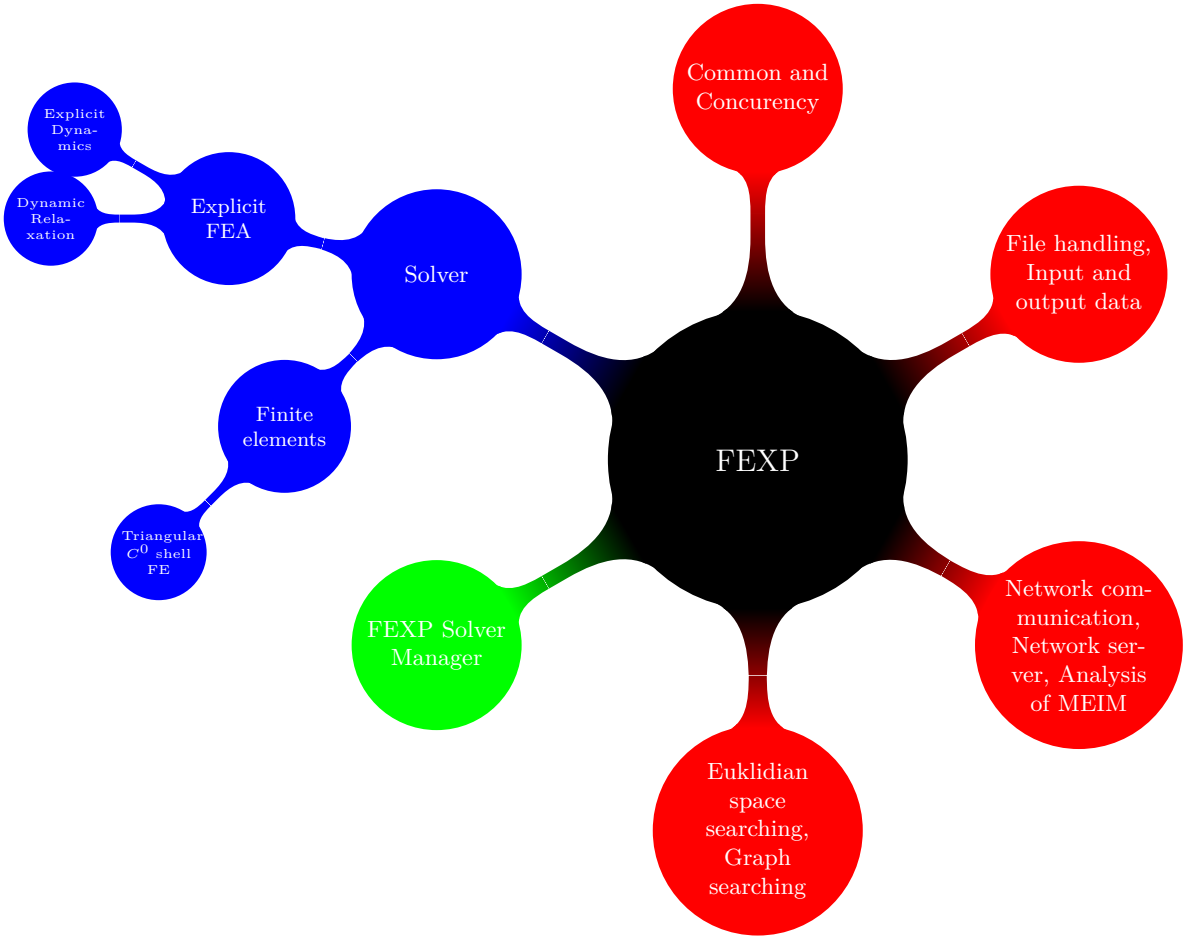


Figure 68: Composition of the FEXP solver.

the dynamically allocated memory, it relates especially to the usage of so-called *smart pointers*.

- *File handling, Input and output data* libraries includes both the functionality focused on handling with input and output files (preprocessing and post processing) and important part related to data tables assembly.

- *Network communication, Network server, Analysis of MEIM* libraries include both inter-process communication functionality for the scope of the computer network and the analysis-related functionality required for the data exchange within the considered computer network. It is based on the analysis of a non-oriented graph representing the interaction of individual bodies (MEIM) during the numerical simulation. Here, it is necessary to also include the implementation of the network server.

- *Euklidian space searching, Graph searching* libraries include both the basic algorithms

required for the general solution of the nonlinear contact problem, as well as algorithms required for the representation of the non-oriented interaction multigraph and the basic algorithms for their analysis.

- *Solver* includes the functionality required for the numerical simulation of dynamic processes. It concerns in particular the library containing functionality for the integration of specific FEs, as well as the functionality that controls the process of explicit numerical integration of equations of motion in terms of the FEA. It includes the parts designed purely for the calculation in a single workstation, as well as the part concerning the computations within a computer cluster.

- *FEXP Solver Manager* is an external graphical UI based application written in the C# programming language. It is not a necessary part of the FEXP solver solution. It is intended mainly for simplifying the work with the FEXP solver and for presentation purposes.

The listing above contains a basic view of the FEXP solver software solution. It is only a simplified look into the entire designed software solution, especially due to its huge overall size. However, despite considerable simplification, it is possible to subsequently identify and understand the individual parts of the source code.

The following text of this chapter is focused primarily on those parts that are important to understand the entire FEXP software solution. The first part deals with the definition of the type and the character of the input data, the next part relates to the numerical simulation itself, and the final part deals with the presentation of the results. The numerical simulation section is then described in a more detailed fashion, all with regard to parallel data processing. A significant part is then devoted primarily to the hybrid-parallel form of the FEXP solver. It deals with numerical computations within the heterogeneous computer cluster. The FEXP Solver Manager section is appropriately included throughout the text as suggests its purpose.

Where appropriate, the source code fragments are included in each section to give better insight into the implementation of the particular portion involved. With respect to the considerable amount of a source code lines. An effort is made to highlight only those parts of the source code representing an important point for the programmer's orientation. The relational UML diagram is then presented at the beginning of given sections to present the object relationships of an appropriate program section. For the purpose of assembly UML diagrams, the program tool Visual Paradigm (https://www.visual-paradigm.com/) was used. The given program is a commonly used tool for object oriented design purposes while software solution designing process.

6.4.1 Preprocessing

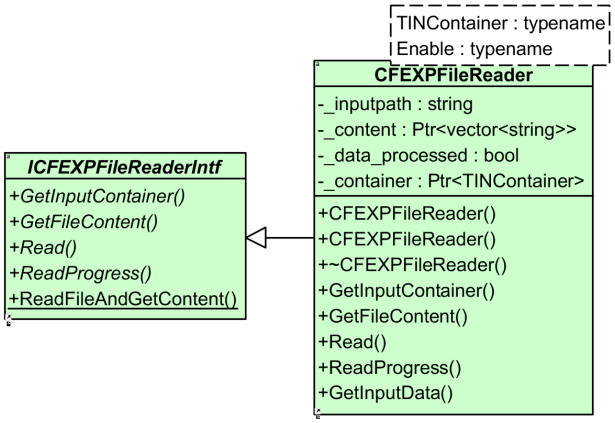


Figure 69: UML diagram of designed file reader.

The processing of the input file text stream is performed by the template **class CFEXPFileReader** through the interface defined by abstract **class ICFEXPFileReaderIntf** whose structure is shown in the Fig. 69. Such functionality occupied the source file `FEXPFileHendler.h`. As a result the appropriate input data container is filled, as shown in the following fragment of source code.

```
/** @brief It reads content of text file.
*/
template<typename TINContainer>
void CFEXPFileReader<TINContainer, typename std::enable_if
<std::is_base_of<ICFEXPInpDataContBase, TINContainer>::value>::type>::Read()
{
  if (_data_processed)
    return;
  // read file and fill content vector
  if(_content->empty())
    ReadFileAndGetContent(_inputpath, *_content.get());
  // create data
  _container->ProcessLines(*_content.get());
  // mark data as already processed
  _data_processed = true;
}
```

Such a functionality is used to handle all types of input data files that are utilized in the FEXP solver. The processing of input files at the startup of the FEXP solver on a separated workstation presents the following fragment of a source code.

```
/** @brief Main function (single workstation).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{
...

  // read solver setting
  auto reader = CFEXPDataManager<CFEXPFileReader<CFEXPInpDataContainer>>
    ::SafeAllocInstance(config_path,
    CFEXPSolverInpDataAssemblyFactory::INP_FILE_BLOCK_CLS_MAP);
  if (!reader->ReadProgress())
```

```
  {
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Error: Problem with configuration file!");
    FEXPCOMMON_CONSOLE_PAUSE(std::get<FEXPCOMMON_CMD_MANAGER_INDEX>(cmd));
    return EXIT_FAILURE;
  }

...

  // solver configuration setting
  auto solver_config_setting = FEXPCOMMON_DYNCAST(ICFEXPModelDataIntf,
    CFEXPSolverConfigSetting, builder->GetModelContainer()->GetModelElement(
      ICFEXPSetting::ESettingType::eSolver, ESystemElementType::eSetting));

  // input data file paths
  std::map<std::string, std::string> input_model_file_map;
  FEXPCOMMON_FOREACH_ITER(solver_config_setting->GetFilePaths())
    input_model_file_map.insert(
      MAP_PAIR(CFEXPBaseConvers::NumberToString(IT->first), IT->second));

  // input data file reader
  std::map<std::string, Ptr<ICFEXPFileReaderIntf>> file_reader_map;
  // read input file and create map of input file containers
  auto read_succ = true;
  FEXPCOMMON_FOREACH_ITER(input_model_file_map)
  {
    auto key = IT->first;
    auto pth = IT->second;
    file_reader_map.insert(MAP_PAIR(key,
      CFEXPDataManager<CFEXPFileReader<CFEXPInpDataContainer>>::SafeAllocInstance(
        pth, CFEXPInpDataModelAssemblyFactory::INP_FILE_BLOCK_CLS_MAP)));
    if (file_reader_map[key]->ReadProgress())
      continue;
    read_succ = false;
    break;
  }

...
}
```

6.4.2 Preprocessing-Structure Model Input Data

For the purpose of model assembly an approach through the structured text files based on [*.csv] file format with the semicolon character as a delimiter was chosen. This type of structured file is supported by spreadsheet programs such as Microsoft Office Excel, OpenOffice Calc or LibreOffice Calc. Thus it enables their easy creation and especially editing. Therefore all input data defining the respective model are formatted by well-arranged tables in thematic blocks. For the mentioned reasons, a new input file type with extension [*.fexin] was designed. A description of the individual input file blocks is contained in the following text.

File Block—*#CALCULATION.dynamic*

This block contains general data for the control of a numerical simulation. These are the data representing the beginning and the end of the dynamic process, the initial time step, an output data storing frequency, type of solver, and the number of threads used. It

occupies the input table with the name $TAB_D1. A formal representation of the table $TAB_D1 with the illustrative values is shown in the Tab. 61.

Table 61: Table $TAB_D1.

start [s]	**stop** [s]	**step** [s]	**print** [–]	**solver** [–]	**threads** [–]
0.0	2.0	1.578609007956055E-4	10	1	0

Table 62: Description of the table $TAB_D1.

Column Name	**Description**	**Value**
start	Start time of simulation.	$\mathbb{F}_d^+$
stop	End time of simulation.	$\mathbb{F}_d^+$, **start** $\leq$ **stop**
step	Initial time step.	$\mathbb{F}_d^+$, **step** $\geq$ **stop** $-$ **start**
print	Result data storing frequency. It denotes the number ot time steps to be taken between individual exports of results.	$\mathbb{S}\backslash\{0\}$
solver	Value indicates the kind of solver to use.	$\{0, 1, 2\} \in \mathbb{S}$, 0—sequential solver, 1—parallel solver, 2—hybrid solver (network+CPU (+GPU) parallel, it requires special treatment!)
threads	Value indicates the numeber of threads to use for parallel computation. Value is ignored for sequential type of solver.	$\mathbb{S}$ (the number 0 has the special meaning for solver type 1 and 2, which means automatic determination of the number of threads)

A description of the individual columns of table $TAB_D1 is contained in the Tab. 62. An example of a respective block in an input text file, currently with only one table $TAB_D1 has the formatted form shown in the Fig. 610.

File Block—*#CALCULATION.material*

This block contains the definition of the materials used. Homogenuous isotropic linearly elastic materials can only be currently considered. It occupies the input table with the name $TAB_M1. A formal representation of the table $TAB_M1 with the illustrative values is shown in the Tab. 63.

```
#CALCULATION.dynamic
$TAB_D1;start;stop;step;print;solver;threads
0.0;2.0;1.578609007956055E-4;10;1;0
```

Figure 610: Example of the $TAB_D1 table in block #CALCULATION.dynamic.

Table 63: Table $TAB_M1.

id [–]	**rho** $[kg\ m^{-3}]$	**E** $[kg\ m^{-1}s^{-2}]$	**nu** [–]
1	7800	210E9	0.3
2	2500	35E9	0.3
...			

Table 64: Description of the table $TAB_M1.

Column Name	Description	Value
id	Material identifier number.	$\mathbb{S}\backslash\{0\}$
rho	Material density.	$\mathbb{F}_d^+$
E	Young's elasticity modulus.	$\mathbb{F}_d^+$
nu	Poisson's ratio.	$\mathbb{F}_d^{(0,0.5)}$

Descriptions of the individual columns of table $TAB_M1 are contained in the Tab. 64. An example of a respective block in an input text file, currently with only one table $TAB_M1 has the formatted form shown in the Fig. 611.

```
#CALCULATION.material
$TAB_M1;id;rho;E;nu
1;7800;210E9;0.3
2;2500;35E9;0.3
...
```

Figure 611: Example of the $TAB_M1 table in block #CALCULATION.material.

File Block—*#CALCULATION.constrains*

This block contains the definition of constraint conditions related to velocities and accelerations of finite element nodes which were used. The velocity and acceleration constrains occupies the input tables with the name $TAB_CSTR_V and $TAB_CSTR_A respect-

ively. A formal representation of the tables $TAB_CSTR_V and $TAB_CSTR_A with the illustrative values are shown in the Tab. 65.

Table 65: Table $TAB_CSTR_V/$TAB_CSTR_A.

id [−]	**time** [s]	**val_x**	**st_x** [−]	**val_y**	**st_y** [−]	**val_z**
1/2	0/0	0/0	1/1	0/0	1/1	0/0
...						

st_z [−]	**val_rx**	**st_rx** [−]	**val_ry**	**st_ry** [−]	**val_rz**	**st_rz** [−]
1/1	0/0	1/1	0/0	1/1	0/0	1/1
...						

Table 66: Description of the table $TAB_CSTR_V/$TAB_CSTR_A.

Column Name	**Description**	**Value**
id	Load identifier number.	$\mathbb{S}\backslash\{0\}$
time	Corresponding time.	$\mathbb{F}_d^+$
val_[α]	Value associated with displacements, $\alpha \in \{x, y, z\}$.	$\mathbb{F}_d$
st_[α]	State of constrain associated with displacements in corresponding **time** and axis $\alpha \in \{x, y, z\}$, respectively.	$\{0, 1\} \in \mathbb{S}$ (respective values have a meaning 0—Off and 1—On)
val_r[α]	Value associated with rotations, $\alpha \in \{x, y, z\}$.	$\mathbb{F}_d$
st_r[α]	State of constrain associated with rotations in corresponding **time** and axis $\alpha \in \{x, y, z\}$, respectively.	$\{0, 1\} \in \mathbb{S}$ (respective values have a meaning 0—Off and 1—On)

Descriptions of individual columns of the tables $TAB_CSTR_V and $TAB_CSTR_A, respectively, are contained in the Tab. 66. Corresponding units for velocities are $[ms^{-1}]$ and $[rad\ s^{-1}]$ (rotational velocity), and for acceleration is applied $[ms^{-2}]$ and $[rad\ s^{-2}]$ (rotational acceleration), respectively. An example of respective block in an input text file both with tables $TAB_CSTR_V and $TAB_CSTR_A has the formatted form shown in the Fig. 612.

```
#CALCULATION.constrains
$TAB_CSTR_V;id;time;val_x;st_x;val_y;st_y;val_z;st_z;val_rx;st_rx;val_ry;
1;0;0;1;0;1;0;1;0;1;0;
...
$TAB_CSTR_A;id;time;val_x;st_x;val_y;st_y;val_z;st_z;val_rx;st_rx;val_ry;
2;0;0;1;0;1;0;1;0;1;0;
...
```

```
st_ry;val_rz;st_rz
1;0;1
...
st_ry;val_rz;st_rz
1;0;1
...
```

Figure 612: Example of the tables $TAB_CSTR_V and $TAB_CSTR_A in block #CALCULATION.constrains.

File Block—*#CALCULATION.loads*

This block contains a load definition represented by the nodal force load table and directly by the nodal acceleration load table. The nodal force load and nodal acceleration load occupies the input tables with the names $TAB_LOAD_A and $TAB_LOAD_F, respectively. A formal representation of the tables $TAB_LOAD_A and $TAB_LOAD_F with the illustrative values are shown in the Tab. 67.

Table 67: Table $TAB_LOAD_A/$TAB_LOAD_F.

id [−]	**time** [s]	**val_x**	**st_x** [−]	**val_y**	**st_y** [−]	**val_z**
1/2	0/0	0/0	0/0	0/0	-9.81/0	1/0
...						

st_z [−]	**val_rx**	**st_rx** [−]	**val_ry**	**st_ry** [−]	**val_rz**	**st_rz** [−]
0/0	0/0	0/0	0/0	0/0	0/0	0/0
...						

Descriptions of the individual columns of the tables $TAB_LOAD_A and $TAB_LOAD_F, respectively, are contained in the Tab. 68. Corresponding units for velocities are $[ms^{-1}]$ and $[rad\ s^{-1}]$ (rotational load), and for acceleration is applied $[ms^{-2}]$ and $[rad\ s^{-2}]$ (rotational acceleration), respectively. An example of respective block in an input text file both

Table 68: Description of the table $TAB_LOAD_A/$TAB_LOAD_F.

Column Name	Description	Value
id	Load identifier number.	$\mathbb{S}\backslash\{0\}$
time	Corresponding time.	$\mathbb{F}_d^+$
val_[α]	Value associated with displacements, $\alpha \in \{x, y, z\}$.	$\mathbb{F}_d$
st_[α]	State of load associated with displacements in corresponding **time** and axis $\alpha \in \{x, y, z\}$, respectively.	$\{0, 1\} \in \mathbb{S}$ (respective values have a meaning 0—Off and 1—On)
val_r[α]	Value associated with rotations, $\alpha \in \{x, y, z\}$.	$\mathbb{F}_d$
st_r[α]	State of load associated with rotations in corresponding **time** and axis $\alpha \in \{x, y, z\}$, respectively.	$\{0, 1\} \in \mathbb{S}$ (respective values have a meaning 0—Off and 1—On)

with tables $TAB_LOAD_A and $TAB_LOAD_F has the formatted form shown in the Fig. 613.

```
#CALCULATION.loads
$TAB_LOAD_A;id;time;val_x;st_x;val_y;st_y;val_z;st_z;val_rx;st_rx;val_ry;
1;0;0;0;0;0;-9.81;1;0;0;0;0;0;0
...
$TAB_LOAD_F;id;time;val_x;st_x;val_y;st_y;val_z;st_z;val_rx;st_rx;val_ry;
2;0;0;0;0;0;0;0;0;0;0;0;
...
```

```
st_ry;val_rz;st_rz
0;0;0
...
st_ry;val_rz;st_rz
0;0;0
...
```

Figure 613: Example of the tables $TAB_LOAD_A and $TAB_LOAD_F in block #CALCULATION.loads.

File Block—*#GEOMETRY.nodes*

This block contains all data related to the finite element nodes. It occupies the input table with the name $TAB_D1. A formal representation of the table $TAB_ND1 with the illustrative values is shown in the Tab. 69.

Table 69: Table $TAB_ND1.

id [–]	**x** [m]	**y** [m]	**z** [m]	**constrain_v_id** [–]	**constrain_a_id** [–]	**load_id** [–]
1	0	0	0	0	0	1
2	2	0	0	1	0	0
...						

Table 610: Description of the table $TAB_ND1.

Column Name	**Description**	**Value**
id	Node identifier number.	$\mathbb{S}\backslash\{0\}$
x	Node coordinate on X-axis.	$\mathbb{F}_d$
y	Node coordinate on Y-axis.	$\mathbb{F}_d$
z	Node coordinate on Z-axis.	$\mathbb{F}_d$
constrain_v_id	Velocity constrain identifier.	$\mathbb{S}$
constrain_a_id	Velocity constrain identifier.	$\mathbb{S}$
load_id	Load identifier.	$\mathbb{S}$

Descriptions of the individual columns of table $TAB_ND1 are contained in the Tab. 610. An example of respective block in an input text file with table $TAB_ND1 has the formatted form shown in the Fig. 614.

```
#GEOMETRY.nodes
$TAB_ND1;id;x;y;z;constrain_v_id;constrain_a_id;load_id
1;0;0;0;0;0;1
2;2;0;0;1;0;0
...
```

Figure 614: Example of the table $TAB_ND1 in block #GEOMETRY.nodes.

File Block—*#GEOMETRY.elements*

This block contains all data related to the finite elements. It occupies the input table with the name $TAB_EL1. Triangular 3-noded shell finite element described in chapter

3.4.2 is currently applied only. A formal representation of the table $TAB_EL1 with the illustrative values is shown in the Tab. 611.

Table 611: Table $TAB_EL1.

id [–]	**type** [–]	**nid1** [–]	**nid2** [–]	**nid3** [–]	**material** [–]	**t** [m]
1	1	1	2	8	1	0.001
2	1	8	7	1	1	0.001
...						

Table 612: Description of the table $TAB_EL1.

Column Name	Description	Value
id	Element identifier number.	$\mathbb{S}\backslash\{0\}$
type	Element type.	Currently $\{1\} \in \mathbb{S}$ only
nid1	First node identifier number.	$\mathbb{S}\backslash\{0\}$
nid2	Second node identifier number.	$\mathbb{S}\backslash\{0\}$
nid3	Third node identifier number.	$\mathbb{S}\backslash\{0\}$
material	Material identifier number.	$\mathbb{S}$ (0—for rigid body)
t	Element thickness.	$\mathbb{F}_d^+$

Descriptions of the individual columns of table $TAB_EL1 are contained in the Tab. 612. An example of a respective block in an input text file with table $TAB_EL1 has the formatted form shown in the Fig. 615.

```
#GEOMETRY.elements
$TAB_EL1;id;type;nid1;nid2;nid3;material;t
1;1;1;2;8;1;0.001
2;1;8;7;1;1;0.001
...
```

Figure 615: Example of the table $TAB_EL1 in block #GEOMETRY.elements.

6.4.3 Preprocessing-Solver Setting Input Data

All required solver configuration data are stored in the file with the extension [*.fexcfg]. The formatting style of the file is the same as in the [*.fexin] file. The considered data

content varies according to the type of the solver. The input configuration data for the network solver contains additional data related to the initialization and configuration of the computer network. Other data not contained in the input files come into the FEXP solver start process as a command line arguments and they are explained as last.

File Block—*#SERVER.init*

This block is used to initialize the network server for the hybrid parallel type of FEXP solver. Currently it contains one table only, namely $TAB_IPS. Table $TAB_IPS contains the initial number of network-client nodes except that server. A formal representation of the table $TAB_IPS with the illustrative values is shown in the Tab. 613.

Table 613: Table $TAB_IPS.

id [–]	**ip_address** [–]
1	127.0.0.1
2	127.0.0.1
...	

Table 614: Description of the table $TAB_IPS.

Column Name	Description	Value
id	Serial number of the network-client node.	$\mathbb{S}\backslash\{0\}$
ip_address	IP address of network-client node.	Valid IPv4 address

Descriptions of an individual columns of table $TAB_IPS are contained in the Tab. 614. An example of a respective block in an input text file with table $TAB_IPS has the formatted form shown in the Fig. 616.

```
#SERVER.init
$TAB_IPS;id;ip_address
1;127.0.0.1
2;127.0.0.1
...
```

Figure 616: Example of the table $TAB_IPS in block #SERVER.init.

File Block—*#SOLVER.input_files*

This block contains the list of input files containing separated macro model data. Multiple files representing individual macro model data relate primarily to the hybrid parallel mutation of the FEXP solver. In the case of a workstation out of network cluster of the FEXP solver, the form of structural input data is arbitrary. A formal representation of the table $TAB_FLS with the illustrative values is shown in the Tab. 615.

Table 615: Table $TAB_FLS.

id [–]	**path** [–]
1	C:\...\macro_1.fexin
2	C:\...\macro_2.fexin
...	

Table 616: Description of the table $TAB_FLS.

Column Name	Description	Value
id	Structural model identifier number.	$\mathbb{S}\backslash\{0\}$
path	Structural model (macro) input data file.	Valid Windows OP file path

The respective block contains one more table, namely $TAB_FLS_DFLT, which concerns to the hybrid form of the FEXP solver. Such a referenced file is sent to the network client-node when it is dynamically initialized. Its definition is contained in the Tab. 617.

Table 617: Table $TAB_FLS_DFLT.

path [–]
C:\...\default_file_1.fexin

Table 618: Description of the table $TAB_FLS_DFLT.

Column Name	Description	Value
path	Structural model (macro) input data file.	Valid Windows OS file path

Descriptions of the individual columns of the tables are contained in Tab. 616 and Tab. 618, respectively. An example of a respective block in an input text file with tables $TAB_FLS and $TAB_FLS_DFLT has the formatted form shown in the Fig. 617.

```
#SOLVER.input_files
$TAB_FLS;id;path
1;C:\...\macro_1.fexin
2;C:\...\macro_1.fexin
...
\$TAB_FLS_DFLT;path
C:\...\default_file_1.fexin
```

Figure 617: Example of tables $TAB_FLS and $TAB_FLS_DFLT in block #SOLVER.input_files.

File Block—*#SOLVER.calc_behaviour*

This block contains settings related to the computation result data output. This concerns both the exporting of the detailed simulation results and the observation of the computation behaviour for the selected degree of freedom of a some FE node. A formal representation of the table $TAB_CALC with the illustrative values is shown in the Tab. 619.

Table 619: Table $TAB_CALC.

nid [–]	**dof** [–]	**text_out** [–]	**res_name** [–]	**res_dir** [–]
1	3	1	result	C:\...\RESULTS\

Table 620: Description of the table $TAB_CALC.

Column Name	Description	Value
nid	FE node identifier number.	$\mathbb{S}\backslash\{0\} \in$ **id**
dof	Nodal degree of freedom.	$\mathbb{S} \in \{1, \ldots, 6\}$.
text_out	Runtime degree of freedom value behaviour.	$\mathbb{S}\backslash\{0\}$, 1—on, 0—off
res_name	Name of results.	Valid Windows OS file name without file an extension
res_dir	Results directory path.	Valid Windows OS directory path

Descriptions of the individual columns of table are contained in the Tab. 620, respectively. An example of a respective block in an input text file with table $TAB_CALC has the formatted form shown in the Fig. 618.

```
#SOLVER.calc_behaviour
$TAB_CALC;nid;dof;text_out;res_name;res_dir
1;3;1;result;C:\...\RESULTS\
```

Figure 618: Example of the table $TAB_CALC in block #SOLVER.calc_behaviour.

6.4.4 Finite Element Model Assembly

The data structures used and the process of an FE model assembly are described. The data model assembly for an explicit numerical method is somewhat different from the standard approach of implicit numerical methods. In the implicit approach, it is necessary to assemble a global stiffness matrix not always symmetric. It requires the application of a special data structures capable of efficiently handling these sparse matrices that generally arise in the FEM. The problem becomes more complicated when it is necessary to use such a data structure for parallel computations, when the assembly of a global stiffness matrix becomes a narrow throat of an entire computation.

Between the already developed such a data structures capable of doing the aforementioned onerous task by an efficient way belongs a data structure based on the algorithm developed by D. Langr et al. (see [70], [71]) from the Faculty of Information Technology of the CTU in Prague. The commonly used data structures for sparse matrix storage include those contained in library LAPACK (abbreviation of the Linear Algebra PACKage, see http://www.netlib.org/lapack/) or BLAS (see https://software.intel.com/en-us/mkl-developer-reference-c-overview) in the commercial library Intel®Math Kernel Library (MKL).

The explicit method does not require the use of these special data structures, so for the purpose of data storage it is sufficient to use the standard data structures as arrays, vectors or maps. These data structures are often already included in the standard libraries of a particular programming language. In the case of C++ programming language, a number of such data structures are contained in the STL (abbreviation of the Standard Template Library, see http://www.cplusplus.com/reference/stl/) library.

However, an important requirement for the functionality of these data structures is their thread safety while attempting to read data from them from a more than one thread. The requirement for the thread safety of the read operation is met in the case of used data containers from the STL library. In the parts where is a need to change their contents, the write operation is secured using the appropriate logical structure of the program and by using of some synchronization operations, which are currently a part of standard C++ libraries. Since the entire FEXP solver and the other supporting programs use the asynchronous approach to process many types of computing operations, a lot of different types of synchronization operations in many representations are applied.

Due to the trend in computer technology, which is going to a lot of effort to increase the number of processor cores rather than to increase processor performance through its clock frequency, software support for multithreaded control has increased, thus so much required functionality no longer has to be complicatedly programmed.

The most used synchronization in the FEXP solver involves synchronizing threads by the thread barrier. Such synchronization procedures for CPU threads is described in detail in article of V. Rek and I. Němec (see [105]). For the data synchronization during write operation, the so-called *critical sections* (CS) are used. Unfortunately, however, all synchronization procedures reduce the performance of the parallel application and must be considered as a sequential running code in the performance analysis (see [28]).

Data Containers

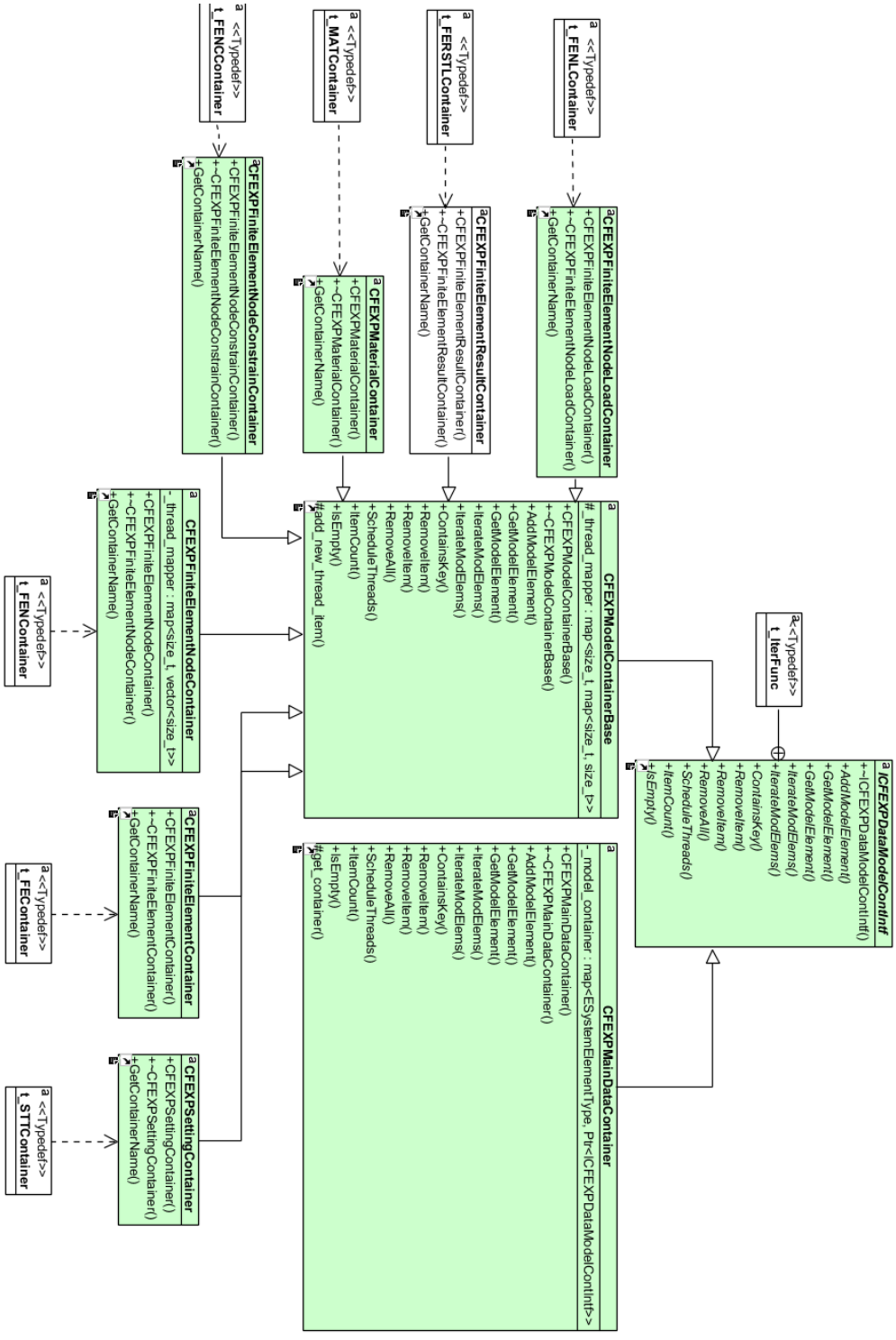


Figure 619: UML diagram of designed container classes.

For the model data storage purpose, a data container system has been designed. The current structure of the used data containers is presented in the Fig. 619, and a short description of their purpose is listed in the Tab. 621.

Table 621: Description of the container classes.

Class Name	Purpose
CFEXPFiniteElementNodeContainer	It keeps all the FE nodes.
CFEXPFiniteElementContainer	It keeps all the FEs.
CFEXPMaterialContainer	It keeps all defined materials.
CFEXPSettingContainer	It keeps all solver settings except special setting for the network server.
CFEXPFiniteElementNodeConstrainContainer	It keeps all defined nodal constrains.
CFEXPFiniteElementNodeLoadContainer	It keeps all defined nodal loads.
CFEXPMainDataContainer	It is a central repository for accessing model data.

Adding individual elements to the container storage while the model assembly process is performed through the interface represented by the abstract **class ICFEXPDataModelContIntf**, which is implemented by all data storage containers in the FEXP solver. The central data repository is represented by the **class CFEXPMainDataContainer**, which encapsulates all the other data containers to which it facilitates the access. Instance assembly of this container class presents the following fragment of the respective source code (class constructor from the source code file `FEXPDataContainer.cpp`)

```
CFEXPMainDataContainer::CFEXPMainDataContainer()
{
  // container for settings
  _model_container[ESystemElementType::eSetting   ] =
    FEXPCOMMON_STACAST(t_STTContainer, ICFEXPDataModelContIntf,
    CFEXPDataManager<t_STTContainer>::SafeAllocInstance());

...

  // container for finite element nodes
  _model_container[ESystemElementType::eNode      ] =
    FEXPCOMMON_STACAST(t_FENContainer, ICFEXPDataModelContIntf,
      CFEXPDataManager<t_FENContainer>::SafeAllocInstance());
  // container for finite elements
  _model_container[ESystemElementType::eElement   ] =
    FEXPCOMMON_STACAST(t_FEContainer , ICFEXPDataModelContIntf,
      CFEXPDataManager<t_FEContainer>::SafeAllocInstance());

...
}
```

The next functionality related to parallel data processing is explained in the section describing the individual parts of the process of numerical computations.

Process of Data Model Assembly

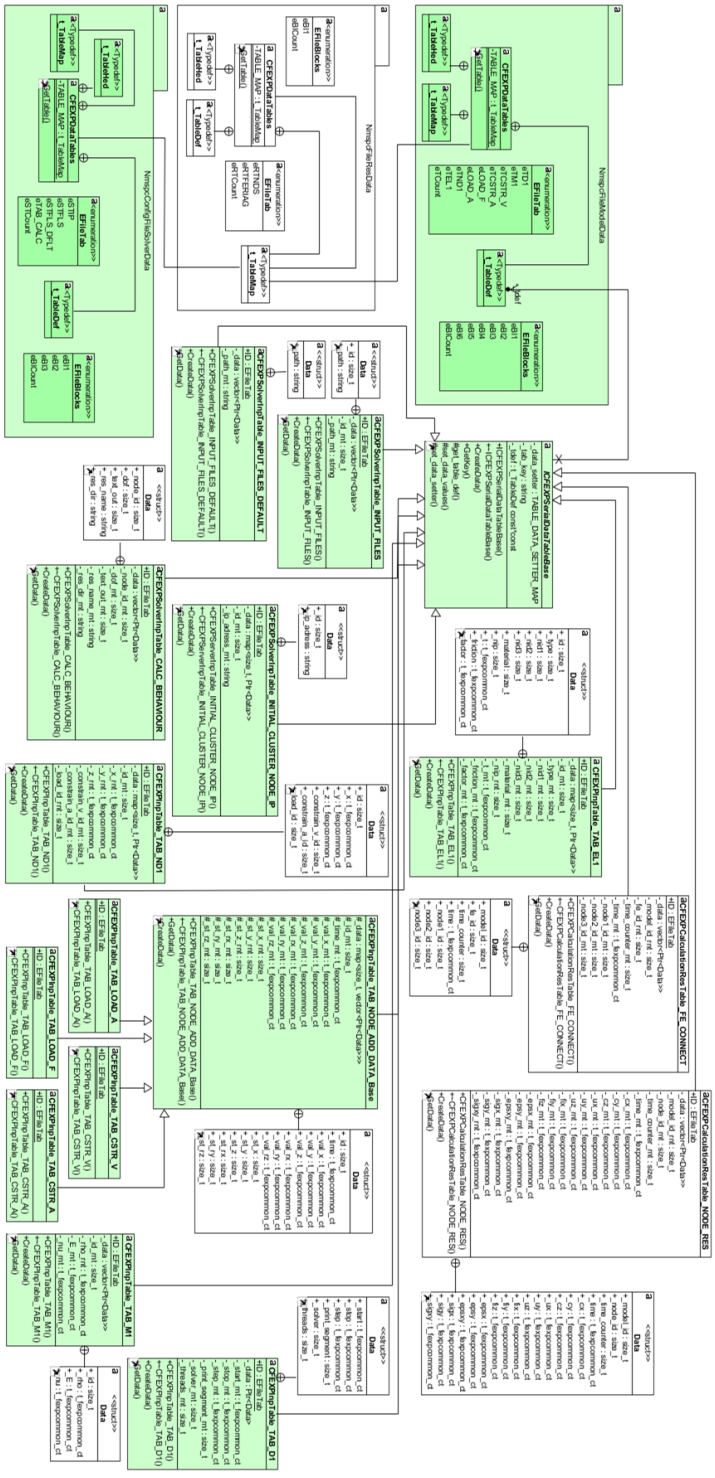


Figure 620: UML diagram of designed structure for tabulated I/O data.

For the general data model assembly in the FEXP solver, a multi-level system of input tables processing was designed with the final composition of a computing data. The Fig. 620 presents the current structure state of tables containing data from rough deserialation/serialization of files. It is based on the predefined structure of table columns in the FEXP source code file `FEXPSerialization.cpp`. A fragment of source code for table columns definition (base solver setting, isotropic linearly elastic material) is as follows

```
// Structure of tables in model data input file
NmspcFileModelData::CFEXPDataTables::t_TableMap
NmspcFileModelData::CFEXPDataTables::TABLE_MAP =
{
  { INP_FILE_STRUCT_BLCS[NmspcFileModelData::EFileBlocks::eBl1],
    {
      { INP_FILE_STRUCT_TABS[NmspcFileModelData::EFileTab::eTD1],
        {{ "start"   , FEXPCOMMON_CLCFLT_TYPE_NAME },
         { "stop"    , FEXPCOMMON_CLCFLT_TYPE_NAME },
         { "step"    , FEXPCOMMON_CLCFLT_TYPE_NAME },
         { "print"   , FEXPCOMMON_SIZE_T_TYPE_NAME },
         { "solver"  , FEXPCOMMON_SIZE_T_TYPE_NAME },
         { "threads", FEXPCOMMON_SIZE_T_TYPE_NAME }}}
    }
  },
  { INP_FILE_STRUCT_BLCS[NmspcFileModelData::EFileBlocks::eBl2],
    {
      { INP_FILE_STRUCT_TABS[NmspcFileModelData::EFileTab::eTM1],
        {{ "id" , FEXPCOMMON_SIZE_T_TYPE_NAME },
         { "rho", FEXPCOMMON_CLCFLT_TYPE_NAME },
         { "E"  , FEXPCOMMON_CLCFLT_TYPE_NAME },
         { "nu" , FEXPCOMMON_CLCFLT_TYPE_NAME }}}
    }
  },

  ...
}
```

From the sample of a source code it is obvious that the table columns definition consists of an appropriate name and the data type to be used for text parsing. The data type names definition constants contains source code file `FEXPCommon.h`, and is follows:

```
////--> Floating point data type for computations
using t_fexpcommon_ct = double;

...

////--> Primitive data types
#define FEXPCOMMON_DATA_TYPE(type) std::type_index(typeid(type))
#define FEXPCOMMON_DATA_TYPE_NAME(type) FEXPCOMMON_DATA_TYPE(type).name()
// type index of common types
#define FEXPCOMMON_CLCFLT_TYPE      FEXPCOMMON_DATA_TYPE(t_fexpcommon_ct)
#define FEXPCOMMON_SIZE_T_TYPE      FEXPCOMMON_DATA_TYPE(size_t         )
#define FEXPCOMMON_STRING_TYPE      FEXPCOMMON_DATA_TYPE(std::string    )
// data type represantation of used types
#define FEXPCOMMON_CLCFLT_TYPE_NAME FEXPCOMMON_DATA_TYPE_NAME(t_fexpcommon_ct)
#define FEXPCOMMON_SIZE_T_TYPE_NAME FEXPCOMMON_DATA_TYPE_NAME(size_t         )
#define FEXPCOMMON_STRING_TYPE_NAME FEXPCOMMON_DATA_TYPE_NAME(std::string    )

...
```

For the tabular data launching process, a more general system of working with spreadsheets was designed based on their respective defined structure. Such functionality contains abstract **class ICFEXPSerialDataTableBase** in source files `FEXPSerializeDataTables.h`

and `FEXPSerializeDataTables.cpp`, respectively. It provides a common interface of an input tables containing rough parsed data as their base class. Parsing of individual input data tables with a validity check is performed using the following source code

```
void ICFEXPSerialDataTableBase::set_data_values(
const std::vector<std::string> & tab_def, const std::vector<std::string> & tab_data,
TABLE_LAMBDA_AFTRCL lambda_rw_set)
{
#define ERROR_WRONG_COL_COUNT "@Error: Input table has wrong number of columns: \""
  auto table = get_table_def();
  auto crrsz = tab_def.size();
  if (table.size() != crrsz)
  {
    std::string error = ERROR_WRONG_COL_COUNT + GetKey() + "\" --> \"";
    error += CFEXPBaseConvers::NumberToString(crrsz) + "\":";
    error += "\"";
    error += FEXPCOMMON_NEW_LINE;
    FEXPCOMMON_FOREACH_ITER_FNC(tab_def, { error += IT + FEXPCOMMON_DELIMITER; });
    error += "\"!!!" + std::string(FEXPCOMMON_NEW_LINE);
    error += "--> Right col. num.: \""
        + CFEXPBaseConvers::NumberToString(table.size()) + "\" --> \"";
    FEXPCOMMON_FOREACH_ITER_FNC(table, { error += IT.first + FEXPCOMMON_DELIMITER; });
    error += "\".";
    FEXPCOMMON_EXCEPTION(error);
  }

#define ERROR_WRONG_COL_RUNTIME "Error: Table does not contain col. def: \""
  FEXPCOMMON_FOREACH_ITER(tab_data)
  {
    size_t counter  = FEXPCOMMON_DEFAULT_VALUE;
    auto str_values = CFEXPBaseConvers::SplitString(*IT, FEXPCOMMON_DELIMITER);
    FEXPCOMMON_FOREACH_ITER_FNC(tab_def,
      {
        if (!_data_setter.count(IT))
          FEXPCOMMON_EXCEPTION(ERROR_WRONG_COL_RUNTIME + GetKey()
                + "\" --> \"" + IT + "\"!!!");
        _data_setter[IT](str_values[counter++]);
      });
    // call after one row data set
    lambda_rw_set();
  }
}
```

This code uses the inner functionality of the classes representing the input data tables to assembly those specific data as follows (table for model structure materials, **class CFEXPInpTable_TAB_M1**)

```
CFEXPInpTable_TAB_M1::CFEXPInpTable_TAB_M1()
  : ICFEXPSerialDataTableBase(INP_FILE_STRUCT_TABS[ID],
   INP_FILE_TAB(INP_FILE_STRUCT_BLCS[NmspcFileModelData::EFileBlocks::eBl2], ID))
{
  set_data_setter("id" , FEXPCOMMON_SIZE_T_TYPE, DATA_TO_SET_LAMBDA_BODY(
  { _id_rnt  = CFEXPBaseConvers::StringToNumber<size_t>          (txt); }));
  set_data_setter("rho", FEXPCOMMON_CLCFLT_TYPE, DATA_TO_SET_LAMBDA_BODY(
  { _rho_rnt = CFEXPBaseConvers::StringToNumber<t_fexpcommon_ct>(txt); }));
  set_data_setter("E"  , FEXPCOMMON_CLCFLT_TYPE, DATA_TO_SET_LAMBDA_BODY(
  { _E_rnt   = CFEXPBaseConvers::StringToNumber<t_fexpcommon_ct>(txt); }));
  set_data_setter("nu" , FEXPCOMMON_CLCFLT_TYPE, DATA_TO_SET_LAMBDA_BODY(
  { _nu_rnt  = CFEXPBaseConvers::StringToNumber<t_fexpcommon_ct>(txt); }));
}
```

It should be noted that the order of the table columns can be arbitrary, amount of each table columns, column's names and the data type must match their definition. The list of

classes with a rough description is in the Tab. 622 representing the individual tables for the structure model input data described in chapter 6.4.2.

Table 622: Description of the classes for the tabulated structural input data.

Class Name	Purpose
CFEXPInpTable_TAB_D1	Data for the control of numerical simulation.
CFEXPInpTable_TAB_M1	Data for the homogenuous isotropic linerly elastic materials.
CFEXPInpTable_TAB_CSTR_V	Data for the velocity constrain conditions.
CFEXPInpTable_TAB_CSTR_A	Data for the acceleration constrain conditions.
CFEXPInpTable_TAB_LOAD_F	Data for the nodal force loads.
CFEXPInpTable_TAB_LOAD_A	Data for the nodal acceleration loads.
CFEXPInpTable_TAB_ND1	Data for the FE nodes.
CFEXPInpTable_TAB_EL1	Data for the triangular 3-noded shell FE.

The system of described table classes represents the lowest layer of input data launching process. For the computing data model assembly, a special class which uses an asynchronous paradigm to perform such a task was designed (see Fig. 621).

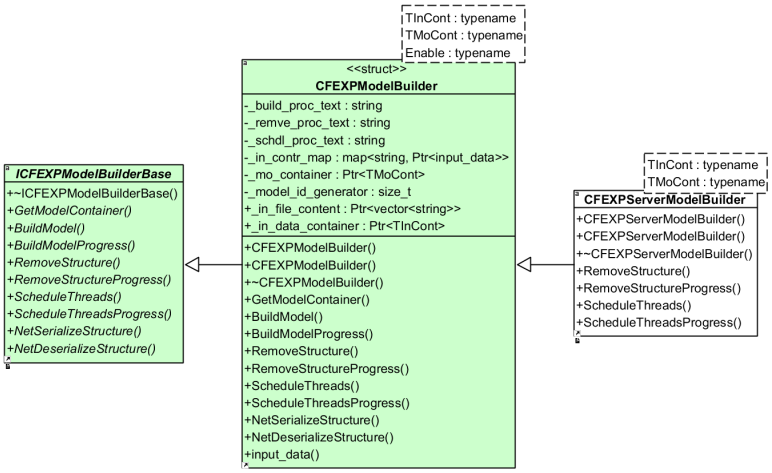


Figure 621: UML diagram of model builder template classes.

The **class CFEXPModelBuilder** also provides central access to the possibility of the reconfiguration of a numerical model situated on a single workstation. This option applies mainly to the computation of large models composed from many separate structural macro parts. It also provides access to serialization/deserialization of a computational model data at runtime for the network transfers and also provides the possibility of re-scheduling used threads after network data transfer. The primary purpose is to assemble a data model that represents the following fragment of a source code

```
/** @brief Main function (single workstation).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{
...

  // builder instance
  Ptr<ICFEXPModelBuilderBase> builder = CFEXPDataManager<
    CFEXPModelBuilder<ICFEXPDataContIntf, CFEXPMainDataContainer>>
      ::SafeAllocInstance(
        "Building finite element model",
        "Removing finite element model",
        "Scheduling of threads");

...

  // build FE data model
  auto build_succ = true;
  FEXPCOMMON_FOREACH_ITER(file_reader_map)
  {
    if (builder->BuildModelProgress(IT->first, IT->second->GetFileContent(),
      FEXPCOMMON_DYNCAST(CFEXPFEInpContBase, ICFEXPDataContIntf,
        IT->second->GetInputContainer())))
      continue;
    build_succ = false;
    break;
  }

...
}
```

The specific functionality for the final composition of the FE model data is contained in **class CFEXPInpDataModelAssemblyFactory**. The input data file for the FEXP solver configuration is processed by a similar class named **class CFEXPSolverInpDataAssemblyFactory** in source files `FEXPSerializeData.h` and `FEXPSerializeData.cpp`, respectively. The input table processing for assembling a specific type of material contains the following source code fragment

```
void CFEXPInpDataModelAssemblyFactory::AssemblyMaterials(
Ptr<ICFEXPDataModelContIntf> & model_cont, CFEXPInpDataContainer & incont)
{
#define WARNING_MATERIAL "Warning: Duplicated material ID in input data" ...
 "(the old one will be considered): "
  auto block = incont.GetData<CFEXPCalcMaterial>();
  if (!block)
    return;
  auto table = block->GetTableData<CFEXPInpTable_TAB_M1>();
  if (table)
  {
    FEXPCOMMON_FOREACH_ITER_FNC(table->GetData(),
      {
        // check unique material id
        auto mat_to_add = FEXPCOMMON_STACAST(CFEXPSimpleElasticMaterial,
        ICFEXPModelDataIntf, CFEXPDataManager<CFEXPSimpleElasticMaterial>
          ::SafeAllocInstance(IT->_id, IT->_rho, IT->_E, IT->_nu));
        if (model_cont->ContainsKey(mat_to_add->GetId(), mat_to_add->GetType()))
        {
          CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + WARNING_MATERIAL);
          CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "--> " +
            CFEXPBaseConvers::NumberToString<size_t>(mat_to_add->GetId()));
          return;
        }
        // add material to container
```

```
        model_cont->AddModelElement(mat_to_add, mat_to_add->GetType());
      });
  }

  // add rigid default material
  auto rig_material = FEXPCOMMON_STACAST(CFEXPRigidMaterial, ICFEXPModelDataIntf,
    CFEXPDataManager<CFEXPRigidMaterial>::SafeAllocInstance(
      FEXPCOMMON_DEFAULT_VALUE, FEXPCOMMON_DEFAULT_VALUE,
      FEXPCOMMON_DEFAULT_VALUE, FEXPCOMMON_DEFAULT_VALUE));
  model_cont->AddModelElement(rig_material, rig_material->GetType());
}
```

The given functionality for assembly the specific data model is invoked for the particular type of container that stores the data. It presents the following source code from source file `FEXPModelBuilder.h`:

```
template<typename TInCont, typename TMoCont>
bool CFEXPModelBuilder<TInCont, TMoCont,
typename std::enable_if
<
std::is_base_of<ICFEXPDataContIntf      , TInCont>::value &&
std::is_base_of<ICFEXPDataModelContIntf, TMoCont>::value>::type
>
::BuildModel(const std::string & key,
Ptr<std::vector<std::string>> fcontent, Ptr<TInCont> data)
{
  if (_in_contr_map.count(key))
    return false;
  _in_contr_map.insert(MAP_PAIR(key,
    CFEXPDataManager<input_data>::SafeAllocInstance(fcontent, data)));
  data->SetData(FEXPCOMMON_STACAST(TMoCont, ICFEXPDataModelContIntf, _mo_container), key);
  return true;
}
```

After the whole model has been assembled and before the start of computation, the last part comes into play, which takes care of scheduling of the FE model to the planned amount of the used threads for parallel computation. As mentioned earlier, such activity is turned on from the outside using the **class CFEXPModelBuilder** instance and is represented by the following fragment of source code:

```
/** @brief Main function (single workstation).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{
...

  if (!build_succ)
  {
    FEXPCOMMON_CONSOLE_PAUSE(std::get<FEXPCOMMON_CMD_MANAGER_INDEX>(cmd));
    return EXIT_FAILURE;
  }

  // thread scheduling of FE nodes and FEs
  builder->ScheduleThreadsProgress(ESystemElementType::eNode   );
  builder->ScheduleThreadsProgress(ESystemElementType::eElement);

...
}
```

Thread scheduling to finite element nodes and also to finite elements of mesh is performed by the following code contained in the source file `FEXPDataContainer.cpp`:

```
void CFEXPModelContainerBase::ScheduleThreads(size_t opt_id)
{
  auto element_count = get_container().size();
  if (!element_count)
  {
    _thread_mapper.clear();
    return;
  }
  // thread scheduling
  auto counter = size_t(FEXPCOMMON_DEFAULT_VALUE); _thread_mapper.clear();
  CFEXPModelContainerBase::IterateModElems([this, element_count, &counter](auto item)
    {
      auto threadid = CFEXPConcurencyTools::GetCpuThreadIdToItem(
        counter++, item->GetThreadNumber(), element_count);
      item->SetThreadId(threadid);
      // map item to thread
      if (threadid)
        add_new_thread_item(item->GetThreadId(), item->GetId());
      return true;
    }, FEXPCOMMON_DEFAULT_VALUE);
}
```

6.4.5 FEXP Computational Parts

In this chapter, the FEXP solver part belonging to the numerical computation is dealt. This such part consists of three levels that relate to the interfaces that trigger the whole computation, the level that takes care of the sequence of an individual computational steps like computation represented by the integration of internal forces and direct integration of equtions of motion, and the most bottom level directly focused on the pure numerics of the mentioned integrations. The relevant numerical background of related numerical computations is described in chapter 3.

Here, the composition of an explicit computation and required synchronization blocks are described. The type of FEXP solver, which is designed for computation in the scope of a computer network, is covered here only partially. A detailed description is contained in chapter 6.4.8. Specific numerical integrations are included in the respective implementation of the finite elements. In the FE nodes an integration of the equations of motion is performed, which is based on the already integrated internal and external forces from the FEs connected to the respective FE node. This section is presented in an indicative manner only. In the case of implementation of the other FE types in the near future, it offers the possibility of optimization using some of the GPGPU technology for selected matrix operations, even for the entire force integration process as described in the article of V. Rek and I. Němec (see [104]).

Type of Solvers

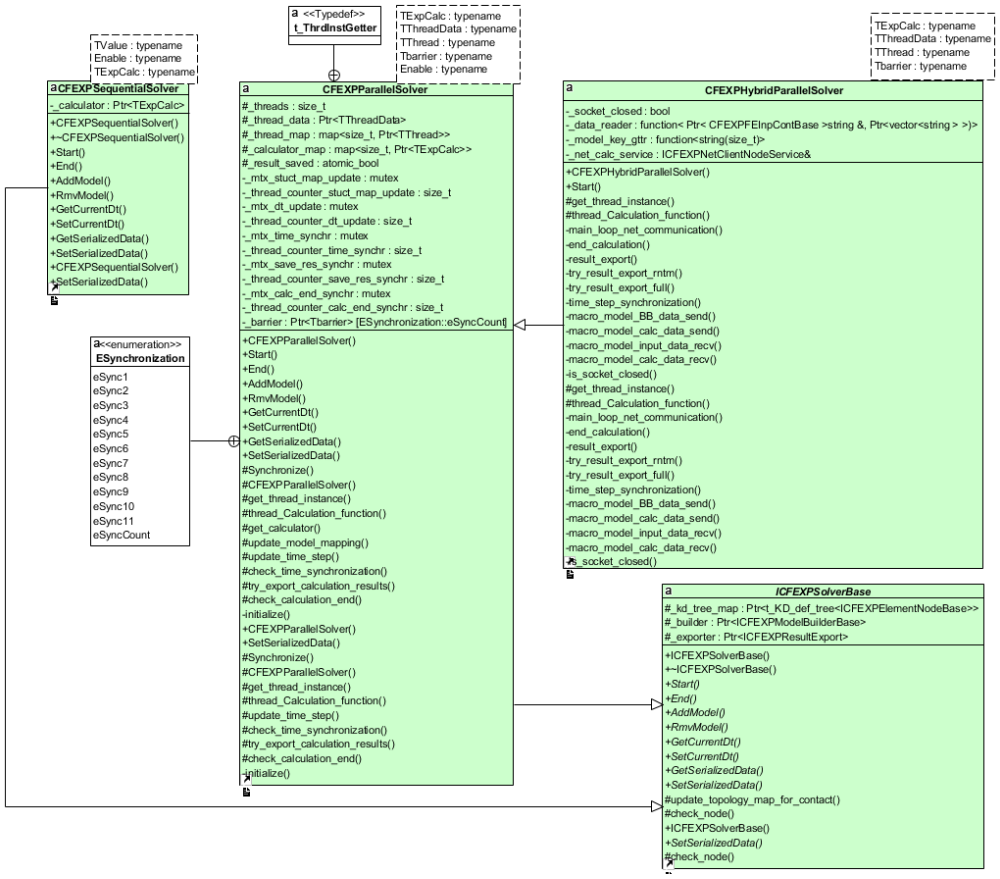


Figure 622: UML diagram of classes for solver types.

This part represents the highest layer of the object oriented design of the FEXP software solution regarding a numerical computations. Especially for the testing purposes, three main types of solvers were designed. The implementation of the appropriate classes is found in the source code files `FEXPSolver.h` and `FEXPSolver.cpp`, respectively. Classes are the following

0. Sequential solver is represented by the **class CFEXPSequentialSolver**.

1. Parallel solver is represented by the **class CFEXPParallelSolver**.

2. Hybrid parallel solver is represented by the **class CFEXPHybridParallelSolver**.

Sequential type of FEXP solver was designed primarily for the purpose of analyzing the effectiveness of individual solutions as a source of reference value. Its secondary purpose is to debug the numerical computation with respect to the validity of numerical simulation results. The sequential type of the FEXP solver contains the basic form of an explicit

numerical computation control. Thus, such a form as the numerical computation control is represented by the following source code:

```
template<typename TExpCalc>
void CFEXPSequentialSolver<TExpCalc,
typename std::enable_if
<
std::is_base_of<ICFEXPExplicitCalcBase, TExpCalc>::value
>::type>::Start()
{
  try
  {
    auto result_saved = false;
    // Check simulation end
    while (true)
    {
      // 1. Step: Check end of calculation
      if (_calculator->CheckEnd(FEXPCOMMON_DEFAULT_VALUE))
      {
        if (!result_saved)
          _calculator->TrySaveResults(FEXPCOMMON_DEFAULT_VALUE, true);
        break;
      }
      // 2.  Step: Time increment consistency
      _calculator->SimulationTimeIncrement      (FEXPCOMMON_DEFAULT_VALUE);
      // 3.  Step: Prepare data for new time level
      _calculator->PrepareDataForNewTimeLevel   (FEXPCOMMON_DEFAULT_VALUE);
      // 4.  Step: Update mapping of fe nodes for contact searching
      _calculator->UpdateModelMapping           (FEXPCOMMON_DEFAULT_VALUE);
      // 5.  Step: Transform position and velocities to local coordinate system
      _calculator->GlobalToLocalTransformation  (FEXPCOMMON_DEFAULT_VALUE);
      // 6.  Step: Integration of internal, external and contact forces
      _calculator->CalculateForces              (FEXPCOMMON_DEFAULT_VALUE);
      // 7.  Step: Calculate new displacemnets based on explicit integr. of EOM
      _calculator->CalculateNewGeometry         (FEXPCOMMON_DEFAULT_VALUE);
      // 8.  Step: Save calculation results
      result_saved = _calculator->TrySaveResults(FEXPCOMMON_DEFAULT_VALUE);
      // 9.  Step: Print out results
      _calculator->PrintOutResults              (FEXPCOMMON_DEFAULT_VALUE);
      // clear all results --> data space optimization
      _calculator->ClearResults                 (FEXPCOMMON_DEFAULT_VALUE);
      // 10. Step: Control of numerical stability
      _calculator->StabilityControl             (FEXPCOMMON_DEFAULT_VALUE);
      // 11. Step: Update time step dt
      _calculator->SetNewTimeStep(_calculator->GetCalculatedCriticTimeStep());
    }
  }
  catch (const std::exception & ex)
  {
    CFEXPLog::WriteLine();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT +
      "Error: Computation ended before the simulation end.");
    // print out the problem of exception
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "An exception occurred:");
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "-> " + ex.what());
  }
}
```

The entire process of sequential computation is commenced up by the following snippet of a source code:

```
/** @brief Main function (single workstation).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{
```

```
...

  switch (setting->GetSolver())
  {
  case CFEXPBaseSetting::EFEXPSolverType::eSequential : // sequential type of solver
    CFEXPDataManager<CFEXPSequentialSolver<CFEXPCalculation>>
      ::SafeAllocInstance(builder, result_exporter)->Start();
    break;

...

  default: FEXPCOMMON_EXCEPTION("Error: Unknown type of a solver!");
  }

...
}
```

The next type of FEXP solver already uses a parallel run with utilization of CPU cores. Its structure is much more complex than in the case of the sequential solver type. Here it is necessary to ensure the data synchronization, as well as the synchronization of the sequences refering to the individual steps of numerical computations. The approach to the synchronization between the individual steps of a numerical computations was chosen to be identical to the approach originally designed to the use of NVIDIA®CUDA® technology for GPGPU from the article of V. Rek and I. Němec (see [104]).

As mentioned earlier, synchronization of the thread progression between the individual steps of the numerical computations is carried out by means of thread barriers. Thus, the internal structure of the FEXP parallel solver representing the execution and synchronization of the individual computational phases is presented by the following source code:

```
template<typename TExpCalc, typename TThreadData, typename TThread, typename Tbarrier>
void CFEXPParallelSolver<TExpCalc, TThreadData, TThread, Tbarrier, typename std::enable_if
<
std::is_base_of<ICFEXPExplicitCalcBase    , TExpCalc    >::value &&
std::is_base_of<ICFEXPThreadDataBase      , TThreadData >::value &&
std::is_base_of<ICFEXPThreadBase          , TThread     >::value &&
std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier    >::value
>::type>
::thread_Calculation_function(size_t thread_id, Ptr<TThreadData> data)
{
  try
  {
    bool calculation_end = false; auto calculator = get_calculator(thread_id);
    while (true)
    {
      // 1. Step: Set time consistency
      data->SetCalcTimeIncrement(thread_id,
        calculator->SimulationTimeIncrement(thread_id));
      Synchronize(ESynchronization::eSync1);
      // 2. Step: Check if time is the same for all threads
      // this require only one thread to do this action
      check_time_synchronization(data);
      // check end of calculation --> wait for thread saving results
      calculation_end = check_calculation_end(data, thread_id);
      Synchronize(ESynchronization::eSync2);
      // 3. Step: Check end of computation
      if (calculation_end)
        break;
      Synchronize(ESynchronization::eSync3);
```

```
      // 4.   Step: Prepare data for new time level
      calculator->PrepareDataForNewTimeLevel(thread_id);
      // 5.   Step: Update mapping of fe nodes for contact search
      // this require only one thread to do this action
      update_model_mapping(thread_id);
      // 6.   Step: Transform position and velocities to local coordinate system
      calculator->GlobalToLocalTransformation(thread_id);
      Synchronize(ESynchronization::eSync4);
      // 7.   Step: Integration of internal, external and contact forces
      calculator->CalculateForces(thread_id);
      Synchronize(ESynchronization::eSync5);
      // 8.   Step: Compute new displacements based on explicit integration of EM
      calculator->CalculateNewGeometry(thread_id);
      Synchronize(ESynchronization::eSync6);
      // 9.1. Step: Export results
      try_export_calculation_results(data, thread_id);
      // 9.2. Step: Print out results
      calculator->PrintOutResults(thread_id);
      Synchronize(ESynchronization::eSync7);
      // 10.  Step: Control of numerical stability
      calculator->StabilityControl(thread_id);
      data->SetTimeStep(thread_id, calculator->GetCalculatedCriticTimeStep());
      Synchronize(ESynchronization::eSync8);
      // 11.  Step: Update time step dt
      // this require only one thread to do this action
      update_time_step(data);
      // 12.  Step: Synchronize befor new loop
      Synchronize(ESynchronization::eSync9);
    }
  }
  catch (const std::exception & ex)
  {
    CFEXPLog::WriteLine();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT +
      "Error: Calculation ended before simulation end.");
    // print problem of exception
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "An exception occurred:");
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "--> " + ex.what());
  }
}
```

Similarly as in the case of sequential type of FEXP solver, the entire process of parallel computation is started up by the following snippet of source code:

```
/** @brief Main function (single workstation).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{
...

  switch (setting->GetSolver())
  {

...

  case CFEXPBaseSetting::EFEXPSolverType::eParallelCpu: // parallel type of solver
    CFEXPDataManager<SIMPLE_PARALLEL_SOLVER>
      ::SafeAllocInstance(builder, result_exporter)->Start();
    break;

...

  default: FEXPCOMMON_EXCEPTION("Error: Unknown type of a solver!");
  }
```

```
...
}
```

The last type of the FEXP solver is a hybrid-parallel solver running on a computer network. Compared to the previously mentioned two types of FEXP solvers, it absolutely requires special handling. Parallelization is based on a hybrid form of domain decomposition that uses a parallel processing approach identical to those on a single workstation, with the approach based on the data migration of macro entities between the individual workstations of the considered computer network. For the given reason, an executable program for one workstation was partitioned into a part belonging to a single workstation out of the cooperating workstations connected to the computer cluster, and a portion dedicated to one cooperative workstation within the computer cluster.

As a result, this type of FEXP solver is not considered to run in the same way as the previous two solvers. It represents the following snippet of a source code:

```
/** @brief Main function (single workstation).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{
...

  switch (setting->GetSolver())
  {

...

  case CFEXPBaseSetting::EFEXPSolverType::eHybrid      :
    FEXPCOMMON_EXCEPTION("Error: Hybrid solver is not allowed here!");
  default: FEXPCOMMON_EXCEPTION("Error: Unknown type of a solver!");
  }

...
}
```

Thanks to the already mentined reasons and many others, such a type of the FEXP solver is devoted to the entire chapter dealing with the proposed solution for dynamic simulation on a computer network.

Explicit Numerical Computations

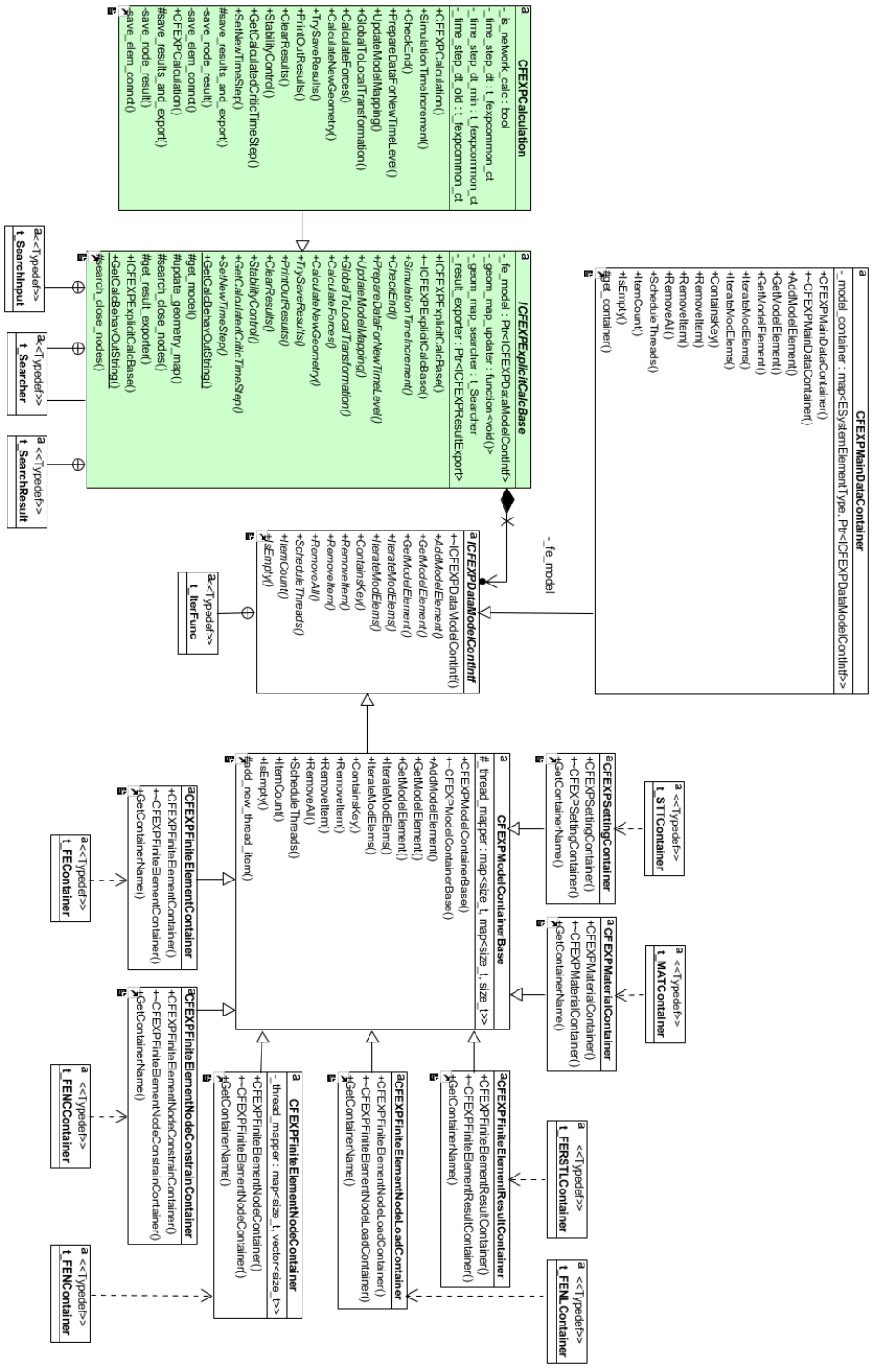


Figure 623: UML diagram of class for explicit computation.

Full control and triggering of the individual computational steps is performed by the appropriate type of solver described in previous text. The individual steps of an explicit numerical computation, its behaviour control, and the data export are included in the **class CFEXPCalculation** contained in the source files `FEXPCalculation.h` and `FEXPCalculation.cpp`, respectively. It is presented to the outside through the implementation of the interface the abstract **class ICFEXPExplicitCalcBase** as showen in the Fig. 623. The numerical computations within are performed on the data elements stored in the central storage represented by the **class CFEXPMathModelElementContainer** as shown in the Fig. **??**. Triggering of a numerical computation an internal and contact forces represents the following snippet of the source code:

```
/** @brief It computes forces (internal, external, contact).
*/
void CFEXPCalculation::CalculateForces(size_t thread_id)
{
  // compute external and internal forces
  get_model()->IterateModElems([this](auto item)
    {
      auto element = FEXPCOMMON_STACAST(ICFEXPModelDataIntf, ICFEXPElementBase, item);
      // 1. stresses
      element->CalcStress(_time_step_dt);
      // 2. internal forces
      element->CalcIntForce();
      // 3. contact forces
      std::map<size_t, Ptr<ICFEXPElementNodeBase>> element_nodes_map;
      auto nodes = element->GetNodes();
      FEXPCOMMON_FOREACH_ITER(nodes)
        element_nodes_map.insert(MAP_PAIR(IT->lock()->GetId(), IT->lock()));
      // FE bounding boxes
      auto bbounds       = *element->GetBoundingBoxBounds().get();
      auto boundary_max = std::vector<t_fexpcommon_ct>
      {
        bbounds[CFEXGeomTools::EBoundingBox::eMax]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_x),
        bbounds[CFEXGeomTools::EBoundingBox::eMax]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_y),
        bbounds[CFEXGeomTools::EBoundingBox::eMax]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_z)
      };
      auto boundary_min = std::vector<t_fexpcommon_ct>
      {
        bbounds[CFEXGeomTools::EBoundingBox::eMin]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_x),
        bbounds[CFEXGeomTools::EBoundingBox::eMin]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_y),
        bbounds[CFEXGeomTools::EBoundingBox::eMin]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_z)
      };
      auto found_nodes = search_close_nodes(boundary_min, boundary_max);
      if (found_nodes)
        element->CalcConForce(*found_nodes.get());
      return true;
    }, ESystemElementType::eElement, thread_id); // loop through FEs
}
```

The given source code represents a unified approach to performing the computations, which is common for both sequencing and parallel computing on a single workstation, and for a hybrid parallel computation performed on a computer network as well. In the case of a parallel computation, the loop through the individual model elements is performed with

the help of their mapping to individual threads. It applies for both FE nodes also FEs, respectively. Loop over the individual elements mapped to the specific threads is provided by the following source code:

```
void CFEXPModelContainerBase::IterateModElems(t_IterFunc function,
size_t opt_id, size_t thread_id)
{
  if (!thread_id)
  {
    IterateModElems(function, opt_id);
    return;
  }
  if (_thread_mapper.empty())
    return;
  FEXPCOMMON_FOREACH_ITER(_thread_mapper[thread_id])
  {
    if (!function(GetData(IT->first)))
      return;
  };
}
```

Due to the evolutionary character of a computation, that is, the interdependence of each solution over time to proceed, it is necessary to ensure synchronization of the time between the individual threads. In the case of a computation in a computer network it gets high importance.

The part dealing with general contact detection is important to mention. The algorithm for the range searching in the Euclidean space based on the use of the kd-tree data structure is described in detail in chapter 4. The given functionality is represented by the **class CFEXPTopologieKDTree** in the source files `FEXPTopologieKDTree.h` and `FEXPTopologieKDTree.cpp`, respectively. In the case of a parallel type of the FEXP solver (hybrid, single workstation), such a step is always processed only by the thread that reaches this level as first. This represents the following source code fragment:

```
template<typename TExpCalc, typename TThreadData, typename TThread, typename Tbarrier>
void CFEXPParallelSolver<TExpCalc, TThreadData, TThread, Tbarrier, typename std::enable_if
<
std::is_base_of<ICFEXPExplicitCalcBase    , TExpCalc    >::value &&
std::is_base_of<ICFEXPThreadDataBase      , TThreadData>::value &&
std::is_base_of<ICFEXPThreadBase          , TThread     >::value &&
std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier    >::value
>::type>
::thread_Calculation_function(size_t thread_id, Ptr<TThreadData> data)
{
  try
  {
    bool calculation_end = false; auto calculator = get_calculator(thread_id);
    while (true)
    {
...

      // 5. Step: Update mapping of fe nodes for contact search
      // this require only one thread to do this action
      update_model_mapping(thread_id);

...
    }
  }
  catch (const std::exception & ex)
  {
```

```
...

  }
}
```

where for this purpose, the critical section is used for a data synchronization as suggested the following source code:

```
template<typename TExpCalc, typename TThreadData, typename TThread, typename Tbarrier>
void CFEXPParallelSolver<TExpCalc, TThreadData, TThread, Tbarrier, typename std::enable_if
<
std::is_base_of<ICFEXPExplicitCalcBase    , TExpCalc    >::value &&
std::is_base_of<ICFEXPThreadDataBase      , TThreadData>::value &&
std::is_base_of<ICFEXPThreadBase          , TThread     >::value &&
std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier    >::value
>::type>
::update_model_mapping(size_t thread_id)
{

  // critical section for model mapping update
  std::unique_lock<std::mutex> lock(_mtx_stuct_map_update, std::defer_lock);
  lock.lock();
  ++_thread_counter_stuct_map_update; // increment thread counter
  if (_thread_counter_stuct_map_update == THREAD_COUNT_UPDATER)
    get_calculator(thread_id)->UpdateModelMapping(thread_id);
  if (_thread_counter_stuct_map_update == _threads)
    _thread_counter_stuct_map_update = FEXPCOMMON_DEFAULT_VALUE;
  lock.unlock();
}
```

The final filling the data structure with all the FE nodes is performed at each simulation time step using the following function from a source code:

```
void ICFEXPSolverBase::update_topology_map_for_contact()
{
  _kd_tree_map = CFEXPDataManager<
    CFEXPTopologieKDTree<ICFEXPElementNodeBase, t_fexpcommon_ct>>
      ::SafeAllocInstance(FEXPFEGeom::EFEXPFECoordinates::eC_Count);
  _kd_tree_map->CreateTopologyTree(_builder->GetModelContainer(),
    ESystemElementType::eNode);
}
```

On the kd-tree data structure side, the following function from the source code takes care of filling the data:

```
template<typename TData, typename TValue>
void CFEXPTopologieKDTree<TData, TValue>
::CreateTopologyTree(Ptr<ICFEXPDataModelContIntf> container, size_t opt_id)
{
  container->IterateModElems([this](auto item)
    {
      InsertData(FEXPCOMMON_DYNCAST(ICFEXPModelDataIntf, TData, item));
      return true;
    }, opt_id);
}
```

The final step is to find all the FE nodes that fall into the so-called *bounding box* of the respective finite element to check their possible penetration into the respective shell FE thickness. The nodes of the FE being examined are always filtered out of the resulting node set coming from the *range searching query*. This process is carried out

within the computation of the internal, external and contact forces respectively as suggest the following snippet of a source code:

```
/** @brief It computes forces (internal, external, contact).
*/
void CFEXPCalculation::CalculateForces(size_t thread_id)
{
  // compute external and internal forces
  get_model()->IterateModElems([this](auto item)
    {
...

      // 3. contact forces
      std::map<size_t, Ptr<ICFEXPElementNodeBase>> element_nodes_map;
      auto nodes = element->GetNodes();
      FEXPCOMMON_FOREACH_ITER(nodes)
        element_nodes_map.insert(MAP_PAIR(IT->lock()->GetId(), IT->lock()));
      // FE bounding boxes
      auto bbounds      = *element->GetBoundingBoxBounds().get();
      auto boundary_max = std::vector<t_fexpcommon_ct>
      {
        bbounds[CFEXGeomTools::EBoundingBox::eMax]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_x),
        bbounds[CFEXGeomTools::EBoundingBox::eMax]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_y),
        bbounds[CFEXGeomTools::EBoundingBox::eMax]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_z)
      };
      auto boundary_min = std::vector<t_fexpcommon_ct>
      {
        bbounds[CFEXGeomTools::EBoundingBox::eMin]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_x),
        bbounds[CFEXGeomTools::EBoundingBox::eMin]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_y),
        bbounds[CFEXGeomTools::EBoundingBox::eMin]->
          GetCoordinate(FEXPFEGeom::EFEXPFECoordinates::eC_z)
      };
      auto found_nodes = search_close_nodes(boundary_min, boundary_max);
      if (found_nodes)
        element->CalcConForce(*found_nodes.get());
      return true;
    }, ESystemElementType::eElement, thread_id); // loop through FEs
}
```

It is important to note that the searching process can be performed in parallel for each set of FEs belonging to an appropriate thread. Optimization can go even further, due to the use of parallelization in the kd-tree data structure algorithm. Here is mentioned the possibility of using the GPGPU technology (see [138], [113] or for Octree-based structure see [31]).

Finally, chapter 5 describes how the kd-tree data structure is used for the purpose of the MEIM assembly in terms of the hybrid-parallel FEXP solver described further.

6.4.6 Post Procesing-Output Data

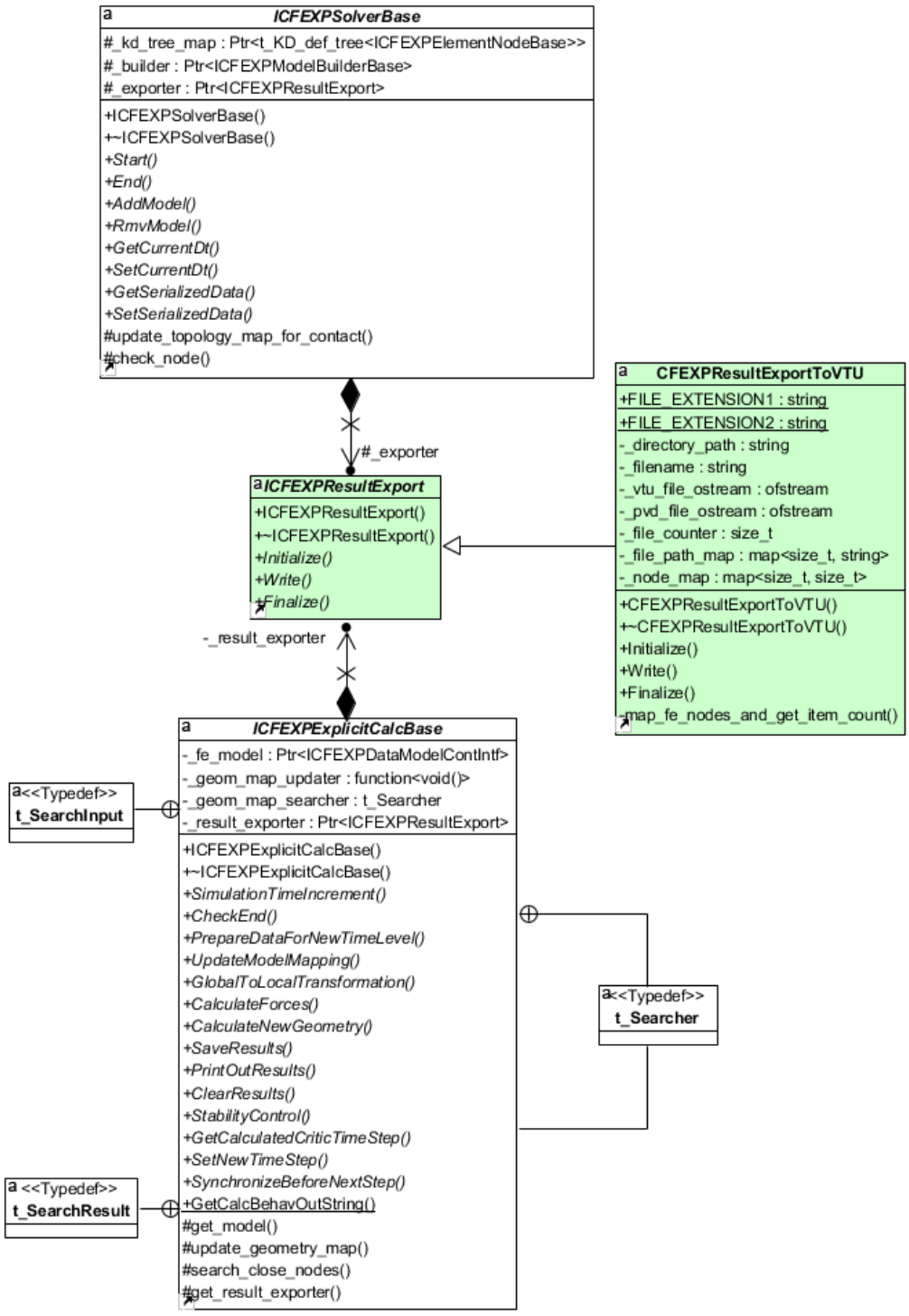


Figure 624: UML diagram of classes for export of results to VTU file for ParaView.

Any analysis does not dispense without the results visualization. In the case of dynamic analysis of a contact/impact mechanics of a flexible bodies this is an absolute necessity as well as in a standard static stress state analysis. The vast majority of a programs dealing with the such a type of numerical analyzes contain their own advanced graphical interface for result handling. Such a user-friendly program is undoubtedly the RFEM program from Dlubal company (see https://www.dlubal.com/en/products/rfem-fea-software/what-is-rfem).

However, the development of such an advanced graphical software tool is a rather complicated and expensive task, so many companies use external configurable graphical software tools developed for such purposes of data preprocessing and post processing. Perhaps the most widely used advanced graphical software tools in this context are GID (https://www.gidhome.com/) and free open-source program SALOME (http://www.salome-platform.org/). Their disadvantage is their complexity given by their generality and possibility of customization. Of course, the disadvantage is related to the short-term usage.

In the context of the testing solver, it was necessary to provide graphical visualization of the resulting data from the numerical simulation in the simplest and most efficient way. For such reasons, the ParaView vizualization free open-source software tool was chosen for the presentation of results from the numerical simulations (https://www.paraview.org/). It is also does advanced graphical visualization tool that supports a large number of file formats and also provides a programmable interface. For simplicity, a VTK (abbreviation of the Visualization ToolKit, it is an open-source, freely available software system for 3D computer graphics, image processing, and visualization) text file format was chosen (see https://www.vtk.org/, https://www.vtk.org/wp-content/uploads/2015/04/file-formats.pdf). Visual presentation of numerical results is also possible in smartphones running on the Google Android OS or iPhone OS environments using the application KiwiViewer (see http://www.kiwiviewer.org/).

The structure of the VTK file format is applied in a form based on the XML format. Appropriate results from the specific time steps of numerical simulation are stored in the files with the extension [*.vtu]. An example of the structure of such a file is shown in List. 6.1.

Listing 6.1: Example of result file content for ParaView vizualization.

```
<!-- TimeStep 1055 Computed Mon Mar 26 08:15:07 2018
 -->
<VTKFile type="UnstructuredGrid" version="0.1" byte_order="LittleEndian">
<UnstructuredGrid>
<Piece NumberOfPoints="10" NumberOfCells = "6">
<Points>
<DataArray type="Float64" NumberOfComponents="3" format="ascii">
0.000000000000000e+00 0.000000000000000e+00 0.000000000000000e+00
0.000000000000000e+00 1.000000000000000e+00 0.000000000000000e+00
1.005321119818887e+00 9.576152776164464e-01 -1.514586728668224e-01
9.998500810750470e-01 -1.932467268182697e-02 -8.318050736105997e-02
2.000000000000000e+00 0.000000000000000e+00 0.000000000000000e+00
1.988359493183459e+00 9.517658223691466e-01 -8.382451735676008e-02
1.795716129044378e+00 6.750442893077370e-01 -4.756455804170117e+00
1.830061054668710e+00 6.880499608200705e-01 -5.005210120675233e+00
1.500000000000000e+00 1.500000000000002e+00 -5.329123199502698e+00
1.800000000000000e+00 1.499999999999999e+00 -5.329123199502698e+00
```

```
</DataArray>
</Points>
<Cells>
<DataArray type="Int32" Name="connectivity" format="ascii">
0 1 2
0 2 3
2 3 5
3 5 4
6 7 9
6 8 9
</DataArray>
<DataArray type="Int32" Name="offsets" format="ascii">
3
6
9
12
15
18
</DataArray>
<DataArray type="UInt8" Name="types" format="ascii">
5
5
5
5
5
5
</DataArray>
</Cells>
<PointData Scalars="" Vectors="FEXP: Displacement u[m]" Tensors="" >
<DataArray type="Float64" Name="FEXP: Displacement u[m]" NumberOfComponents="3"
format="ascii">
0.000000000000000e+00 0.000000000000000e+00 0.000000000000000e+00
0.000000000000000e+00 0.000000000000000e+00 0.000000000000000e+00
5.321119818886758e-03 -4.238472238355365e-02 -1.514586728668224e-01
-1.499189249529957e-04 -1.932467268182697e-02 -8.318050736105997e-02
0.000000000000000e+00 0.000000000000000e+00 0.000000000000000e+00
-1.164050681654076e-02 -4.823417763085340e-02 -8.382451735676008e-02
2.957161290443777e-01 1.750442893077369e-01 -5.256455804170117e+00
3.006105466871013e-02 1.880499608200706e-01 -5.505210120675233e+00
-5.342708603934427e-16 2.064401470719396e-15 -5.829123199502698e+00
-7.988911667531997e-17 -1.256564195232448e-15 -5.829123199502698e+00
</DataArray>
</PointData>
<CellData Scalars="" Vectors="" Tensors="">
</CellData>
</Piece>
</UnstructuredGrid>
</VTKFile>
```

The sequence of individual result files corresponding to the individual simulation time steps is then written into the next text file with the extension [*.pvd], whose fragment presents the List. 6.2.

Listing 6.2: Example of the list of resulting simulation files for ParaView vizualization.

```
<?xml version="1.0"?>
<VTKFile type="Collection" version="0.1">
<Collection>

...

<DataSet timestep="211" group = "" part = "" file = "result.out.m0.1055.vtu"/>

...
```

```
</Collection>
</VTKFile>
```

An example of a ParaView main frame window with a loaded model is presented in the Fig. 625.

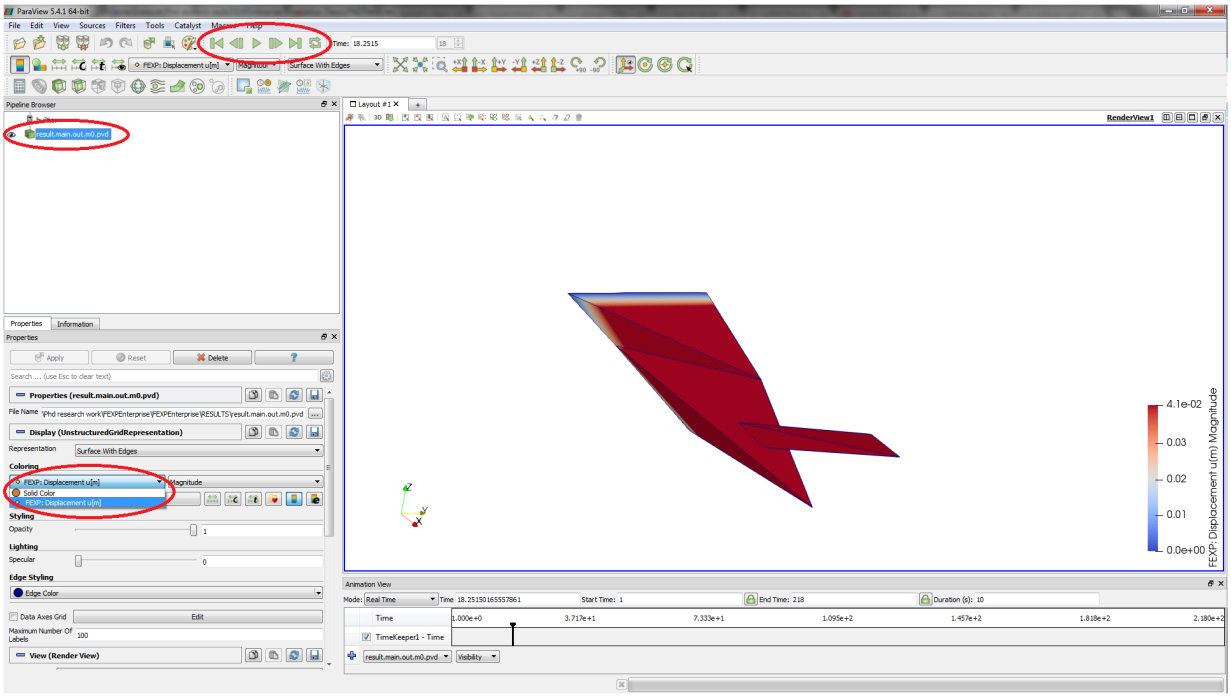


Figure 625: ParaView main frame window presenting the loaded result data from the numerical simulation.

The export of numerical simulation results to the VTK file format ensures the **class CFEXPResultExportToVTU** implementing the interface represented by the abstract **class ICFEXPResultExport** contained in the source files `FEXPResultExport.h` and `FEXPResultExport.cpp`. The export of results is in the diction of an appropriate type of FEXP solver. This represents the following source code fragment:

```
template<typename TExpCalc, typename TThreadData, typename TThread, typename Tbarrier>
void CFEXPParallelSolver<TExpCalc, TThreadData, TThread, Tbarrier, typename std::enable_if
<
std::is_base_of<ICFEXPExplicitCalcBase    , TExpCalc    >::value &&
std::is_base_of<ICFEXPThreadDataBase      , TThreadData>::value &&
std::is_base_of<ICFEXPThreadBase          , TThread     >::value &&
std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier    >::value>::type
>
::thread_Calculation_function(size_t thread_id, Ptr<TThreadData> data)
{
  try
  {
    bool calculation_end = false; auto calculator = get_calculator(thread_id);
    while (true)
    {
...
```

```
      // 9.1. Step: Export results
      try_export_calculation_results(data, thread_id);
      // 9.2. Step: Print out results
      calculator->PrintOutResults(thread_id);

...
    }
  }
  catch (const std::exception & ex)
  {

...

  }
}
```

The given fragment of a source code shows both the attempt to export the results to the [*.vtu] text file and the attempt to send data about the behaviour of computational process. The frequency of printing out the results of a numerical simulation into an appropriate file determines the settings of the FEXP solver contained in the input data file as described in chapter 6.4.2. The export of results is always provided by one thread only, namely by the thread that reached the given block first. The following source code represents such an effort:

```
template<typename TExpCalc, typename TThreadData, typename TThread, typename Tbarrier>
void CFEXPParallelSolver<TExpCalc, TThreadData, TThread, Tbarrier, typename std::enable_if
<
std::is_base_of<ICFEXPExplicitCalcBase    , TExpCalc    >::value &&
std::is_base_of<ICFEXPThreadDataBase      , TThreadData>::value &&
std::is_base_of<ICFEXPThreadBase          , TThread     >::value &&
std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier    >::value
>::type>
::try_export_calculation_results(Ptr<TThreadData> data, size_t thread_id)
{
  // critical section for export of results from numerical simulation
  std::unique_lock<std::mutex> lock(_mtx_save_res_synchr, std::defer_lock);
  lock.lock();
  ++_thread_counter_save_res_synchr; // increment thread counter
  if (_thread_counter_save_res_synchr == THREAD_COUNT_UPDATER)
  {
    auto calculator = get_calculator(thread_id);
    _result_saved.store(calculator->SaveResults(FEXPCOMMON_DEFAULT_VALUE));
    // clear all results --> data space optimization
    calculator->ClearResults(FEXPCOMMON_DEFAULT_VALUE);
  }
  if (_thread_counter_save_res_synchr == _threads)
    _thread_counter_save_res_synchr = FEXPCOMMON_DEFAULT_VALUE;
  lock.unlock();
}
```

Whether to save or not to save the results of a numerical simulation controls the class managing the individual operations of an explicit computation described in chapter 6.4.5. It represents the following part of the source code:

```
/** @brief It saves the resulting simulation data for the current simulation time.
*/
bool CFEXPCalculation::TrySaveResults(size_t thread_id, bool forced/*= false*/)
{
  auto is_print = forced; auto model = get_model();
  // check simulation end
```

```
  if (!is_print)
  {
    model->IterateModElems([this, &is_print](auto item)
      {
        if (!FEXPCOMMON_STACAST(ICFEXPModelDataIntf, ICFEXPElementNodeBase, item)->
          GetIsTimeToSaveRes())
        {
          is_print = false;
          return is_print;
        }
        else
          is_print = true;
        // to check all nodes --> return true
        return false;
      }, ESystemElementType::eNode, thread_id); // loop through FE nodes
  }
  // save results if need, not for hybrid-parallel solver (--> network computation)
  if(is_print)
    save_results_and_export(thread_id, model);
  return is_print;
}
```

The collection of appropriate data for the saving of results is then performed in a manner that represents the following source code listing:

```
void CFEXPCalculation
::save_results_and_export(size_t thread_id, Ptr<ICFEXPDataModelContIntf> reslt_cont)
{
  // 1. save node results and check time step consistency
  auto current_time_counter = size_t(FEXPCOMMON_DEFAULT_VALUE); auto model = get_model();
  model->IterateModElems([this, reslt_cont, &current_time_counter](auto item)
    {
      auto time = save_node_result(FEXPCOMMON_STACAST(
        ICFEXPModelDataIntf, ICFEXPElementNodeBase, item), reslt_cont);
      if (current_time_counter == FEXPCOMMON_DEFAULT_VALUE)
        current_time_counter = time;
      if (current_time_counter != time)
        FEXPCOMMON_EXCEPTION("Error: Time step inconsistency!!!");
      return true;
    }, ESystemElementType::eNode    , thread_id); // loop through FE nodes

  // 2. save element connectivity
  model->IterateModElems([this, reslt_cont](auto item)
    {
      save_elem_connct(
        FEXPCOMMON_STACAST(ICFEXPModelDataIntf, ICFEXPElementBase, item), reslt_cont);
      return true;
    }, ESystemElementType::eElement, thread_id); // loop through FEs
  // 3. call exporter to save results
  auto exporter = get_result_exporter();
  if (!exporter)
    return;
  exporter->Write(FEXPCOMMON_DYNCAST(ICFEXPModelDataIntf, CFEXPCalculationModelNodeResult,
    reslt_cont->GetModelElement(current_time_counter, ESystemElementType::eResult)));
}
```

The final result data storage is provided by a particular export class. Here it is represented by the **class CFEXPResultExportToVTU**. It ensures the correct data formatting into the resulting text file [*.vtu] and also ensures additional file management. Keep in mind that the printing of the file is a blocking operation performed here sequentially. Often it is performed through the asynchronous type of operation. The following source code fragment shows the part of the result data formatting to [*.vtu] file:

```
void CFEXPResultExportToVTU::Write(Ptr<CFEXPCalculationModelNodeResult> result)
{
  if (!result)
    return;
  // current time of result write
  auto cnt_tm = std::time(nullptr);
  auto dttime = std::string(std::ctime(&cnt_tm));
  // increment file number
  ++_file_counter;

  // add new file to map
  auto filename_out = _filename + ".out.m0." +
    CFEXPBaseConvers::NumberToString(result->GetId()) + "." + FILE_EXTENSION1;
  auto filename_pth = _directory_path + filename_out;
  _file_path_map.insert(MAP_PAIR(_file_counter, filename_out));
  // create file
  _vtu_file_ostream.open(filename_pth);
  if (!std::experimental::filesystem::exists(filename_pth))
    FEXPCOMMON_EXCEPTION("Error: File path for export of results is incorrect!");

  // start write to file

...

  // 1. --->
  // write node coordinates
  _vtu_file_ostream
  <<
    "<DataArray type=\"Float64\" NumberOfComponents=\"3\" format=\"ascii\">"
  << std::endl;
  FEXPCOMMON_FOREACH_ITER_FNC(result->GetNodeResults(),
    {
      _vtu_file_ostream << CFEXPBaseConvers::NumberToString(IT->_cx) + FEXPCOMMON_EMCHR_STRING
                        + CFEXPBaseConvers::NumberToString(IT->_cy) + FEXPCOMMON_EMCHR_STRING
                        + CFEXPBaseConvers::NumberToString(IT->_cz) << std::endl;
    });
  _vtu_file_ostream << "</DataArray>" << std::endl;
  _vtu_file_ostream << "</Points>" << std::endl;
  _vtu_file_ostream << "<Cells>" << std::endl;

...

  // close file
  _vtu_file_ostream.close();
}
```

The instancioning of the class for export of results is provided to the FEXP solver from the outside where its explicit initialization and finalization is also performed. It is represented by the following source code fragment:

```
/** @brief Main function (single workstation).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{

...

  // create result exporter for Paraview and initialize it
  Ptr<ICFEXPResultExport> result_exporter = CFEXPDataManager<CFEXPResultExportToVTU>
    ::SafeAllocInstance(solver_config_setting->GetResultDirPath(),
      solver_config_setting->GetResultName());
  result_exporter->Initialize();
  try
  {
```

```
    switch (setting->GetSolver())
    {
    case CFEXPBaseSetting::EFEXPSolverType::eSequential : // sequential type of solver
      CFEXPDataManager<CFEXPSequentialSolver<CFEXPCalculation>>
        ::SafeAllocInstance(builder, result_exporter)->Start();
      break;
    case CFEXPBaseSetting::EFEXPSolverType::eParallelCpu: // parallel type of solver
      CFEXPDataManager<SIMPLE_PARALLEL_SOLVER>
        ::SafeAllocInstance(builder, result_exporter)->Start();
      break;

...

    }
  }
  catch (const std::exception & ex)
  {

...

  }
  // finalize actins of exporter
  result_exporter->Finalize();

...
}
```

In the export finalization phase, a list of names of formatted [*.vtu] files is collected to the [*.pvd] file, which is then used as an input to the ParaView tool to visualize the results of numerical simulation. It provides the folowing source code:

```
void CFEXPResultExportToVTU::Finalize()
{
  if (_file_path_map.empty())
    return;
  _pvd_file_ostream.open(_directory_path + _filename + ".main.out.m0." + FILE_EXTENSION2);
  _pvd_file_ostream << "<?xml version=\"1.0\"?>" << std::endl;
  _pvd_file_ostream << "<VTKFile type=\"Collection\" version=\"0.1\">" << std::endl;
  _pvd_file_ostream << "<Collection>\n";

  // write out all files with time results
  size_t idx;
  FEXPCOMMON_FOREACH(FEXPCOMMON_DEFAULT_VALUE + 1, _file_path_map.size(), idx)
    _pvd_file_ostream << "<DataSet timestep=\""
      + CFEXPBaseConvers::NumberToString(idx)
      + "\" group=\"\" part=\"\" file=\""
      + _file_path_map[idx]
      + "\"/>" << std::endl;

  _pvd_file_ostream << "</Collection>" << std::endl;
  _pvd_file_ostream << "</VTKFile>" << std::endl;
  _pvd_file_ostream.close();
  // reset file counter
  _file_counter = FEXPCOMMON_DEFAULT_VALUE;
}
```

Deliberately, the solution for the network form of the FEXP solver is not dealt with, mainly due to its many specific features. Not only these specifics but also many others are described in more detail in chapter 6.4.8. The part concerning the export of partial results at runtime during the computation is discussed in more detail in chapter 6.4.7 dealing with the FEXP Solver Manager application.

6.4.7 FEXP Solver Manager

All complex equipment should be delivered with an interface that makes working with it easier. In terms of software equipment, such an approach mainly applies to attitudes towards customers of commercially-produced software systems where a user-friendly working environment of the given product is one of the attributes of success represented by larger product expansion. Configuration, control, and also monitoring of a given software equipment often require asynchronous processing of various data, especially with respect to time demanding operations. Appropriate commands are often provided through the use a graphical user interface (GUI) that must remain responsive even during more challenging operations. Such an approach is critical in the case of enterprise software systems (PDM, ERP, etc.) connecting a number of users through the computer network, where a more time-consuming response happens caused by mutual communications over the computer network. Due to these reasons, a software tool providing easier handling with the FEXP solver was designed and programmed. This is an application that communicates with the user through the GUI.

There are a number of libraries providing the possibility to design and program the GUI under the Windows OS. The most well-known libraries for GUI programming in native C++ programming language include Qt (originally provided and developed by Nokia company for the Symbian OS programming, see https://www.qt.io/) and MFC (abbreviation of the Microsoft Foundation Class).

Due to the complexity and complicated character of the C++ programming language, the development of simpler programming languages has risen since the turn of the millenium. They are now able to support not only the possibilities of object oriented programming (OOP), but they also support a modern functional programming approach compared to the usual imperative way. Programming languages such as Microsoft .NET/C# or Scala (built on Java programming language) currently belong among the modern object-oriented programming languages supporting the functional paradigm of programming. With respect to the Windows OS target platform and the use of a variety of its development tools, the Microsoft .NET/C# programming language was chosen with the libraries .NET WinForms for the GUI designing and programming. For the programming of a large number of functionalities, the modern object-oriented and functional approach were applied with maximal utilization of asynchronous type programming.

Asynchronous programming under the Windows OS is already supported in the Win32 API library using the so-called callback function. It is invoked after the end of the given operation. Due to such an approach, the given asynchronous operation does not block the invoking thread, which can continue to launch the following operations after such a call. An asynchronous execution of operations in this fashion originally has taken over the C# programming language since .NET Framework 3.5. The definition of such an asynchronous operation required two specially designed functions that defined one asynchronous function. This approach was mainly adopted by .NET/WCF technology for programming service-oriented applications. With .NET Framework 4.0, a significant improvement in thread creation by so-called tasks representing a non-blocking operation running in a new

thread. However the disadvantage was dealing with the complicated callback function, which had to be properly treated for the correct behaviour of the program. It can be said that a revolutionary event was the arrival of the .NET Framework 4.5, which introduced new constructions of the C# programming language, which significantly simplified asynchronous programming (see [12]). This applies to the new pattern the *async - await*. The modern asynchronous approach represented by the last asynchronous constructions of the C# programming language is widely used in the FEXP Solver Manager for managing a large number of originally blocking operations.

Editing Table Data

The table data format introduced in chapter 6.4.2 can be read and edited using the FEXP Solver Manager. For testing purposes of the FEXP solver, it has been included into the application for easier editation and validity check of model input data. The feature is located in the tab named *"Input Files"* of the main application window. Here, using the button named *"Add Input File"*, text files containing the spreadsheet data can be loaded. It is illustratively shown in the Fig. 626.

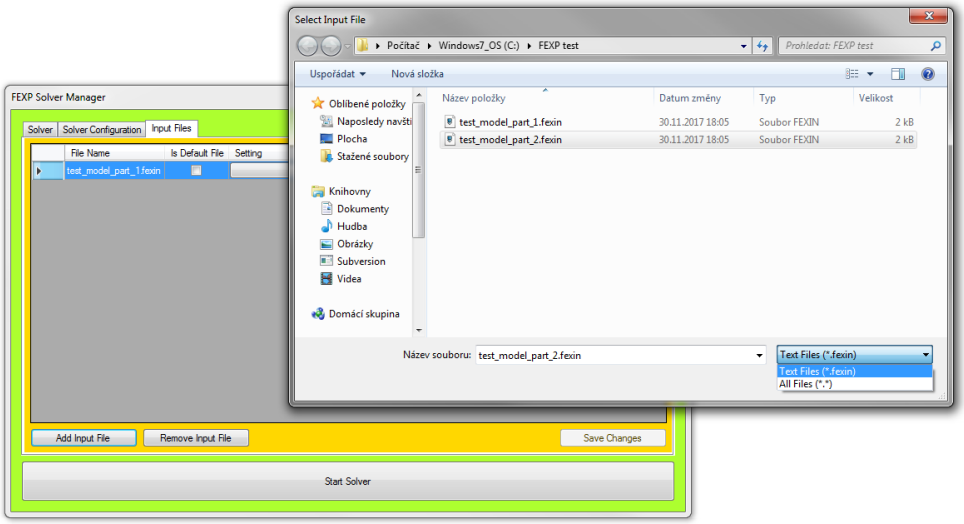


Figure 626: The listing of a model input data files.

Adding files to an application worksheet is provided by the following functions from the source code representing the event for button click

```
/// <summary>
/// Event is invoked for adding new row into (input file) the respective spreadsheet.
/// </summary>
private void OnAddInputFileClickEvent(object sender, EventArgs e)
{
  // set up columns in input data file grid
  var paths = FEXPCommonFunctions.GetFileToOpen("Select Input File", "fexin", true);
```

```
  if (!paths.AnyEx())
    return;
  if (InFileGridController == null)
  {
    InFileGridController = new FEXPDataGridController<InFileCellData>(
      _input_files_grid, DataGridViewSelectionMode.FullRowSelect,
      InFileGridCols.Select(pcol => Tuple.Create(pcol.Key.GetDescription(),
          pcol.Value.Item1)));
    InFileGridController.OnCellClickEvent += OnInFileGridOnCellClickEvent;
  }

  var progress = new FEXPProgress2Dlg();
  progress.Text = "Loading items to spreadsheet";
  progress.NameOfProgressAction = "Loading";
  progress.LongLastingActionAsync = async (indicator) =>
    {
      // compose row data for spreadsheet
      var rows = new List<GridRowData<InFileCellData>>();
      paths.ForEach(path =>
        {
          if (_input_data_change_map.ContainsKey(path))
            return;
          var row = new GridRowData<InFileCellData>();
          var dta = new InFileCellData(path);
          InFileGridCols.ForEach(pcol => row.Add(pcol.Key.GetDescription(), dta));
          rows.Add(row);
         _input_data_change_map[path] = false;
        });
      // load rows into the spreadsheet
      await InFileGridController.LoadDataAsync(rows, indicator).ConfigureAwait(false);
    };
  progress.ShowDialog();
}
```

For each listed file in spreadsheet it is possible to show its structure consisting of blocks and corresponding tables as shown in the Fig. 627. The display of the respective table for editing in the spreadsheet is invoked by double-clicking on the appropriate table from an input file block in the *Input File Data Structure* dialog.

Spreadsheets are designed to be capable of arbitrarily editing the data. The resulting changes can be saved to the appropriate file. For the given purpose, a spreadsheet control controller has been designed to use asynchronous operations, especially for a data loading operation. It applies primarily to a big model data load. Data loading into the spreadsheet represents listing of a source code related to the spreadsheet controller

```
public async Task LoadDataAsync(IEnumerable<GridRowData<TData>> data, IProgress<int>
    progress)
{
  if (data == null || !data.Any())
    return;
  if (Grid.Columns.Count == 0)
```

```
  SetUpGridColumns(GridColumns.ToList());
var rows = data.ToArray();
await Task.Run(() =>
  {
    var step = FEXPProgress2Dlg.GetProgressStep(rows.Count()); var counter = 0;
    for (int indx = 0; indx < rows.Length; ++indx)
    {
      // update progress
      if ((++counter % step.Item2) == 0)
        progress?.Report((int)(counter / step.Item1));

      var temp_row = new DataGridViewRow();
      var row_item = rows[indx];
      GridColumns.ForEach(col =>
        {
          var value = row_item.ContainsKey(col.Item1) ? row_item[col.Item1] : null;
          var cell_data = new GridCellDataWrapper<TData>(col.Item1, value);
          DataGridViewCell cell;
          switch(col.Item2)
          {
            case ECellEditor.eButton:
              cell = new DataGridViewButtonCell();
              break;
            case ECellEditor.eCheckBox:
              cell = new DataGridViewCheckBoxCell();
              break;
            default:
              cell = new DataGridViewTextBoxCell();
              break;
          }
          // associate value
          cell.Value = cell_data.ToString();
          cell.Tag = cell_data;
          cell.ToolTipText = value.ToolTip;
          // add cell to row
          temp_row.Cells.Add(cell);
          cell.ReadOnly = !value.IsEditable;
        });
      // add row
      Grid.MyInvoke(() => Grid.Rows.Add(temp_row));
      temp_row.Tag = row_item;
    }

    // time consumption operation --> update cells
    var progress1 = new FEXPProgress1Dlg();
    progress1.Text = "Spreadsheet update";
    progress1.NameOfProgressAction = "Updating ...";
    // auto resize columns based on the spreadsheet content
    progress1.LongLastingActionAsync = async () =>
    await Task.Run(() => Grid.MyInvoke(() =>
        Grid.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells)));
```

```
      progress1.ShowDialog();
    }).ConfigureAwait(false);
}
```

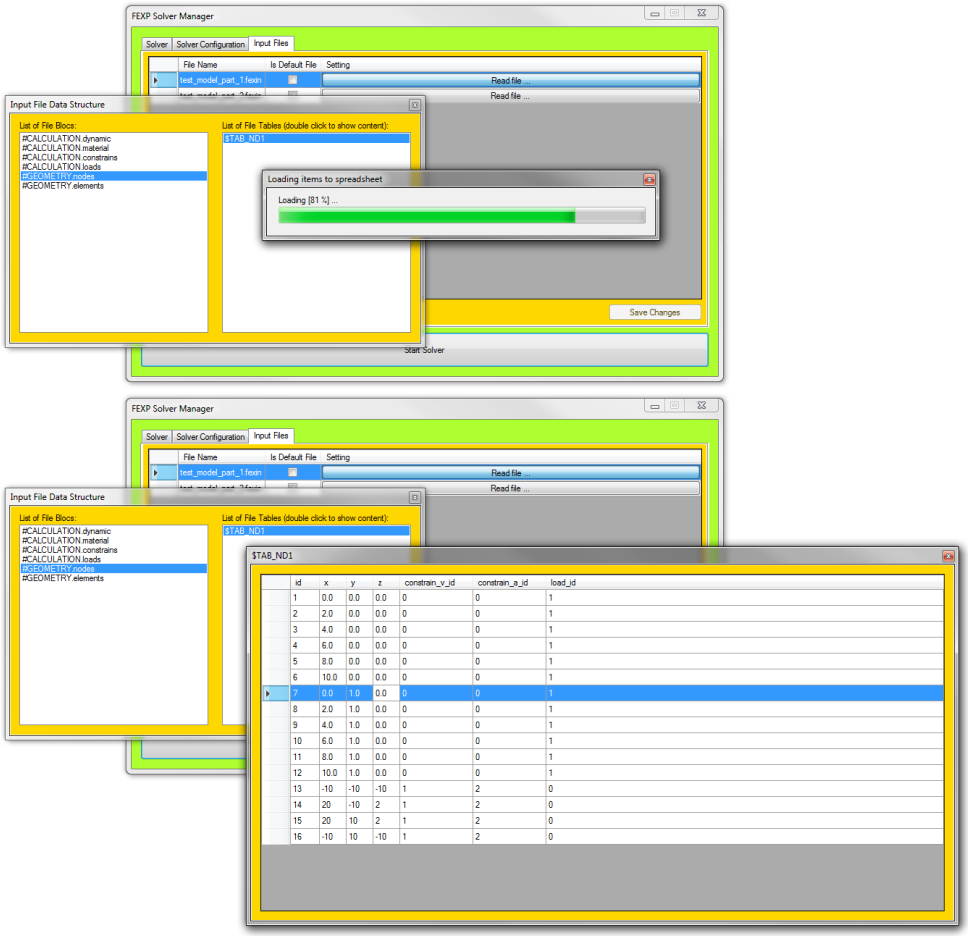


Figure 627: Loaded input file structure.

The FEXP Solver Setting Assembly

A configuration setting file is required to start up the FEXP solver with the specific command line arguments. These arguments depend on the type of the used FEXP solver, and whether the solver is started up and monitored by the FEXP Solver Manager application.

In order to create a solver configuration file, the type of FEXP solver must be selected in the tab *"Solver"*. If the target is the network form of FEXP solver, it is necessary to check-on the checkbox named *"Network Solver"* and fill in the data concerning the IP address and the port of the server where it will listen for a connection of the client's workstation to start the solution process. The steps to create the configuration file are basically arbitrary, except the step where it is necessary to choose the type of FEXP

solver. Based on such an option, another tab has been added in the main window of FEXP Solver Manager which refers to the assignment of individual workstations required to start the process of numerical computation on the computer network. The corresponding text editation fields in the tab *"Solver"* are a pre-filled by IP address of localhost and the port number corresponding to the randomly selected number in the allowed range is shown in the Fig. 628.

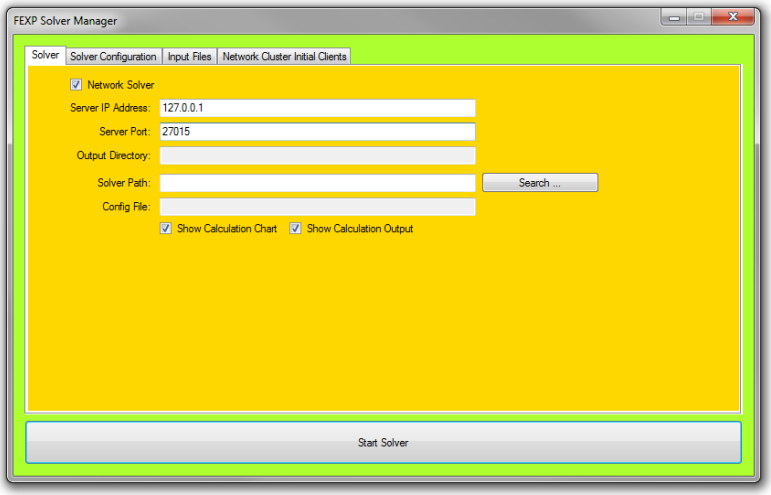


Figure 628: Setting of the FEXP solver type, and setting of the solver behaviour monitoring.

The next step in the configuration process is the selection of files containing model data with the selection of a file so-called the default (network solution) as shown in the Fig. 629.

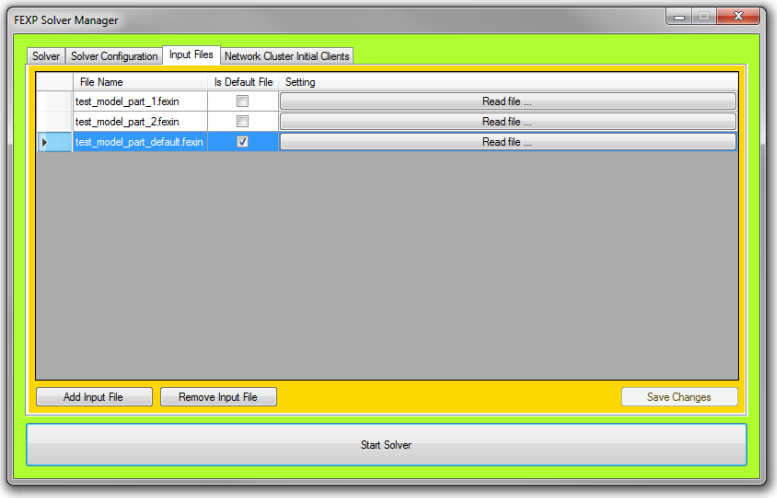


Figure 629: Listing a model input files.

If the network form of the FEXP solver has been selected, it is necessary to list the required number of connected workstations required to start up the numerical simulation. Each workstation is identified by its IP address as shown in the Fig. 630.

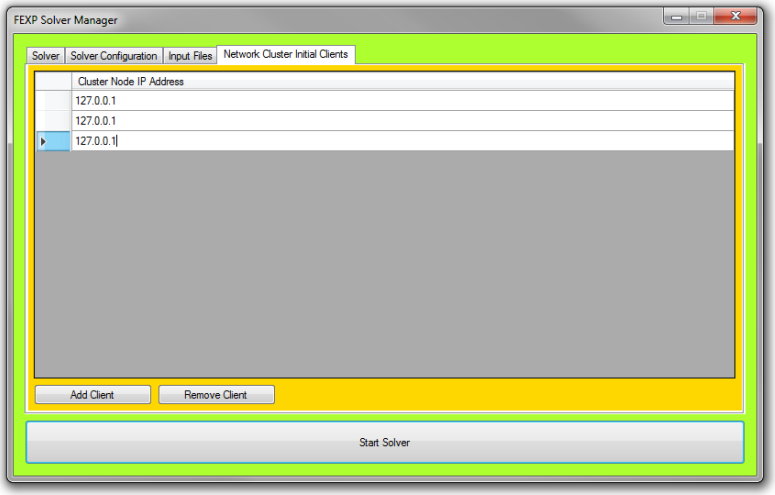


Figure 630: Listing of the network initialization workstations.

The FEXP Solver Process Control

With respect to the control of long-lasting process behaviour of explicit numerical computations, and also due to the control over the computational details, an appropriate monitor was designed.

The behaviour of designed FEXP solver monitor was inspired by the same monitor used in the RFEM program, but in a simpler form. The respective FEXP solver monitor is based on hooking the events fired by flushing the data buffer of the standard output stream at runtime. This is usually one of the simplest ways used for inter-process communication. The particular functionality of the FEXP solver monitor is implemented in the **class FEXPSolverMonitorDlg** from the source code file `FEXPSolverMonitorDlg.cs`. Here the FEXP solver as an external native application is started up with the appropriate settings enabling the previously mentioned monitoring.

For the successful start of the process, the full path to the executable file [*.exe] of the appropriate FEXP solver must be known before the process start up. The path to the executable file must be typed in the appropriate text edit field in the FEXP Solver Manager tab named *"Solver"*. Together with the specified path of the executable file, it is also necessary to provide additional settings for the monitor and thus also the FEXP solver itself. The FEXP solver acquires the given setting through the configuration file and the command line arguments discussed before. The necessary settings are shown in the Fig. 628. Creation of the FEXP solver process then presents the following part of the source code

```
/// <summary>
/// It creates instance of the FEXP solver process.
/// </summary>
/// <returns>Process</returns>
private Process CreateProcess()
{
  var process = new Process();
  process.StartInfo.FileName = SolverSetting.SolverExecutablePath;
  var input = new StringBuilder();
  if(string.IsNullOrEmpty(SolverSetting.ServerIP) ||
     string.IsNullOrEmpty(SolverSetting.ServerPort))
    input.Append("1");
  else
    input.Append("1"
     + " " + SolverSetting.ServerIP
     + " " + SolverSetting.ServerPort);
  process.StartInfo.Arguments = input.ToString();
  process.StartInfo.Verb = "runas";
  process.StartInfo.UseShellExecute = false;
  process.StartInfo.RedirectStandardOutput = true;
  process.StartInfo.CreateNoWindow = true;
  process.OutputDataReceived += OnProcessOutputDataReceived;
  return process;
}

...

/// <summary>
/// Event is invoked as reaction to flushing the std output buffer by
/// the FEXP solver process.
/// </summary>
private void OnProcessOutputDataReceived(object sender, DataReceivedEventArgs e)
{
  // checking cancel request --> stop the process
  if (_token.IsCancellationRequested)
  {
    StopProcess();
    return;
  }
  if (string.IsNullOrEmpty(e.Data))
    return;
  // line number to each line of the output.
  _progress?.Report(_lineCount++);
  // try print data
  PrintRealTimeValues(e.Data, e.Data.Split(LINE_DELIMITER));
}
```

Running the process itself by using the mouse to click the button named *"Start Solver"* performs the following functions from the source code

```
/// <summary>
```

```
/// It starts the FEXP solver process and waits for its end.
/// </summary>
private async Task StartProcessAsync()
{
  await Task.Run(() =>
    {
      _process.Start();
      _processName = _process.ProcessName;
      _process.BeginOutputReadLine();
      _process.WaitForExit();
      _process.Close();
      _process.Dispose();
    }).ConfigureAwait(false); SetChartToDefault();
}
```

In the case of the selection of the option managing the computational process behaviour of the selected node, the required data are sent from the FEXP solver to the standard output stream. Such data are subsequently parsed by the FEXP Solver Manager and conveniently displayed in the appropriate dialog box. The mapping of the FEXP solver result behaviour from the console window (standard output stream) to the output in the FEXP Solver Manager is shown in the Fig. 631.

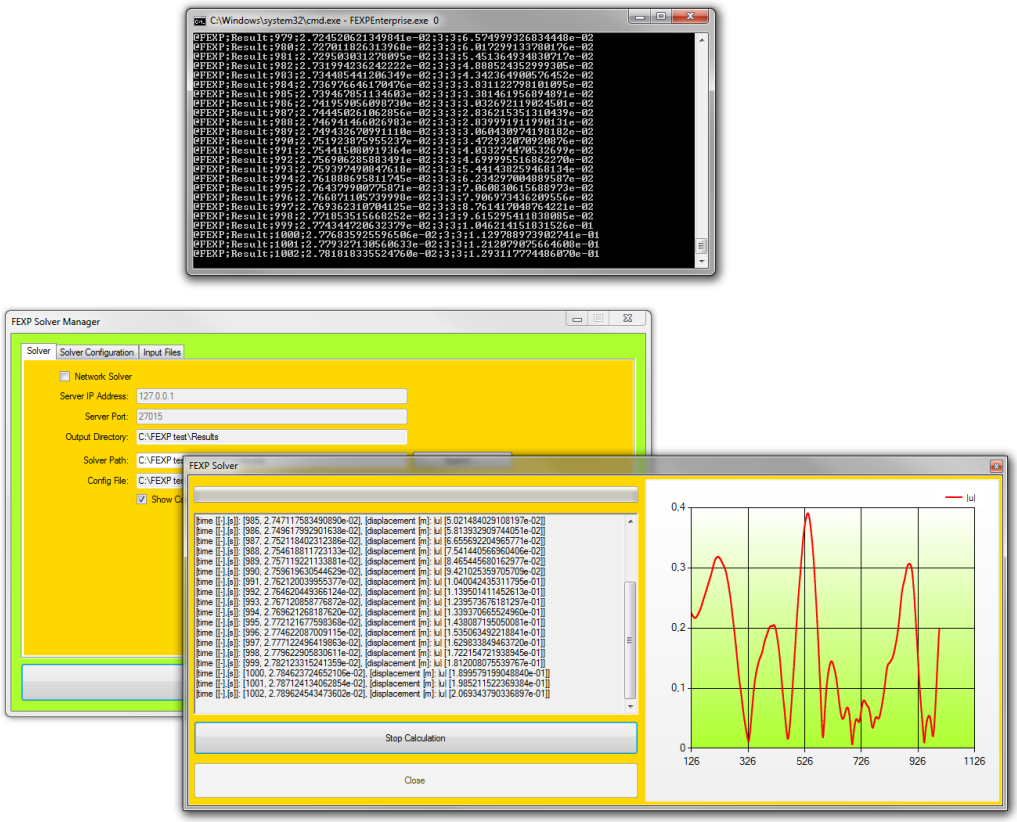


Figure 631: Result behaviour in a console window vs. FEXP Solver Manager.

The printing out of the results on the solver side manages the function represented by the following part of the C++ source code

```
/** @brief It prints out appropriate result data from numerical simulation to console window.
*/
void CFEXPCalculation::PrintOutResults(size_t thread_id)
{
  // setting from the solver config file
  auto config_setting = FEXPCOMMON_DYNCAST(ICFEXPModelDataIntf, CFEXPSolverConfigSetting,
    get_model()->GetModelElement(
      ICFEXPSetting::ESettingType::eSolver, ESystemElementType::eSetting));
  if (!config_setting->ShowCalcOutput())
    return;
  // format text to print out
  auto text = GetCalcBehavOutString(FEXPCOMMON_STACAST(
    ICFEXPModelDataIntf, ICFEXPElementNodeBase,
      get_model()->GetModelElement(config_setting->NodeIdToShow()
       ESystemElementType::eNode, thread_id)), config_setting->NodeDofToShow());
  if (text == FEXPCOMMON_EMPTY_STRING)
    return;
  CFEXPLog::WriteLine(text);
}
```

The output in the FEXP Solver Manager dialog box is then provided by the following C# code

```
/// <summary>
/// It prints out the data at runtime.
/// </summary>
private int PrintRealTimeValues(string data, string[] items)
{
  var tuple = GetValueForChart(items);
  // invoking printing of solver output to text area
  if (ShowCalcOutput || tuple.Item1 < I_DEFAULT_VALUE)
  {
    if(tuple.Item1 <= I_DEFAULT_VALUE)
    {
      if(data?.StartsWith("@") == true)
         _console_manager.WriteLine(items[items.Length - 1]);
    }
    else if(ShowCalcOutput)
    {
      if (!string.IsNullOrEmpty(tuple.Item3))
        _console_manager.WriteLine(tuple.Item3);
    }
  }
  // invoking print to chart control
  if (ShowRealTimeChart && tuple.Item1 >= I_DEFAULT_VALUE)
    _realTimeChart.MyInvoke(() => AddDataToChart(tuple.Item1, tuple.Item2));
  return tuple.Item1;
}
```

The content of the previous C# code statements also includes a feature that provides a graphical presentation of the progress of the monitored FE node variable at runtime. The chart is optionally displayed in the FEXP Solver Manager dialog box (see Fig. 631). The

presentation of the chart is available through the option placed in the main window of the FEXP Solver Manager (see Fig. 629).

The computational process can be arbitrarily stopped at any time using the *"Stop Calculation"* button (see Fig. 631) and subsequent execution of the related event reprezented by the following source code

```
/// <summary>
/// Event is invoked as reaction to the user action to stop
/// the running FEXP solver process.
/// </summary>
private void OnStopCalcClickEvent(object sender, EventArgs e)
{
  if (_token == null)
    return;
  var diagres = MessageBox.Show(this, "Do you want to kill the process?", "Stop
      Calculation", MessageBoxButtons.OKCancel, MessageBoxIcon.Question);
  if (diagres == DialogResult.Cancel)
    return;
  if (!IsProcessExits())
    return;
  if (_processCallTimeSpan.Seconds < _processMaxResponseSeconds)
    _token.Cancel();
  else
    StopProcess();

  _isKill = true;
  _buttonStopCalc.Enabled = false;
  _buttonCloseWindow.Enabled = false;
}
```

6.4.8 Client-Server based Network Distributed Computation

As mentioned earlier, this section deals with a purely hybrid-parallel (network) form of the FEXP solver (see Fig. 622). The solver considers a computer's interconnection within a LAN or VPN based on a client-server architecture through the TCP/IP communication protocol of transport layer. The communication of the central server workstation with a connected client workstation is illustratively shown in the Fig. 632. In the case of a FEXP solver, the server takes care of secure connection and disconnection of the individual client workstations, it synchronizes the time during numerical computation, and it also takes care of right data exchanging between the individual workstations and exporting the model simulation results.

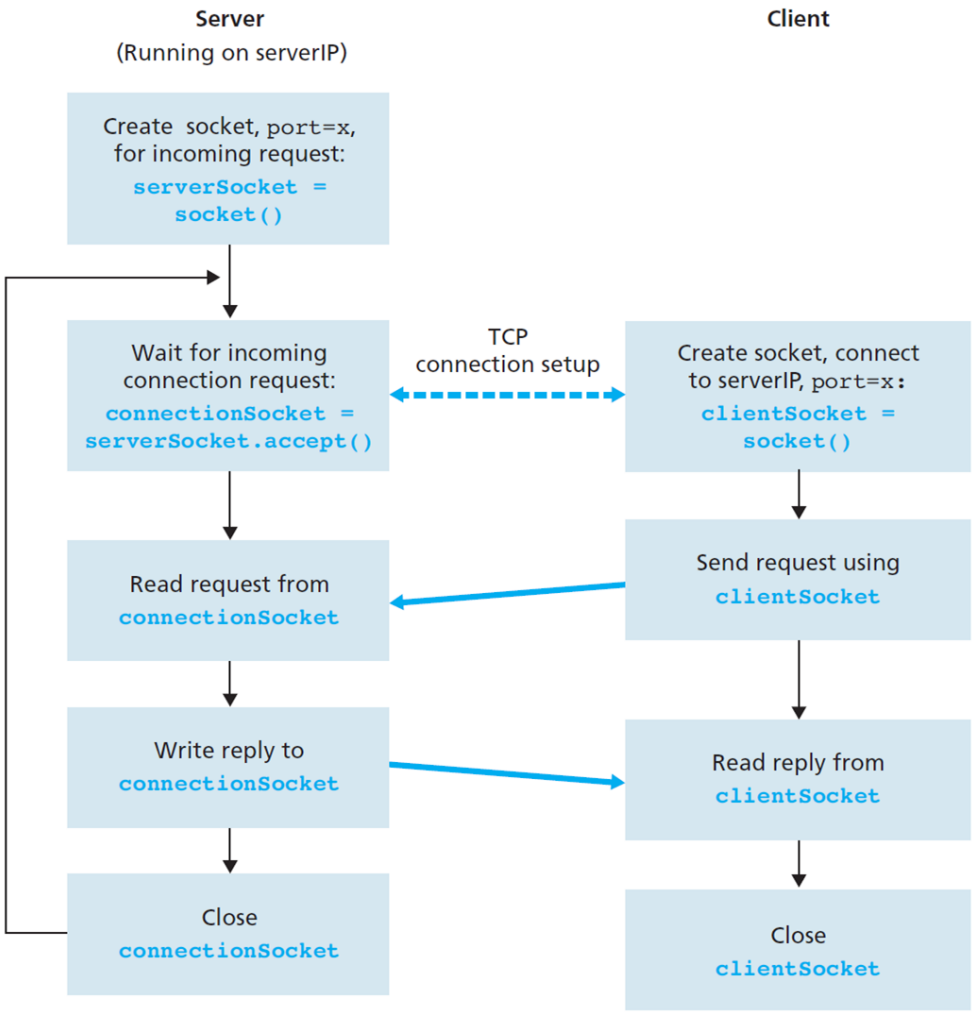


Figure 632: Client-server architecture using TCP/IP network.

In terms of object oriented design, such a type of FEXP solver is an extension of a parallel type of FEXP solver focused only on an individual workstation as shown in the

Fig. 622. As opposed to the locally parallel running computation, the hybrid type of FEXP solver includes a functionality that takes care of network load balance if possible. It depends on the analysis of the MEIM, which is composed based on the current state of the numerically simulated dynamic model at runtime, as described in chapter 5.

A large number of chaotically interacting structures spatially defined by a dense meshes of finite elements are those macro entities that can move freely on a computer network. The data distribution itself is based on the current contact state of a spatial entities. Each step of the explicit time integration of equations of motion requires assembly of MEIM. The MEIM must be analyzed for the subsequent network computing balance and solution of micro details relating to the integration of internal forces.

While designing this part of the server, the possibility of the application the FETI method was taken into account for the next FEXP solver development. The approach developed for the presented purpose of explicit numerical simulations of a structural dynamic phenomenas then appears to be as the next alternative of application of existing solutions applied in terms of the FETI method exclusively based on the MPI technology.

Due to the applied management algorithm for each connection, it is necessary to have a server located on a powerful workstation. This primarily applies to larger computer networks dealing with challenging and extensive models. The given server workstation performance requirements are otherwise common for all distributed software applications based on the dealt with network architecture containing the central server. Otherwise, insufficient server performance would have a significant impact on the whole process of numerical solution.

In order to run the numerical simulation, it is necessary to ensure the correct sequence of start ups of the individual applications. Here, it is important to note that it does not matter whether it is performed through the graphical UI represented by the FEXP Solver manager or whether it is about the startup of the native FEXP solver executables. There are two steps that are as follows:

1. As first, the FEXP server must be started. It then automatically starts listening to the network socket for the possible further connections of client computers. All the required information for a successful startup is contained in the configuration file described earlier.

2. If the server startup is successful, as the second step, it is necessary to run executables belonging to the individual client workstations within the computer network establishing the computer cluster. As soon as the number of connected client workstations reaches those defined in the configuration input setting file, the FEXP server automatically starts up the numerical simulation process.

Socket Based Network Communnication

As mentioned earlier, the network communication is secured through a low-level connection via the socket provided by the Windows OS interface. Compared to common enterprise software systems, the hybrid-parallel FEXP solver is not the SOA paradigm reliant. Some

of the SOA based technologies were briefly dealt with in the previous chapters devoted to cloud solutions. Advanced high-level technologies such as .NET/WCF are usually used for distributed enterprise software systems, and they use the socket form of communication, but on the lowest layers of their architecture. It is obviously hidden from the software developer. In the current development phase, the FEXP solver application is Windows OS dependant due to the usage of a Win32 API library for the network communication. The use of portable Boost.Asio library is still in the testing phase.

For network communication, available protocols of a network transport layer are in particular a delivery secured TCP protocol or delivery not secured UDP protocol. However, the UDP protocol is used primarily for network broadcasting or multimedia communications where partial data loss is not critical. For example, it is worthwhile to mention Microsoft Skype, which uses UDP communication protocol. However, a number of security risks are associated with UDP protocol. Its use is often blocked by a system firewall. As a result, a number of professional interventions are required to ensure the communication tunnel through a system firewall to provide secure communication. For the mentioned reasons and many others, the TCP protocol has been selected, thus the hybrid-parallel FEXP solver uses communication supported by opening a low-level network TCP/IP socket using a specific protocol for subsequent communication designed exclusively for the purpose of FEXP solver.

Within the hybrid-parallel FEXP solver, the macro entity model data transfer over the computer network managed by the analysis of MEIM often occurs. The transfer of such a data file between the server and the client workstation within the computer network using the TCP/IP protocol is illustrated in the Fig. 633.

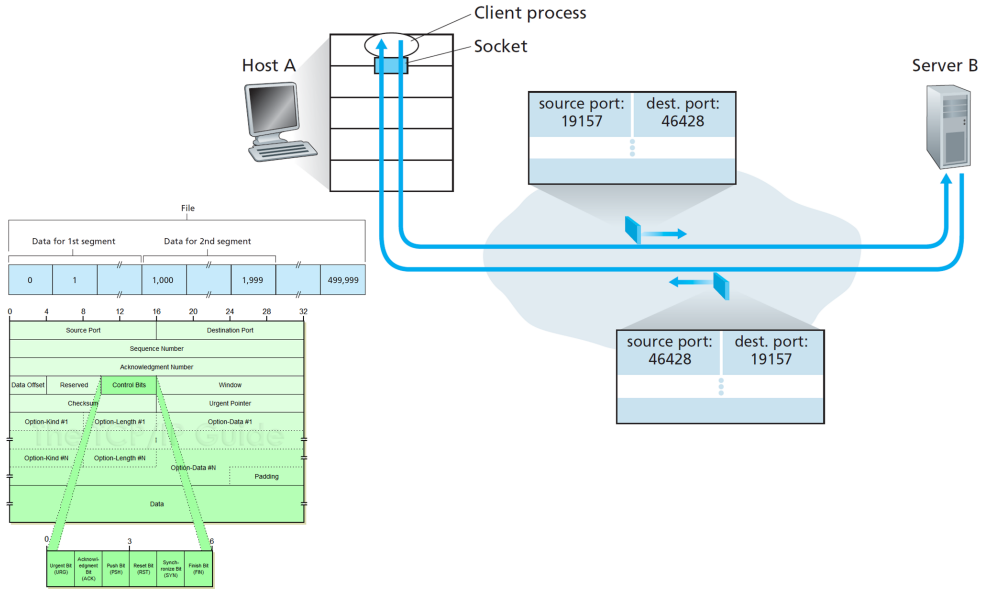


Figure 633: Example of data file sending using TCP/IP packets in client-server architecture.

For the purposes of reading and writing to the Windows socket, the **class CFEXPNETWinMessage** functionality is used. The class definition contains the following code from the source file `FEXPNetworkWinSocket.h`.

```
#define MAX_MSG_IN_LEN 0x1000 // 1 x 4096 [B]
#define MAX_MSG_OT_LEN 0x1000 // 1 x 4096 [B]
/** @brief Communication message.
*/
class CFEXPNETWinMessage
{
public:
  /** @brief Content of request message.
  */
  using REQUEST =
    struct rqs_msg
  {
    char        record[MAX_MSG_IN_LEN]; // message content
    rqs_msg() { set_to_default(); }
    void set_to_default ()
    {
      memset((void*)&record, FEXPCOMMON_ZERO_END_CHAR, sizeof(record));
    }
    void set_data_to_send(const std::string & data)
    {
      set_to_default();
      data.copy((char*)record, MAX_MSG_IN_LEN);
    }
  };
  /** @brief Content of response message.
  */
  using RESPONSE =
    struct rsp_msg
  {
    char        record[MAX_MSG_OT_LEN]; // message content
    rsp_msg() { set_to_default(); }
    void         set_to_default()
    {
      memset((void*)&record, FEXPCOMMON_ZERO_END_CHAR, sizeof(record));
    }
    std::string get_send_data ()
    {
      return CFEXPBaseConvers::StrTrim(std::string((char*)record));
    }
  };

  // for win socket communication
  static t_ENetMessage WinSocketDataRead (SOCKET socket, CFEXPNETWinMessage::REQUEST & request,
    CFEXPNETWinMessage::RESPONSE & response, t_ModelData & data);
  static void          WinSocketDataWrite(SOCKET socket, CFEXPNETWinMessage::REQUEST & request,
    CFEXPNETWinMessage::RESPONSE & response, t_ModelData & data, t_ENetMessage message);
protected:
  // for win socket communication
  static bool          win_write_to_socket (REQUEST  & request ,
    SOCKET socket, const std::string        & send_data);
  static t_ENetMessage win_read_from_Socket(RESPONSE & response,
    SOCKET socket, std::vector<std::string> & read_data);
private:
  // [no private members  ] ------------------------------
};
```

The appropriate socket write/read functions includes the following code listing from the source file `FEXPNetworkWinSocket.cpp`.

```
bool CFEXPNETWinMessage::win_write_to_socket(REQUEST & request, SOCKET socket,
```

```
  const std::string & send_data)
{
  auto result = true;
  try
  {
    request.set_data_to_send(send_data);
    auto result = send(socket, (const char *)request.record,
      MAX_MSG_OT_LEN, FEXPCOMMON_DEFAULT_VALUE);
    if (result != MAX_MSG_OT_LEN)
      FEXPCOMMON_EXCEPTION("Error: Sent less data than buffer size: "
        + CFEXPBaseConvers::NumberToString(result) + "!!!");
  }
  catch (const std::exception & ex)
  {
    CFEXPLog::WriteLine();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "An exception occurred:");
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "-> " + ex.what());
    result = false;
  }
  return result;
}

t_ENetMessage CFEXPNETWinMessage::win_read_from_Socket(RESPONSE & response, SOCKET socket,
  std::vector<std::string> & read_data)
{
  auto mssg = t_ENetMessage::eError;
  try
  {
    // get and send data
    size_t recv_len = FEXPCOMMON_DEFAULT_VALUE; std::string recv_message;
    do
    {
      response.set_to_default();
      auto result = recv(socket, (char*)response.record,
        MAX_MSG_OT_LEN, FEXPCOMMON_DEFAULT_VALUE);
      if (result == FEXPCOMMON_DEFAULT_VALUE)
        FEXPCOMMON_EXCEPTION("Error: Socket closed (try reading)!!!");
      if (result == SOCKET_ERROR)
        FEXPCOMMON_EXCEPTION("Error: Socket error reading!!!");
      recv_len     += result;
      recv_message += response.get_send_data();
    } while (recv_len != MAX_MSG_OT_LEN); // check received message length

    // try get message
    mssg = CEFEXPNETProtocol::GetNetMessage(recv_message);
    if (mssg == t_ENetMessage::eMessageCount)
      read_data.push_back(recv_message);
  }
  catch (const std::exception & ex)
  {
    CFEXPLog::WriteLine();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "An exception occurred:");
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "-> " + ex.what());
  }
  return mssg;
}
```

The listed functions already assume the connection has already been established and nothing prevents network communication otherwise the system generates an appropriate error message. The communication settings are performed both on the server side and on the client side, respectively. On the side of network server, the particular settings are represented by the following code from the source file `FEXPNetworkServer.h`. The listening

to the socket is set to be in blocking form, the setting for a non-blocking form of socket is commented on in the code for testing purposes.

```
template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
void CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier     >::value>::type
  >
  ::create_socket()
{
#define ERROR_SERVER_SOCKET_CREATION "Error: Cannot create socket (server): \n"
  // information about the Windows Sockets implementation
  WSADATA socket_info;
  // initiates use of the Winsock DLL by a process
  auto result = WSAStartup(WINSOCKET, &socket_info);
  if (CONNECTION_ERRORS_1.count(result))
    FEXPCOMMON_EXCEPTION(ERROR_SERVER_SOCKET_CREATION + CONNECTION_ERRORS_1[result]);

  // fill server info for connection
  memset(&_socket_server_adress, FEXPCOMMON_DEFAULT_VALUE, sizeof(_socket_server_adress));
  _socket_server_adress.sin_family      = WINNETWRK;
  _socket_server_adress.sin_addr.s_addr = WINANYADR;
  _socket_server_adress.sin_port        = htons((u_short)GetCommPort());
  // create server socket
  _socket_server = socket(WINNETWRK, WINSCTYPE, WINPR_TCP/*WINPR_DMY*/);
  if (CONNECTION_ERRORS_2.count(result))
    FEXPCOMMON_EXCEPTION(ERROR_SERVER_SOCKET_CREATION + CONNECTION_ERRORS_2[result]);

  // associating a local address with a socket.
  result = bind(_socket_server, (sockaddr*)&_socket_server_adress,
    sizeof(_socket_server_adress));
  if (CONNECTION_ERRORS_4.count(result))
    FEXPCOMMON_EXCEPTION(ERROR_SERVER_SOCKET_CREATION + CONNECTION_ERRORS_4[result]);

  /*// non-blocking recv on socket
  struct timeval tv;
  tv.tv_sec = 10;
  if (setsockopt(_socket_server, SOL_SOCKET, SO_RCVTIMEO, (char *)&tv, sizeof tv))
  ;*/
}
```

The creation of a socket on the side of the client workstation provides an instance of the **class CFEXPNetClientWinSocketConnection** whose definition and implementation contain the source files `FEXPNetworkWinSocket.h` and `FEXPNetworkWinSocket.cpp`, respectively. Appropriate settings for the socket creation then contain the following source code listing.

```
void CFEXPNetClientWinSocketConnection::create_socket()
{
#define ERROR_CLIENT_SOCKET_CREATION "Error: Cannot create socket (client): \n"
  // information about the Windows Sockets implementation
  WSADATA socket_info;
  // initiates use of the Winsock DLL by a process
  auto result = WSAStartup(WINSOCKET, &socket_info);
  if (CONNECTION_ERRORS_1.count(result))
    FEXPCOMMON_EXCEPTION(ERROR_CLIENT_SOCKET_CREATION + CONNECTION_ERRORS_1[result]);

  // create client socket
  _socket = socket(WINNETWRK, WINSCTYPE, WINPR_TCP);
  if (CONNECTION_ERRORS_2.count(result))
```

```
    FEXPCOMMON_EXCEPTION(ERROR_CLIENT_SOCKET_CREATION + CONNECTION_ERRORS_2[result]);

  // server IP address
  in_addr ip_address;
  memset(&ip_address, FEXPCOMMON_DEFAULT_VALUE, sizeof(ip_address));
  if (inet_pton(WINNETWRK, GetServerNodeIP().c_str(), &ip_address) == WINNONADR)
  {
    Close();
    FEXPCOMMON_EXCEPTION(ERROR_CLIENT_SOCKET_CREATION
      + std::string("Null host info --> socket is closed."));
  }

  // fill info for connection
  memset(&_sock_server_adress, FEXPCOMMON_DEFAULT_VALUE, sizeof(_sock_server_adress));
  _sock_server_adress.sin_addr.s_addr = ip_address.S_un.S_addr;
  _sock_server_adress.sin_family      = WINNETWRK;
  _sock_server_adress.sin_port        = htons((u_short)GetCommPort());

  /*// non-blocking recv on socket
  struct timeval tv;
  tv.tv_sec = 10;
  if (setsockopt(_socket, SOL_SOCKET, SO_RCVTIMEO, (char *)&tv, sizeof tv))
  ;*/
}
```

Both on the server side and on the client side, the appropriate settings for the socket are performed during the applications startup. The server side then takes care of accepting the individual client requests for connection. Accepting an individual client connection is then executed as a non-blocking operation, primarily due to the requirement to not run a special working thread for such an action. The source code for the connection accepting operation is contained in the server loop which is presented in further chapters. The following fragment of a source code from the implementation of the server therefore represents the specific such functionality.

```
/** @brief It starts the server process.
*/
template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
void CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier      >::value>::type
  >
  ::Run()
{
#define SERVER_NEW_CLIENT_CONNECT 1
  if (_connection_start.load())
    return;
  // run server thread -- non-blocking accept server --> timeout_ptr != nullptr
  auto result = CFEXPAsyncRunner::RunProcedureTask([this]
    {

...

      while (!_flag_server_close.load() && !_socket_server_run_and_closed.load())
      {
        // network calculation synchronization points
        switch (synchronize_stat_id)
        {

...
```

```
        case t_ENetSynStgs::eClientUpdate:
          {
            fd_set read_set; FD_ZERO(&read_set); FD_SET(_socket_server, &read_set);
            // the maximum time for select to wait,
            // it is provided in the form of TIMEVAL struct,
            timeval timeout;
            timeout.tv_sec  = FEXPCOMMON_DEFAULT_VALUE;
            timeout.tv_usec = FEXPCOMMON_DEFAULT_VALUE;
            // set the timeout parameter to nullptr for blocking operations
            auto timeout_ptr = &timeout /*nullptr*/;
            // new number of threads for update of thread barrier
            auto new_thr_num = size_t(FEXPCOMMON_DEFAULT_VALUE);
            // determines the status of one or more sockets, waiting if necessary,
            // to perform synchronous I/O.
            if (select(_socket_server, &read_set, nullptr, nullptr, timeout_ptr)
              != SERVER_NEW_CLIENT_CONNECT)
              try_rmv_client_thread(); // remove ended client thread if it is need
            else
            {
              // new client node connected
              sockaddr_in from; int from_len = sizeof(from);
              new_thrd_id             = ++client_id_generator;
              auto new_client_socket = accept(
                _socket_server, (struct sockaddr*)&from, &from_len);
              auto new_client_ip_adr = std::string(inet_ntoa(from.sin_addr));
              CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Info: New client connected:");
              CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "--> IPv4: " + new_client_ip_adr);
              add_new_client_thread(new_thrd_id, new_client_socket, new_client_ip_adr);
            }
          }

          // check if main process can start
          check_main_process_init();
          // synchronize and move to new stage
          if (!synchronize_process(synchronize_stat_id++))
            return;
          break;

...

      }
    }, false, "Server Process", true);

...

}
```

The last necessary step required for the seamless functionality of the network-distributed application is to gracefully disconnect each established connection. This in turn concerns both the server side and also the client side of an application, respectively. The graceful termination of the listening to the socket on the side of server (server connection itself and also server client connection) is represented by the following code from the source files `FEXPNetworkServer.h` and `FEXPNetworkWinSocket.cpp`, respectively.

```
template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
void CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier      >::value>::type
  >
```

```
  ::close_socket_conn()
{
  if (_socket_server == WININVSCK)
    return;
  closesocket(_socket_server);
  WSACleanup();
  _socket_server = WININVSCK;
}
```

and

```
void CFEXPNetServerWinClient::close_connection()
{
  // disallow sends and receives
  shutdown(_socket, WINSOCSDW);
  // close client socket
  closesocket(_socket);
  WSACleanup();
  _socket = WININVSCK;
}
```

The same such functionality contains the client side of an application which represents the following code from the source file `FEXPNetworkWinSocket.cpp`.

```
void CFEXPNetClientWinSocketConnection::Close()
{
#define ERROR_SOCKET_CLIENT_CLOSE_CONNECT ...
  "Error: Client connection has already been closed:\n"
  if (!_connection_start)
  {
    auto error = ERROR_SOCKET_CLIENT_CLOSE_CONNECT + std::string("--> Node ID \"")
      + CFEXPBaseConvers::NumberToString<size_t>(GetId()) + "\".";
    FEXPCOMMON_EXCEPTION(error);
  }
  if (_socket == WININVSCK)
    return;
  // disallow sends and receives
  shutdown(_socket, WINSOCSDW);
  // close and clean up
  closesocket(_socket);
  WSACleanup();
  _socket = WININVSCK;
  _connection_start.store(false);
}
```

Server Side of Solver

For the purpose of the hybrid-parallel solution, the FEXP solver is split into a server and client part of an application represented by two separately compiled executables. The server side of the hybrid-parallel FEXP solver represents the control part of the entire solution process. The design of a server is similar to those used in distributed enterprise systems for transactional operations within the database.

For the purpose of connecting individual client workstations and subsequent server-client communication, the mechanism for creation, execution and termination of a client threads on the side of server is applied. The server itself ensures synchronization of client threads and consequently the process of numerical simulation as a whole. Another important activity of the server is the export of simulation results and an analysis of the

numerical model data distribution within the computer network depending on the analysis of MEIM.

The core of the server functionality is represented by the template **class CFEXPNetServer** whose definition and implementation contains source file `FEXPNetworkServer.h`. The creation of an instance respective server class is realised through the design pattern *singleton*. Establishing the server instance is represented by the following source code fragment contained in the source file `fexp_net_server_main.cpp`

```
/** @brief Main function of a server node (hybrid-parallel FEXP solver).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{

...

  if (...) ...
  else
  {

...

    // get instance of server
    auto server_instance = CFEXPNetServer<
      SERVER_CLIENT_DEFINITION, SERVER_CLIENT_THREAD_DEFINITION, SERVER_THREAD_BARRIER>
        ::GetInstance(std::get<FEXPCOMMON_CMD_SERVER_IP_INDEX>(cmd),
          std::get<FEXPCOMMON_CMD_SERVER_PORT_INDEX>(cmd),
            builder->GetModelContainer(), result_exporter, data_trans_analyzer);
    // run server in special control thread
    CFEXPCppThread<void()> server_control_thread([&] { server_instance->Run(); });
    server_control_thread.StartThread();
    server_control_thread.Detach();
    // loop controling request for stopping the server activity by special file existence
    while (true)
    {
      auto path = std::experimental::filesystem::absolute(SERVER_STOP_FILE_NAME);
      if (std::experimental::filesystem::exists(path))
        break;
      using namespace std::chrono_literals;
      std::this_thread::sleep_for(1s);
    }

...

    // close server
    server_instance->Close();
  }

...

}
```

From the source code listing it is clear that the server is started in the special working thread, thus the main thread is not blocked. After the server startup, the main thread proceeds in an infinite loop checking the request from an administrator to suspend the server's activity. Stopping the server process is performed externally by the creation of the special file named `stop_fexp_server.stop` in the directory path of an application startup.

The server setting is defined in the configuration file dealt with in chapter 6.4.2. It

concerns the informations about the client workstations required for the startup of the numerical simulation and the input data of the numerical model. The input data of FE model must be divided into separate files so that each file contains one structural macro entity. Model data contained in the input files are used for possible distribution over the computer network based on an analysis of MEIM. An assembly of settings represents the following part of the source code

```
/** @brief Main function of a server node (hybrid-parallel FEXP solver).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{
  // server network base setting (IP, port), path of solver configuration file
  std::string config_path; std::tuple<std::string, size_t, std::string, size_t> cmd;
  try
  {
    cmd         = GetCmdContent(argv, argc);
    config_path = GetConfigPath(SOLVER_CNFG_FILE_NAME);
  }
  catch (const std::exception & ex)
  {
    CFEXPLog::WriteLine();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "An exception occurred:");
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "-> " + ex.what());
    MessageBox(NULL, (LPCWSTR)L"Command line arguments error!",
      (LPCWSTR)L"FEXP Cmd Error", MB_OK);
    return EXIT_FAILURE;
  }

  // read server setting
  auto reader = CFEXPDataManager<CFEXPFileReader<CFEXPServerInpDataContainer>>
    ::SafeAllocInstance(config_path,
      CFEXPSolverInpDataAssemblyFactory::INP_FILE_BLOCK_CLS_MAP);
  if (!reader->ReadProgress())
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Error: Problem with configuration file!!!");
  else
  {
    // build data
    Ptr<ICFEXPModelBuilderBase> builder = CFEXPDataManager
      <CFEXPServerModelBuilder<ICFEXPDataContIntf, CFEXPMainDataContainer>>
        ::SafeAllocInstance();
    if (!builder->BuildModelProgress(config_path, reader->GetFileContent(),
      FEXPCOMMON_DYNCAST(CFEXPFEInpContBase,
        ICFEXPDataContIntf, reader->GetInputContainer())))
    {
      FEXPCOMMON_CONSOLE_PAUSE(std::get<FEXPCOMMON_CMD_MANAGER_INDEX>(cmd));
      return EXIT_FAILURE;
    }

    // solver setting
    auto setting = FEXPCOMMON_DYNCAST(ICFEXPModelDataIntf, CFEXPSolverConfigSetting,
      builder->GetModelContainer()->GetModelElement(
        ICFEXPSetting::ESettingType::eSolver, ESystemElementType::eSetting));

...

  }

...

}
```

Server activity is ensured by a loop taking care of the synchronization of all commu-

nication activities within the computer network. These activities include connecting and disconnecting individual workstations, the export of results, data analysis, synchronization of time, and the other auxiliary procedures. The loop contains the following source code

```
/** @brief It starts the server process.
*/
template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
void CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier     >::value>::type
  >
  ::Run()
{
#define SERVER_NEW_CLIENT_CONNECT 1
  if (_connection_start.load())
    return;
  // run server thread -- non-blocking accept server --> timeout_ptr != nullptr
  auto result = CFEXPAsyncRunner::RunProcedureTask([this]
    {
      // start listenning on port
      start_listenning();
      // count connected clients -- generate ID
      auto client_id_generator = size_t(FEXPCOMMON_DEFAULT_VALUE);
      auto synchronize_stat_id = size_t(t_ENetSynStgs::eStartPoint);
      auto new_thrd_id         = size_t(FEXPCOMMON_DEFAULT_VALUE);
      auto init_start_clnts_ok = false;
      while (!_flag_server_close.load() && !_socket_server_run_and_closed.load())
      {
        // network calculation synchronization points
        switch (synchronize_stat_id)
        {
        case t_ENetSynStgs::eStartPoint:
          update_barrier(t_ENetSynStgs(synchronize_stat_id));
          // synchronize and move to new stage
          if (!synchronize_process(synchronize_stat_id++))
            return;
          break;
        case t_ENetSynStgs::eBeforeClientUpdate:
          update_barrier(t_ENetSynStgs(synchronize_stat_id));
          // send current data to clients
          invoke_client_Data();
          // synchronize and move to new stage
          if (!synchronize_process(synchronize_stat_id++))
            return;
          break;
        case t_ENetSynStgs::eClientUpdate:
          {
            fd_set read_set; FD_ZERO(&read_set); FD_SET(_socket_server, &read_set);
            // the maximum time for select to wait,
            // it is provided in the form of TIMEVAL struct,
            timeval timeout;
            timeout.tv_sec  = FEXPCOMMON_DEFAULT_VALUE;
            timeout.tv_usec = FEXPCOMMON_DEFAULT_VALUE;
            // set the timeout parameter to nullptr for blocking operations
            auto timeout_ptr = &timeout /*nullptr*/;
            // new number of threads for update of thread barrier
            auto new_thr_num = size_t(FEXPCOMMON_DEFAULT_VALUE);
            // determines the status of one or more sockets, waiting if necessary,
            // to perform synchronous I/O.
            if (select(_socket_server, &read_set, nullptr, nullptr, timeout_ptr)
              != SERVER_NEW_CLIENT_CONNECT)
```

```
            try_rmv_client_thread(); // remove ended client thread if it is need
          else
          {
            // new client node connected
            sockaddr_in from; int from_len = sizeof(from);
            new_thrd_id            = ++client_id_generator;
            auto new_client_socket = accept(
              _socket_server, (struct sockaddr*)&from, &from_len);
            auto new_client_ip_adr = std::string(inet_ntoa(from.sin_addr));
            CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Info: New client connected:");
            CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "--> IPv4: " + new_client_ip_adr);
            add_new_client_thread(new_thrd_id, new_client_socket, new_client_ip_adr);
          }
        }

        // check if main process can start
        check_main_process_init();
        // synchronize and move to new stage
        if (!synchronize_process(synchronize_stat_id++))
          return;
        break;
      case t_ENetSynStgs::eAfterClientUpdate:
        // synchronize and move to new stage
        if (!synchronize_process(synchronize_stat_id++))
          return;
        break;
      case t_ENetSynStgs::eEndPoint:
        update_barrier(t_ENetSynStgs(synchronize_stat_id));
        // synchronize and move to new stage
        if (!synchronize_process(synchronize_stat_id++))
          return;
        break;
      case t_ENetSynStgs::eCheckCalcEnd:
        update_barrier   (t_ENetSynStgs(synchronize_stat_id));
        // receive info about calculation process
        send_msg_and_wait(t_ENetThrdMsg::eEnd, new_thrd_id);
        // synchronize and move to new stage
        if (!synchronize_process(synchronize_stat_id++))
          return;
        break;
      case t_ENetSynStgs::eStartNewTimeStep:
        update_barrier(t_ENetSynStgs(synchronize_stat_id));
        // export results --> send last data to export --> end process
        check_proc_end(new_thrd_id);
        // synchronize and move to new stage
        if (!synchronize_process(synchronize_stat_id++))
          return;
        break;
      case t_ENetSynStgs::eResults:
        update_barrier   (t_ENetSynStgs(synchronize_stat_id));
        // receive calculation results from clients
        send_msg_and_wait(t_ENetThrdMsg::eCalcResults, new_thrd_id);
        // export results
        export_results   (new_thrd_id);
        // synchronize and move to new stage
        if (!synchronize_process(synchronize_stat_id++))
          return;
        break;
      case t_ENetSynStgs::eStabilityControl:
        update_barrier      (t_ENetSynStgs(synchronize_stat_id));
        // receive stabil time steps from clients
        send_msg_and_wait   (t_ENetThrdMsg::eStep, new_thrd_id);
        // set global time step
        update_global_t_step(new_thrd_id);
        // invoke global time step preparation
```

```
          send_msg_and_wait    (t_ENetThrdMsg::eStep, new_thrd_id);
          // synchronize and move to new stage
          if (!synchronize_process(synchronize_stat_id++))
            return;
          break;
        case t_ENetSynStgs::eDataExchange:
          update_barrier       (t_ENetSynStgs(synchronize_stat_id));
          // receive bounding boxes of macro models
          send_msg_and_wait    (t_ENetThrdMsg::eBoundingBox, new_thrd_id);
          // make analysis about need of data network reconfiguration
          analyze_data_transfer(new_thrd_id);
          // invoke message again to invoke waiting for data preparation
          send_msg_and_wait    (t_ENetThrdMsg::eBoundingBox, new_thrd_id);
          // --> 1. data from clients to server
          send_msg_and_wait    (t_ENetThrdMsg::eBoundingBox, new_thrd_id);
          // --> 2. data from server to clients
          // synchronize and move to new stage
          if (!synchronize_process(synchronize_stat_id++))
            return;
          break;
        case t_ENetSynStgs::eEndLoop:
          update_barrier(t_ENetSynStgs(synchronize_stat_id));
          // in common situation there are no data to exchange
          if (_main_proc_initialize_control && _main_proc_initialize_clients)
          {
            // reset data analyzer
            if (_transfer_analyzer)
              _transfer_analyzer->Reset();
          }
          // synchronize
          if (!synchronize_process(synchronize_stat_id))
            return;
          // move to new stage (reset value --> new synchronization loop)
          synchronize_stat_id = size_t(t_ENetSynStgs::eStartPoint);
          // reset added last client id
          new_thrd_id         = FEXPCOMMON_DEFAULT_VALUE;
          // reset time step value
          _stabil_dt_step     = GET_MAX_FLT_VAL(t_fexpcommon_ct);
          break;
        default:
          FEXPCOMMON_EXCEPTION("Error: Wrong process synchronization state!!!");
        }
      }
    }, false, "Server Process", true);
  // invoke close server
  invoke_close(); _flag_server_closed.store(true);
}
```

Each connection to the server realized in a scope of considered computer network, is represented by just one running thread that takes care of network communication with one connected workstation. Establishing a new service thread for communication with the remote client is provided by the following source code

```
template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
void CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier     >::value>::type
  >
  ::add_new_client_thread(size_t id, SOCKET client_socket, std::string & ip)
{
#define MAP_THREAD_IDX        FEXPCOMMON_DEFAULT_INDX
```

```
#define MAP_THREAD_VALUE_IDX 1
  if (_client_thread_map.count(id))
    FEXPCOMMON_EXCEPTION("Error: Try add new client thread with existing ID!!!");
  std::unique_lock<std::mutex> lock(_mtx_client_thread_map_update, std::defer_lock);
  lock.lock();
  auto cnt_tm = std::time(nullptr);
  CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Info: New cluster node ID "
    + CFEXPBaseConvers::NumberToString(id) + ":");
  CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "--> Connected at "
    + std::string(std::ctime(&cnt_tm)));
  // create new client thread data, start thread and detach
  auto server_thread_data = CFEXPDataManager<thread_data>
    ::SafeAllocInstance(id, create_client_thread(id, client_socket, ip),
      FEXPCOMMON_DYNCAST(ICFEXPModelDataIntf, CFEXPSolverConfigSetting,
        _model_cont->GetModelElement(
          ICFEXPSetting::ESettingType::eSolver, ESystemElementType::eSetting)));
  // add thread data to map
  _client_thread_map.insert(MAP_PAIR(server_thread_data->_id, server_thread_data));
  // update number of threads in eStartPoint barrier only
  // --> permutation process
  _barrier[t_ENetSynStgs::eStartPoint] =
    CFEXPDataManager<Tbarrier>::SafeAllocInstance(_client_thread_map.size() + 1);
  // start thread and detach
  server_thread_data->_thread->StartThread();
  server_thread_data->_thread->Detach();
  // set active state
  server_thread_data->_state = true;
  // if calculation is running and new client is connected --> data are required
  server_thread_data->_current_msg_to_send = _main_proc_initialize_control ?
    t_ENetThrdMsg::eRuntimeInit : t_ENetThrdMsg::eContinueProc;
  // need update barriers
  _barrier_update = EBarrierUpdate::eAddUpdate;
  // set request for runtime data printing
  set_rnt_request();
  lock.unlock();
}
```

The source code from the listing above is focused on adding a new working thread whose activity will then be subsequently synchronized with the other already running working threads to ensure the simultaneous sharing of the execution of all the critical actions with other worskstations within the computer network during a numerical simulation. All the actions required for communication with the network client node are contained in a special **class CFEXPNetServerWinClient**, from the source files `FEXPNetworkWinSocket.h` and `FEXPNetworkWinSocket.h`, respectively. It takes care of the details of the data exchange. The communication of the server client working thread with the server thread is provided by an interface that represents a set of *lambda* functions entering the class constructor of a server client thread. Instantiating of this class is secured on the server side by the following source code

```
template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
Ptr<TClientThread> CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier      >::value>::type
  >
  ::create_client_thread(size_t id, SOCKET client_socket, std::string & ip)
{
  return CFEXPDataManager<TClientThread>
```

```
    ::SafeAllocInstance([this](auto client) { client->Run(); },
      CFEXPDataManager<TTClientRunner>
        ::SafeAllocInstance(client_socket, ip, id, GetServerNodeIP(),
      [this](auto id_from, auto id_to           )
        { return read_message(id_from, id_to);   },
      [this](auto id_from, auto id_to           )
        { return lnch_message(id_from, id_to);   },
      [this](auto id_from, auto id_to, auto mssg)
        { send_message(id_from, id_to, mssg);    },
      [this](auto id_from                       )
        { return get_calc_data(id_from);         },
      [this](auto id_from                       )
        { set_end_client_thread(id_from);        },
      [this](auto synstat                       )
        { return synchronize_process(synstat);   },
      [this](auto id_from, auto dt              )
        { set_stabil_t_step(id_from, dt);        },
      [this](auto id_from, auto bbox , auto mid )
        { add_macro_to_tree(bbox, id_from, mid); }
    ));
}
```

To begin the numerical computational process over the network, it is necessary to connect the number of workstations from the computer network that is defined in the setting input file as noted in chapter 6.4.2. In the initial phase, the input data are divided according to the number of workstations required to start the computation. If the computation is already running, it is necessary to decide which data from the existing connected workstations within the computer network should be moved to the newly connected client computer. This decision is based on the analysis of MEIM at runtime in a certain computational stage. The result of such an analysis may also be the decision leading to the conclusion that the newly connected client workstation will not be loaded by a numerical computation yet. However, in order for such a workstation to be ready for later use, the so-called default model data are sent and it does not contain any specific FE model, but only information about the computation settings and the other data relating to e.g. structural load and constrains, respectively. Data initialization then provides the following source code on the server side as previously described.

```
template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
void CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier      >::value>::type
  >
  ::invoke_client_data()
{
  Ptr<std::vector<size_t>> schedule; std::queue<size_t> init_models;
  std::queue<size_t> init_models_schedule;
  FEXPCOMMON_FOREACH_ITER(_client_thread_map)
  {
    auto key = (*IT).first;
    auto dta = (*IT).second;
    auto msg = dta->_current_msg_to_send;
    switch (msg)
    {
    case t_ENetThrdMsg::eInit:
      {
```

```
        // server setting
        auto setting = FEXPCOMMON_DYNCAST(ICFEXPModelDataIntf, CFEXPSolverConfigSetting,
          _model_cont->GetModelElement(
            ICFEXPSetting::ESettingType::eSolver, ESystemElementType::eSetting));
        if (!schedule)
        {
          schedule = setting->GetSimpleModelSchedule();
          // fill queue with models
          FEXPCOMMON_FOREACH_ITER_FNC(setting->GetFileContent(),
          { init_models.push(IT.first); });
          // fill queue with number of models for each client
          FEXPCOMMON_FOREACH_ITER_FNC(*schedule.get(),
          { init_models_schedule.push(IT); });
        }
        auto default_model = init_models.empty();
        auto ifile_content = [=](auto id)
          {
            return (default_model ?
              setting->GetFileContentDefault() :
              setting->GetFileContent())[id];
          };
        // set default empty data if no other model is available
        if (default_model)
        {
          FEXPCOMMON_FOREACH_ITER_FNC(setting->GetFileContentDefault(),
            { init_models.push(IT.first); });
          init_models_schedule.push(init_models.size());
        }
        // add number of models for each client
        size_t idx; size_t mode_count = init_models_schedule.front();
        FEXPCOMMON_FOREACH(FEXPCOMMON_DEFAULT_VALUE, mode_count - 1, idx)
        {
          auto model_id = init_models.front();
          dta->_calc_data->CURRENT_DATA_FOR_SEND.insert(
            MAP_PAIR(CFEXPBaseConvers::NumberToString(model_id),
              ifile_content(model_id)));
          init_models.pop();
        }
        init_models_schedule.pop();
        // start client initialize completed
        _main_proc_initialize_clients = true;
      }
      break;
    case t_ENetThrdMsg::eRuntimeInit:
      {
        // server setting
        auto setting = FEXPCOMMON_DYNCAST(ICFEXPModelDataIntf, CFEXPSolverConfigSetting,
          _model_cont->GetModelElement(
            ICFEXPSetting::ESettingType::eSolver, ESystemElementType::eSetting));
        FEXPCOMMON_FOREACH_ITER_FNC(setting->GetFileContentDefault(),
          {
            // send empty data and let client wait for data exchange event
            dta->_calc_data->CURRENT_DATA_FOR_SEND.insert(
              MAP_PAIR(CFEXPBaseConvers::NumberToString(IT.first), IT.second));
          });
      }
      break;
    default:
      msg = t_ENetThrdMsg::eContinueProc; // continue in process
    }
    // send message to client
    send_message(GetId(), key, msg);
    // reset message
    dta->_current_msg_to_send = t_ENetThrdMsg::eThrdMsgCount;
  };
```

```
}
```

Due to the initialization of the newly connected workstations, but also for the new data model distribution over the network, an analysis of MEIM is required. For such an analysis, the server requests the data containing the bounding box of each macro entity occurring in the numerical simulation. Similarly to the contact solution in the scope of individual workstations, an analysis of contact or more precisely potentional contact of the macro entity bounding boxes is performed. Contact is solved using the range searching query to the kd-tree data structure, but with the difference relating to the amount of data, where analyzed data are much smaller compared to the detailed data containing the individual nodes of the FE meshes in the individual workstations. Thus, the MEIM analysis here is divided into two functions, namely the to function inserting the macro entity bounding box data into the MEIM analyzer, and then to the function that subsequently obtains the data required for the network model decomposition from the analyzer. The given functions are listed bellow.

```
template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
void CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier      >::value>::type
  >
  ::add_macro_to_tree(Ptr<CFEXGeomTools::t_BoundBox> bbox, size_t thrd_id, size_t macro_id)
{
  if (!_transfer_analyzer)
    return;
  std::unique_lock<std::mutex> lock(_mtx_kdtree_update, std::defer_lock);
  lock.lock();
  // add macro bounding box to macro contact analyzer
  _transfer_analyzer->AddMacro(bbox, macro_id, thrd_id);
  lock.unlock();
}

template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
void CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier      >::value>::type
  >
  ::analyze_data_transfer(size_t new_thrd_id)
{
  if (!(_main_proc_initialize_control && _main_proc_initialize_clients))
    return;
  ++_meim_analysis_counter;
  // clear data from previous session
  FEXPCOMMON_FOREACH_ITER_FNC(_client_thread_map,
    {
      IT.second->_calc_data->MODEL_DATA_GET.clear();
      IT.second->_calc_data->MODEL_DATA_SET.clear();
    });
  // analyze data transfer
  if (!_transfer_analyzer->Analyze())
    return;
  // save data from analysis for data transfer
  auto data_from  = _transfer_analyzer->GetTransferComputerModelFrom ();
```

```
  auto data_to    = _transfer_analyzer->GetTransferComputerModelTo    ();
  auto data_assoc = _transfer_analyzer->GetMacroToComputerAssociation();
  if      (data_from.empty() && data_to.empty())
    return;
  else if (data_from.empty() || data_to.empty())
    FEXPCOMMON_EXCEPTION("Error: Wrong data transfer analysis!!!");
  CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Info: Transfer of data required.");
  // store data for data transfer
  FEXPCOMMON_FOREACH_ITER(_client_thread_map)
  {
    auto tid = IT->first;
    auto dta = IT->second->_calc_data;
    if (data_from.count(tid))
      FEXPCOMMON_FOREACH_ITER_FNC(data_from[tid],
      { dta->MODEL_DATA_GET.push_back(IT); });
    if (data_to.count(tid))
      FEXPCOMMON_FOREACH_ITER_FNC(data_to  [tid],
      { dta->MODEL_DATA_SET.push_back(std::make_tuple(IT, data_assoc[IT])); });
  };
  // try to write current transfer data to file
  write_out_transfer();
}
```

The most important step of the entire numerical simulation is the export of the result data as in the case of a single workstation. Compared to the process running on a single workstation only, the resulting data from a computer cluster must be merged into one output file. The following server function takes care of the assembly of such result data.

```
template<typename TTClientRunner, typename TClientThread, typename Tbarrier>
bool CFEXPNetServer
<TTClientRunner, TClientThread, Tbarrier,
  typename std::enable_if
  <
  std::is_base_of<ICFEXPThreadBase          , TClientThread>::value &&
  std::is_base_of<ICFEXPSynchrThreadBarrier, Tbarrier      >::value>::type
  >
  ::export_results(size_t new_thrd_id)
{
  auto result_exported = false;
  if (!(_main_proc_initialize_control && _main_proc_initialize_clients))
    return result_exported;

  std::map<size_t, Ptr<CFEXPCalculationModelNodeResult>> thread_result_map;
  FEXPCOMMON_FOREACH_ITER(_client_thread_map)
  {
    auto thrd_id = (*IT).first;
    if (thrd_id == new_thrd_id)
      continue;
    auto results = (*IT).second->_calc_data->CURRENT_DATA_RECEIVED;
    if (results.empty())
      continue;

    // print out runtime calculation data
    auto resulto_rows = results["rso"];
    if (resulto_rows && !resulto_rows->empty())
    {
      auto output = resulto_rows->at(FEXPCOMMON_DEFAULT_INDX);
      if (output != FEXPCOMMON_EMPTY_STRING)
        CFEXPLog::WriteLine(output);
    }

    auto result_rows = results["rs"];
    if (!result_rows || result_rows->empty())
      continue;
```

```
    // read serialized results
    auto reader = CFEXPDataManager<CFEXPFileReader<CFEXResultDataContainer>>
      ::SafeAllocInstance(FEXPCOMMON_EMPTY_STRING, result_rows,
        CFEXPResultDataModelAssemblyFactory::RES_FILE_BLOCK_CLS_MAP);
    reader->Read();
    // build result instance
    CFEXPDataManager<CFEXPModelBuilder<ICFEXPDataContIntf, ICFEXPDataModelContIntf>>
      ::SafeAllocInstance(_model_cont, FEXPCOMMON_EMPTY_STRING, FEXPCOMMON_EMPTY_STRING,
        FEXPCOMMON_EMPTY_STRING)->BuildModel(
          CFEXPBaseConvers::NumberToString(thrd_id), reader->GetFileContent(),
            FEXPCOMMON_DYNCAST(CFEXPFEInpContBase, ICFEXPDataContIntf,
              reader->GetInputContainer()));
    if (_model_cont->ItemCount(ESystemElementType::eResult) > 1)
      FEXPCOMMON_EXCEPTION("Error: Data from more time levels arrived!!!");
    // add to result map
    _model_cont->IterateModElems([&thread_result_map, &thrd_id](auto item)
      {
        thread_result_map.insert(MAP_PAIR(thrd_id,
          FEXPCOMMON_STACAST(ICFEXPModelDataIntf, CFEXPCalculationModelNodeResult, item)));
        return true;
      }, ESystemElementType::eResult);

    // clear lunched result data
    results.clear();
    // clear results from current client and prepare for next one
    _model_cont->RemoveAll(ESystemElementType::eResult);
  }
  if (thread_result_map.empty())
    return result_exported;

  Ptr<CFEXPCalculationModelNodeResult> results;
  FEXPCOMMON_FOREACH_ITER(thread_result_map)
  {
    if (!results)
    {
      results = (*IT).second;
      continue;
    }
    // node results
    FEXPCOMMON_FOREACH_ITER_FNC((*IT).second->GetNodeResults (),
      { results->GetNodeResults ().push_back(IT); });
    // fe connectivity
    FEXPCOMMON_FOREACH_ITER_FNC((*IT).second->GetConnectivity(),
      { results->GetConnectivity().push_back(IT); });
  }

  // final export of results
  if (_result_exporter)
    _result_exporter->Write(results);
  return !result_exported;
}
```

The server then contains a number of other functions, which are primarily of an auxiliary character and are related to the control of the overall main functionality. By and large, the given functionality relates mainly to the synchronization of client threads during I/O processes in terms of individual computational phases. Specific communication and the data exchange is primarily performed by running client threads on the side of server. Those include program logic of the data exchange between the cluster node and the central server. Such a specific process is then presented in the following part of the chapter.

Client Side of Solver

The client side of the hybrid-parallel FEXP solver represents the specific activity focused primarily on the numerical computation. Unlike in the case, where all computing processes run on a single workstation only, in the client side of hybrid-parallel FEXP solver it requires the additional functionality to process synchronization within the computer cluster. The **class CFEXPHybridParallelSolver** contains the mentioned functionality. It inherits the class **class CFEXPParallelSolver**, where it overwrites some of the specific parts of the parent class.

An important difference here is the need to run a special thread instance that primarily deals with communication with the server. For such a reason, the amount of computational threads is reduced by one thread. However, the disadvantage of missing one hardware thread for numerical computation should be compensated by the decomposition of larger models within the computer cluster. The function that is responsible for the creation of computational threads and also of one communication thread then contains the following source code listing.

```
template<typename TExpCalc, typename TThreadData, typename TThread, typename Tbarrier>
Ptr<TThread> CFEXPHybridParallelSolver<TExpCalc, TThreadData, TThread, Tbarrier>
  ::get_thread_instance(size_t id, Ptr<TThreadData> data, bool is_calc_thread)
{
  return is_calc_thread ?
    // create calculation thread
    CFEXPDataManager<TThread>::SafeAllocInstance(
      [this](auto it1, auto it2)
      {
        CFEXPHybridParallelSolver::thread_Calculation_function(it1, it2);
      }, id, data) :
    // create thread for network communication purpose
    CFEXPDataManager<TThread>::SafeAllocInstance(
      [this](auto it1, auto it2)
      {
        CFEXPHybridParallelSolver::main_loop_net_communication(it1, it2);
      }, id, data);
}
```

The function to manage numerical computations are the same as for a single workstation not linked to a computer cluster. It consists of a few adjustments relating to synchronization purposes required for network communication. See the following source code listing for comparison.

```
template<typename TExpCalc, typename TThreadData, typename TThread, typename Tbarrier>
void CFEXPHybridParallelSolver<TExpCalc, TThreadData, TThread, Tbarrier>
  ::thread_Calculation_function(size_t thread_id, Ptr<TThreadData> data)
{
  try
  {
    bool calculation_end = false; auto calculator = get_calculator(thread_id);
    while (!_socket_closed)
    {
      // 1. Step: Set time consistency
      data->SetCalcTimeIncrement(thread_id,
        calculator->SimulationTimeIncrement(thread_id));
      Synchronize(ESynchronization::eSync1);
      // 2. Step: Check if time is the same for all threads
      // this require only one thread to do this action
      check_time_synchronization(data);
```

```
      // check end of calculation
      calculation_end = check_calculation_end(data, thread_id);
      Synchronize(ESynchronization::eSync2);
      // 3. Step: While is end wait for result sending
      //--> process non-calc thread communicates with server
      Synchronize(ESynchronization::eSync3);
      // 4. Step: Check end of calculation
      if (calculation_end)
        break;
      Synchronize(ESynchronization::eSync4);
      // 5. Step: Prepare data for new time level
      calculator->PrepareDataForNewTimeLevel (thread_id);
      // 6. Step: Update mapping of fe nodes for contact search
      // this require only one thread to do this action
      update_model_mapping(thread_id);
      // 7. Step: Transform position and velocities to local coordinate system
      calculator->GlobalToLocalTransformation(thread_id);
      Synchronize(ESynchronization::eSync5);
      // 8. Step: Integration of internal, external and contact forces
      calculator->CalculateForces            (thread_id);
      Synchronize(ESynchronization::eSync6);
      // 9. Step: Calculate new motion based on explicit integration equations of motion
      calculator->CalculateNewGeometry       (thread_id);
      Synchronize(ESynchronization::eSync7);
      // 10. Step: Save calculation results --> process non-calc thread communicates with server
      Synchronize(ESynchronization::eSync8);
      // 11. Step: Control of calculation stability
      calculator->StabilityControl           (thread_id);
      data->SetTimeStep(thread_id, calculator->GetCalculatedCriticTimeStep());
      Synchronize(ESynchronization::eSync9);
      // 13. Step: Synchronize time steps with network clients
      //--> process non-calc thread communicates with server
      Synchronize(ESynchronization::eSync10);
      // 15. Step: Network data exchange
      //--> process non-calc thread communicates with server
      Synchronize(ESynchronization::eSync11);
    }
  }
  catch (const std::exception & ex)
  {
    CFEXPLog::WriteLine();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT
      + "Error: Calculation ended before simulation end.");
    // print problem of exception
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "An exception occurred:");
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "--> " + ex.what());
  }
}
```

The function that takes care of communication with the central server has the following content

```
template<typename TExpCalc, typename TThreadData, typename TThread, typename Tbarrier>
void CFEXPHybridParallelSolver<TExpCalc, TThreadData, TThread, Tbarrier>
  ::main_loop_net_communication(size_t id, Ptr<TThreadData> data)
{
  try
  {
    bool calculation_end = false;
    while (!_socket_closed)
    {
      Synchronize(ESynchronization::eSync1);
      Synchronize(ESynchronization::eSync2);
      // check calculation end --> send resuts
      calculation_end = end_calculation(data);
```

```
      Synchronize(ESynchronization::eSync3);
      // check calculation end --> end of loop
      if (calculation_end)
        break;
      Synchronize(ESynchronization::eSync4);
      Synchronize(ESynchronization::eSync5);
      Synchronize(ESynchronization::eSync6);
      Synchronize(ESynchronization::eSync7);
      // export of results
      result_export(data);
      Synchronize(ESynchronization::eSync8);
      Synchronize(ESynchronization::eSync9);
      // stability --> time step synchonization
      time_step_synchronization(data);
      Synchronize(ESynchronization::eSync10);
      // macro contact --> data exchange
      // 1. send macro model bounding boxes
      macro_model_BB_data_send   (data);
      // 2. send macro model serialized calculation data
      macro_model_calc_data_send (data);
      // 3. receive macro model input data
      macro_model_input_data_recv(data);
      // 4. receive macro model serialized calculation data
      macro_model_calc_data_recv (data);
      Synchronize(ESynchronization::eSync11);
    }
  }
  catch (const std::exception & ex)
  {
    CFEXPLog::WriteLine();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT
      + "Error: Calculation ended before simulation end.");
    // print problem of exception
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "An exception occurred:");
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "--> " + ex.what());
  }
}
```

From the listing of the source code it is clear that the data communication consists the following:

- Synchronization of the simulation time, i.e. the size of the time step and the number of its increment.
- Data exchange regarding the current spatial position and shape of a solved macro entities within the respective computer cluster node.
- Depending on the server-side analysis of MEIM, the data exchange of the respective macro entities is performed. It is either in one or both directions. In the case of bi-directional data exchange of macro entities, a complete change of the included solved models can be performed within the corresponding individual workstation in the scope of the computer cluster.
- The last part is the export of the numerical simulation results for the appropriate simulation time.

The functions from the source code listing above include a large number of functionality dealing in detail with the serialization and deserialization of the respective compu-

tational data. Those of the aforementioned functionality relating to the network data communication symmetrically relate to the server side code. As mentioned earlier, on the server side, the communication with the corresponding workstation contains the code in **class CFEXPNetServerWinClient** in source files `FEXPNetworkWinSocket.h` and `FEXPNetworkWinSocket.cpp`. A communication loop within it continuously runs through the computing steps as shown in the following source code listing.

```
void CFEXPNetServerWinClient::main_loop()
{
  CFEXPNETWinMessage::REQUEST request;
  try
  {
    auto synchronize_state = size_t(t_ENetSynStgs::eStartPoint);
    while (true)
    {
      ThrowIfCancelRequest<SERVER_CLIENT_THREAD_DEFINITION>();
      // run next stage
      if (!run_calc_synchr_stage(synchronize_state++))
        break;
      // reset stage to default (the first stage)
      if (synchronize_state == size_t(t_ENetSynStgs::eSynchrCount))
        synchronize_state = size_t(t_ENetSynStgs::eStartPoint);
    }
  }
  catch (const std::exception & ex)
  {
    std::string error = SERVER_CLIENT_EXCEPTION
      + CFEXPBaseConvers::NumberToString<size_t>(GetId())
      + FEXPCOMMON_NEW_LINE + ex.what();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + error);
  }
  // finish client
  client_finishing();
}

bool CFEXPNetServerWinClient::run_calc_synchr_stage(size_t stage)
{
  auto continue_next = true;
  try
  {
    // network calculation synchronization points
    switch (stage)
    {
    case t_ENetSynStgs::eStartPoint:
    // synchronize and move to new stage
    continue_next = _thread_synchronizer(stage);
    break;
    case t_ENetSynStgs::eBeforeClientUpdate:
      load_model_data(get_server_data());
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
      break;
    case t_ENetSynStgs::eClientUpdate:
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
      break;
    case t_ENetSynStgs::eAfterClientUpdate:
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
      break;
    case t_ENetSynStgs::eEndPoint:
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
```

```
      break;
    case t_ENetSynStgs::eCheckCalcEnd:
      if (_calculation_started)
      {
        CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Info: Client ID: "
          + CFEXPBaseConvers::NumberToString(GetId()) + ", calculation end control ...");
        calculation_end_control(get_server_data(false));
      }
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
      break;
    case t_ENetSynStgs::eStartNewTimeStep:
      if (_calculation_started && !process_continue_control(get_server_data()))
      {
        continue_next = false;
        break;
      }
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
      break;
    case t_ENetSynStgs::eResults:
      if (_calculation_started)
      {
        CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Info: Client ID: "
          + CFEXPBaseConvers::NumberToString(GetId()) + ", simulation results ...");
        simulation_results(get_server_data());
      }
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
      break;
    case t_ENetSynStgs::eStabilityControl:
      if (_calculation_started)
      {
        CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Info: Client ID: "
          + CFEXPBaseConvers::NumberToString(GetId()) + ", stability control ...");
        stability_control(get_server_data());
      }
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
      break;
    case t_ENetSynStgs::eDataExchange:
      if (_calculation_started)
      {
        CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Info: Client ID: "
          + CFEXPBaseConvers::NumberToString(GetId()) + ", data exchange ...");
        data_exchange_control(get_server_data());
      }
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
      break;
    case t_ENetSynStgs::eEndLoop:
      // synchronize and move to new stage
      continue_next = _thread_synchronizer(stage);
      break;
    default:
      FEXPCOMMON_EXCEPTION("Error: Wrong process synchronization state!!!");
    }
  }
  catch (const std::exception & ex)
  {
    std::string error = SERVER_CLIENT_EXCEPTION
      + CFEXPBaseConvers::NumberToString<size_t>(GetId())
      + FEXPCOMMON_DFLT_TXT + FEXPCOMMON_NEW_LINE + ex.what();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + error);
    continue_next = false;
```

```
  }
  return continue_next;
}
```

A detailed study of the communication functions content and also all others can then be done through the study of the respective application's source files.

Figure 634: Native command line vs. FEXP Solver Manager behaviour of hybrid-parallel FEXP solver (two connected client workstations to server).

As in the case of FEXP solver not linked to the computer cluster, even in the case of a hybrid-parallel FEXP solver, the computational process can be controlled through the FEXP Solver Manager. The FEXP solver configuration setting is similar to those shown in the Fig. 628, 629 and 630, respectively. An important note here is the type of setting related to the content of the configuration input file for the hybrid-parallel FEXP solver. It has to include the definition of the IP address and port of the FEXP server (see

Fig. 628), the list of input files containing the model definitions of the individual macro entities including the default input file (see Fig. 629), and the list of the IP addresses of the client workstations whose connection to the server is the signal for the start-up of the numerical simulation (see Fig. 630). Last but not least, it is necessary to emphasize the selection of the correct executable [*.exe] related to the FEXP server. The behaviour of the computation is then similar to those shown in the Fig. 631.

Since the hybrid-parallel FEXP solver is a distributed application type, it is necessary to divide the FEXP Solver Manager into two parts. The part relating to the FEXP server as described above, and to the part on the side of individual workstations connected to the computer cluster. The part on the cluster node side has the sole purpose of executing the respective executable [*.exe] and tracking its behaviour through a graphical UI, respectively. The process of monitoring is performed through the instance of the **class FEXPProcessMonitorDlg** from source file `FEXPProcessMonitorDlg` written in the C# programming language, as in the case of the rest of the source codes belonging to the FEXP Solver Manager application.

An example of a numerical simulation running in the computer cluster is shown in the Fig. 634, where it runs within the console, and as a counterpart is shown the same process but controled by the FEXP Solver Manager with graphical UI for client process control. Finally, it is need to list the startup C++ code from source file `fexp_net_client_main.cpp` on the side of client workstation, which is as follows

```
/** @brief Main function of a client node (hybrid-parallel FEXP solver).
*/
auto __cdecl main(int argv, char* argc[]) -> int
{
  // path of solver configuration file
  std::tuple<std::string, size_t, std::string, size_t> cmd;
  try
  {
    cmd = GetCmdContent(argv, argc);
  }
  catch (const std::exception & ex)
  {
    CFEXPLog::WriteLine();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "An exception occurred:");
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "-> " + ex.what());
    MessageBox(NULL, (LPCWSTR)L"Command line arguments error!",
      (LPCWSTR)L"FEXP Cmd Error", MB_OK);
    return EXIT_FAILURE;
  }

  // input model data reader
  auto lambda =
  [](const std::string & key, Ptr<std::vector<std::string>> data)
    {
      return CFEXPDataManager<CFEXPFileReader<CFEXPInpDataContainer>>
        ::SafeAllocInstance(key, data,
          CFEXPInpDataModelAssemblyFactory::INP_FILE_BLOCK_CLS_MAP);
    };
  auto result = size_t(EXIT_SUCCESS);
  try
  {
    function_traits<decltype(lambda)>::f_type get_reader_instance = lambda;
    auto client = CFEXPDataManager<CFEXPNetClientWinSocketConnection>
      ::SafeAllocInstance(FEXPCOMMON_DEFAULT_VALUE,
```

```
    std::get<FEXPCOMMON_CMD_SERVER_IP_INDEX  >(cmd),
    std::get<FEXPCOMMON_CMD_SERVER_PORT_INDEX>(cmd),
  [get_reader_instance, cmd, &result](std::function<void(size_t)> id_sttr,
    ICFEXPNetClientNodeService & service)
    {
      // initialize client
      // 1. wait for model initialize (model data, client ID)
      CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT +
        "1. Waiting for data arrival from server (model data, client ID).");
      do
      {
        auto data = service.ReadInstruction();
        if(!data)
          FEXPCOMMON_EXCEPTION(
            "FEXP Cluster Client main Error: Data initialization failed!!!");
        if (data->count(t_ENetMessage::eInitialize))
        {
          auto macro = ICFEXPNetClientNodeService
            ::GetReadModelData(data, t_ENetMessage::eInitialize);
          // add init data to macro model container
          FEXPCOMMON_FOREACH_ITER_FNC(macro,
            { service.AddMacroModelData(IT.first, IT.second); });
        }
        else if (data->count(t_ENetMessage::eIdInit))
        {
          id_sttr((CFEXPBaseConvers::StringToNumber<size_t>(
            ICFEXPNetClientNodeService
              ::GetReadModelData(data, t_ENetMessage::eIdInit)
                ["ID"]->at(FEXPCOMMON_DEFAULT_INDX))));
          break;
        }
        else
          FEXPCOMMON_EXCEPTION(
            "FEXP Cluster Client main Error: " +
            "Unknown message, initialization failed!!!");
      } while (true);

      // 3. initialize data by reader
      CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "3. Initialization.");
      auto rnt_init_data_key =
        CFEXPBaseConvers::NumberToString(FEXPCOMMON_DEFAULT_VALUE);
      std::map<std::string, Ptr<ICFEXPFileReaderIntf>> _file_reader_map;
      auto read_succ   = true;
      auto is_rnt_init = false;
      FEXPCOMMON_FOREACH_ITER(*(service.GetMacroModelData()).get())
      {
        auto key = (*IT).first;
        if (!is_rnt_init && key == rnt_init_data_key)
          is_rnt_init = true;
        _file_reader_map.insert(MAP_PAIR(key, get_reader_instance(key, (*IT).second)));
        if (_file_reader_map[key]->ReadProgress())
          continue;
        read_succ = false;
        break;
      }
      if (!read_succ)
      {
        FEXPCOMMON_CONSOLE_PAUSE(std::get<FEXPCOMMON_CMD_MANAGER_INDEX>(cmd));
        result = EXIT_FAILURE;
        return;
      }

      // 4. build initial fem model
      CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "4. Building FE model.");
      Ptr<ICFEXPModelBuilderBase> builder =
```

```
            CFEXPDataManager<
              CFEXPModelBuilder<ICFEXPDataContIntf, CFEXPMathModelElementContainer>>
                ::SafeAllocInstance(
                  "Building finite element model",
                  "Removing finite element model",
                  "Scheduling of threads");
          auto build_succ = true;
          FEXPCOMMON_FOREACH_ITER(_file_reader_map)
          {
            if (builder->BuildModelProgress((*IT).first, (*IT).second->GetFileContent(),
              FEXPCOMMON_DYNCAST(CFEXPFEInpContBase, ICFEXPDataContIntf,
                (*IT).second->GetInputContainer())))
              continue;
            build_succ = false;
            break;
          };
          if (!read_succ)
          {
            FEXPCOMMON_CONSOLE_PAUSE(std::get<FEXPCOMMON_CMD_MANAGER_INDEX>(cmd));
            result = EXIT_FAILURE;
            return;
          }

          // while runtime initialization --> input macro data remove (dummy macro data)
          if (is_rnt_init)
            builder->RemoveStructure(rnt_init_data_key);

          // 5. create and run solver
          CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT +
            "5. Run Solver (start of computation).");
          CFEXPDataManager<HYBRID_PARALLEL_SOLVER>::SafeAllocInstance(
            [get_reader_instance](auto key, auto data)
            {
              auto reader = get_reader_instance(key, data);
              reader->Read();
              return reader->GetInputContainer();
            }, builder,
            [](auto id) { return CFEXPBaseConvers::NumberToString(id); },
            service)->Start();
        });

    // run client
    client->Run();
    // close client
    client->Close();
    // end of computation
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "Computation process successfully ended.");
  }
  catch (const std::exception & ex)
  {
    CFEXPLog::WriteLine();
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "An exception occurred:");
    CFEXPLog::WriteLine(FEXPCOMMON_DFLT_TXT + "--> Process ended.");
    result = EXIT_FAILURE;
  }
  // wait for cmd key if need
  FEXPCOMMON_CONSOLE_PAUSE(std::get<FEXPCOMMON_CMD_MANAGER_INDEX>(cmd));
  return result;
}
```

In the source code listed above, it is clear to see a complete data initialization procedure for the client process. In the client initialization phase, data are obtained from the side of the server. The received data contains the currently-stacked model's macro entities for

the numerical computation in the respective client workstation of the computer cluster. After an obtaining the client ID granted by the server, the FE model build process then continues. The last step is entry to the startup function of the computational procedure. The given procedures of the initialization and numerical computation startup are triggered by the following source code

```
void CFEXPNetClientWinSocketConnection::Run()
{
#define ERROR_SOCKET_CLIENT_CONNECT "Error: Client cannot start: \n"
  if (_connection_start.load())
  {
    auto error = ERROR_SOCKET_CLIENT_CONNECT + std::string("--> Node ID \"")
      + CFEXPBaseConvers::NumberToString<size_t>(GetId()) + "\":\n"
      + "Socket has already been opened.";
    FEXPCOMMON_EXCEPTION(error);
  }

  // create socket and connect
  create_socket();
  auto result = connect(_socket,
    (sockaddr*)&_sock_server_adress, sizeof(_sock_server_adress));
  if (result == SOCKET_ERROR)
  {
    // throw error connection exception
    auto laste = WSAGetLastError();
    auto error = ERROR_SOCKET_CLIENT_CONNECT + std::string("Node ID --> ") + "\""
      + CFEXPBaseConvers::NumberToString<size_t>(GetId()) + "\":\n";
    if(CONNECTION_ERRORS_3.count(laste))
      error += CONNECTION_ERRORS_3[laste];
    FEXPCOMMON_EXCEPTION(error);
  }
  // now we are connect to server
  _connection_start.store(true);
  // start main client process loop
  _main_loop([this](auto id) { SetId(id); }, *this);
}
```

The first step here is an attempt to establish a connection between the sides of server and the appropriate client workstation. If the attempt to connect to the server is successful, the initialization and computational procedures are subsequently started. The final execution of the mentioned procedures represents the following snippet a source code.

```
void CFEXPNetClientWinSocketConnection::Run()
{

...

  // start main client process loop
  _main_loop([this](auto id) { SetId(id); }, *this);
}
```

6.5 Summary of Chapter

In this chapter important aspects of the software implementation of the proposed solution to an explicit numerical integration of equations of motion in terms of the massive parallel computing were introduced.

Massively parallel calculations today belong to a very large area, including modern computer networks, operating systems and of course to the area of advanced hardware devices. For these reasons, a part of the text was also devoted to cloud systems, whose potential usage in the future is enormous.

The great part of the chapter was devoted mainly to the particular description of important parts of the FEXP solver with the presented fragments of the related source code. Then the UML diagrams related to the object design should be an aid in the overall understanding of the application's composition.

The total amount of source code rows of the FEXP solver and FEXP Solver Manager applications is enormous with regard to the context of the application purpose as a test software tool. This is the reason why the program source code could not be presented in its entire width and depth.

Chapter 7

Results of Simulation Test

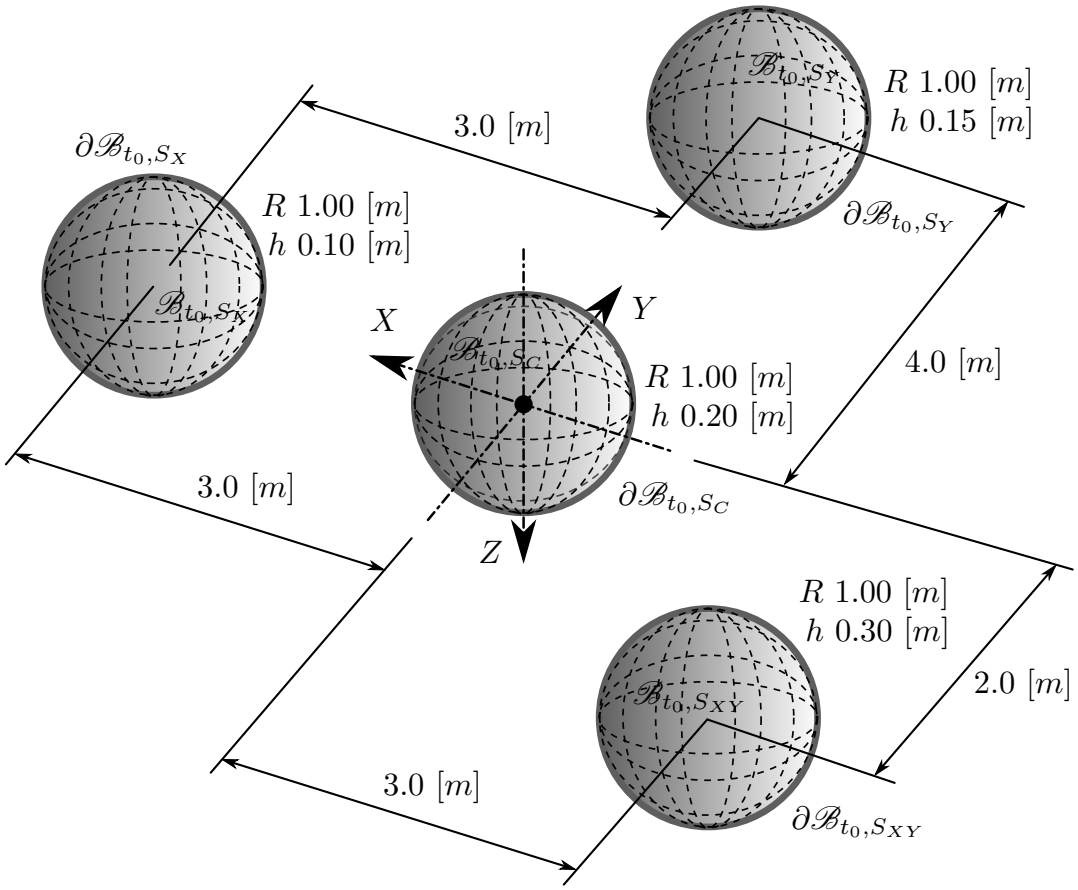


Figure 71: Configuration of spheres at time t_0.

For the testing purposes of the proposed solution represented by the hybrid-parallel FEXP solver, the problem of multibody impact was chosen. The geometrical data of the

respective bodies of the mathematical-physical model and their initial configurations are shown in the Fig. 71. This type of example was chosen primarily due to the expected fluctuation of the individual parts of the model within the simulated computer network during the solution process.

The model geometry and FE meshes were created in the RFEM program. The triangular FE meshes were then extracted from files [*.XYZ] (coordinates of FE nodes) and [*.E2D] (connectivity of 2D FE), respectively. The respective files are used by the numerical kernel (the NE-XX solver from the FEM consulting company) of the RFEM program for the FE model assembly process. The FE statistics are included in the Tab. 71.

Table 71: FE mesh statistics.

	Statistics
2D finite elements	768 (≈ 0.016 $[m^2 m^{-2}]$)
FE mesh nodes	392

The movement of bodies is initiated by the initial conditions represented by the velocity constraints shown in the Fig. 72, where small velocities in Z direction are introduced primarily due to the applied type of contact detection algorithm which is represented by node-to-element contact. This artificial numerical impurity avoids the state represented by element-to-element contact.

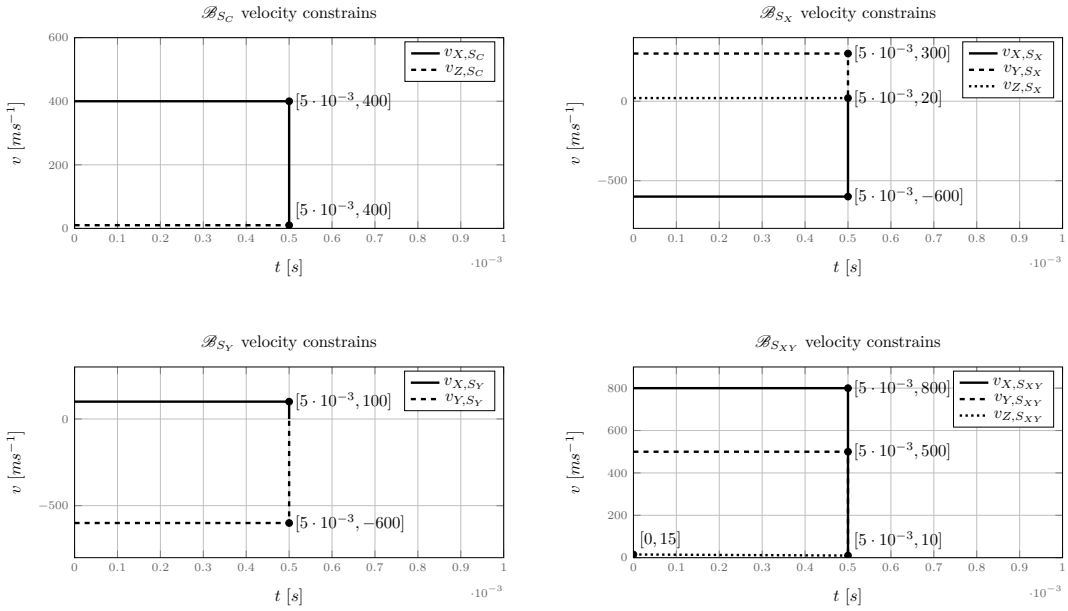


Figure 72: Velocity initial conditions of individual macro entities.

All velocity constraints are switched off at the time $t = 0.0005s$, and subsequently, all of the bodies can move freely based on their inertia. The high initial body velocities are chosen primarily due to the short duration of the transient dynamic simulation and due to their higher destructive effect. For all macro entities (bodies) the same type of material is chosen. The material characteristics are listed in the Tab. 72.

Table 72: Material characteristics.

Characteristics	**Value**	**Unit**
E	210.0	GPa
G	80.8	GPa
ν	0.3	–
ϱ	7850.0	$kg\ m^{-3}$

For network type computations, a computer grid consisting of 3 workstations is chosen for the initiation of the computation. The 4th workstation is connected to the grid during the computation. The initial configuration of the given computer network is schematically shown in the Fig. 73. It shows the initial placement of the macro entity data over the network.

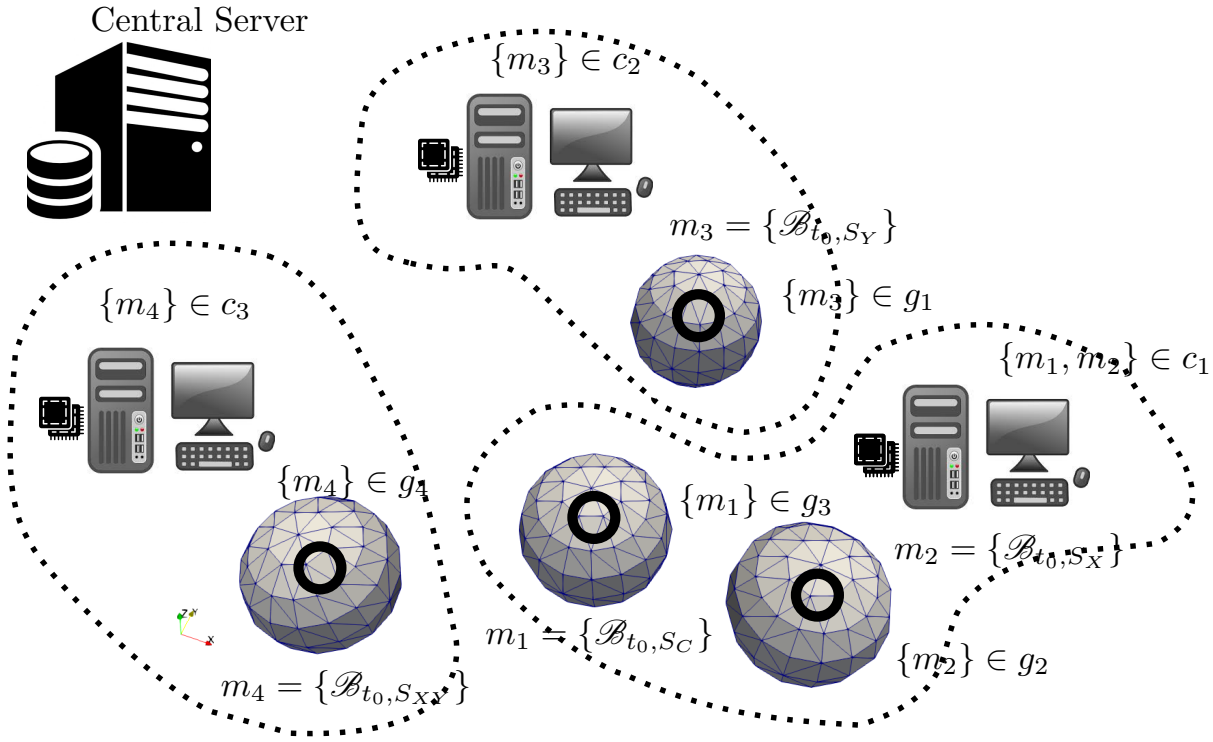


Figure 73: Macro entity distribution over the network at time t_0.

Simultaneously with the parallel processing of tha data over the computer network, the calculation takes place in parallel even within each individual workstation. The number of threads is set uniformly to 3 parallel running threads, i.e. 2 threads for computation and 1 thread for communication with the server. During the calculation, another client workstation was connected to the server. This workstation acquired the necessary data (so-called default data) to be ready for its eventual involvement into the computation.

The model data distribution over the network represented by the initial state lasted until the time step t_{51}, when the first data transfer was initiated based on the analysis of MEIM. At this time step, the computational data of the m_3 model from workstation c_2 was transferred to workstation c_1. However, the new data distribution over the network lasted for only a short time, and at the time step t_{53} another transfer of model data was initiated again. At this time step, the computational data of the m_4 model from workstation c_3 was transferred to workstation c_1. The two previously mentioned data transfers together with the subsequent new data distribution over the network are represented by the Fig. 74.

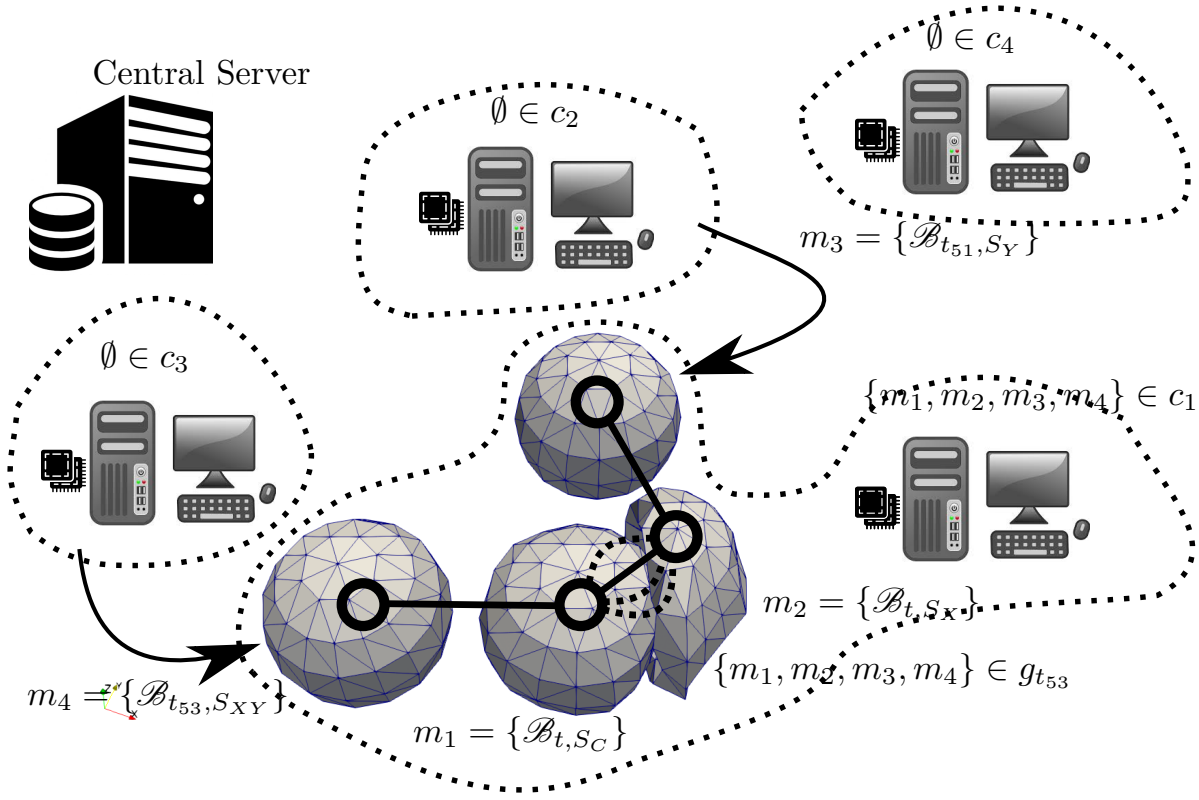


Figure 74: Macro entity distribution over the network at times t_{51} and t_{53}.

After the data transfer from time t_{53} a longer period of time follows in which there is a massive mutual contact interaction between all macro entities. During this time when only one workstation is loaded by numerical computation, all other workstations perform necessary synchronization procedures only.

During the mutual contact interaction of macro entities both a change in their momentum and direction of movement was caused. Thanks to these changes in motion, individual macro entities began to move apart. At time t_{337}, based on the analysis of MEIM, it was assessed that further data transfer is required. In this case, however, it was the most massive data transfer over the network. It was decided that it is necessary to transfer the data of macro entities m_1, $m2$ and m_3 from the workstation c_1 to the workstations c_2, c_3 and c_4, respectively. This event, along with the new data distribution over the network, is presented in the Fig. 75.

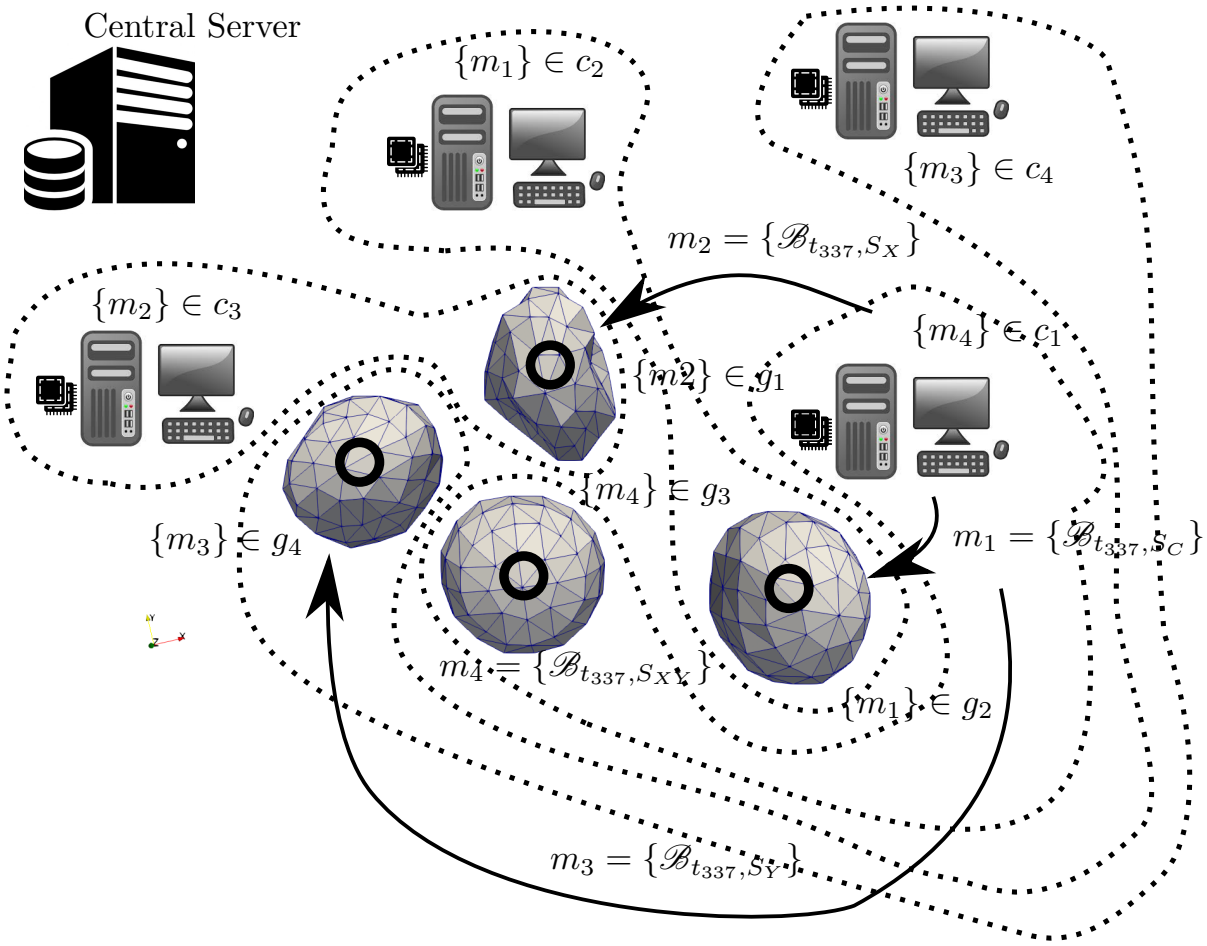


Figure 75: Macro entity distribution over the network at time t_{337}.

The newly acquired state at time t_{337} then lasted until the end of the numerical simulation, i.e. until the time t_{429}. In conjunction with the given model it can be stated that $\approx 66\%$ of the total time of the numerical simulation was spent on one workstation (c_1), i.e. in the time interval from t_{53} to t_{337}. It may be naive to say that about $\approx 34\%$ of the entire computational time has been saved compared to the numerical simulation, which would take place on one workstation only. Here, of course, it is necessary to include the

time spent relating to the data transfers between workstations and the server. However, it can be concluded that in the case of larger models, lower number of data transfers and with the better heuristics related to the analysis of MEIM, it is possible to significantly accelerate the time spent on numerical computations of a similar type of problem.

The input data files for the numerical computation can be found in the attachement of the thesis. This also applies to the result files for the visualization of the respective dynamic simulation in the Paraview program. In this context, videos containing the respective dynamic simulation were created. They are also part of the attachment. Fluctuation of model's macro entities within the computer network during the numerical solution process represents the Tab. 73.

Table 73: Model's macro entities fluctuation within the computer network (state after transfer of macro entities).

Time	**Workstation**	**Workstations content**
t_0	c_1	$\{m_2\} \in g_2, \{m_1\} \in g_3$
	c_2	$\{m_3\} \in g_1$
	c_3	$\{m_4\} \in g_4$
t_{51}	c_1	$\{m_1, m_2, m_3\} \in g_1$
	c_2	$\emptyset$
	c_3	$\{m_4\} \in g_2$
	c_4	$\emptyset$
t_{53}	c_1	$\{m_1, m_2, m_3, m_4\} \in g_1$
	c_2	$\emptyset$
	c_3	$\emptyset$
	c_4	$\emptyset$
t_{337}	c_1	$\{m_4\} \in g_3$
	c_2	$\{m_1\} \in g_2$
	c_3	$\{m_2\} \in g_1$
	c_4	$\{m_3\} \in g_4$

7.1 Summary of Chapter

This chapter presents the results of the numerical simulation and also provides the results of the correct functionality of the proposed algorithmic solution. The model for dynamic simulation was chosen with respect to both the stability of the numerical integration process, its time consumption and also to the assumed physical behavior, respectively.

With respect to the stability of the direct numerical integration of equations of motion, it is necessary to note that this part of the computation is still in the state of development and therefore it was necessary to take into account the given limitations. During the selection of the appropriate setup of this model, a number of program bugs were revealed. The given bugs were subsequently repaired. Therefore, it can be concluded that this type of model example also provides a suitable environment for further program debugging during further development.

In the context of further development in the field of material modeling, the further gradual process of the convergence of numerical simulation results to the real physical behavior of the respective structures can be expected. Subsequently, the proposed solution can be used for the aforementioned demanding numerical simulations in the field of transport safety and vehicle structure or for risk analysis when designing important structures in civil engineering.

Chapter 8

Conclusions

In the dissertation thesis the current issues of the utilization of parallel computations for the purposes of numerical simulations of dynamic processes for assessment, design and optimization of structures in civil and machinery engineering, respectively, are investigated. In view of the current security requirements concerning e.g. the behavior of road restraint systems in the event of accidental vehicle collisions, the safety of vehicles regarding both pedestrian safety and the safety of passengers inside a vehicle in the event of a crash, the safety of strategically important civil structures (nuclear power plants, water supply-related strategic structures, etc.) against the deliberate criminal attempt for destruction related mainly to the problem of world terrorism, thus the numerical simulation of a contact/impact problems became the main interest. Considered type of a numerical algorithm used for the simulation of contact/impact physical phenomana can also be used for rapid dynamic simulations used in the development of firearms.

A source model for the proposed software system approach to the parallelization of numerical computations became the modern way of data processing concerning the so-called cloud computing, i.e. it is related to the so-called big data. The main problem to deal with relates to the distribution of a numerical computations within a computer cluster composed primarily of computer hardware commonly available on the market. It, of course, often corresponds to the state of computer equipment within a number of engineering design studios.

Since the recent attention for the big data processing in the field of computational mechanics has been pursued to the network and local computer machine based solutions related primarily to the implicit based numerical methods, e.g. in conjunction with the FETI family methods, thus the main objective of interest here has become explicit numerical methods applied in the field of massive paralell computations. Current efforts to paralelize explicit numerical methods consider local type of parallel processing on appropriate workstations requiring the presence of expensive computer hardware. The approach proposed in the thesis is able to not only exploit the possibilities of expensive high-performance hardware, but it is also able to distribute the big data of complex numerical models to the scope of a computer network which further accelerates given numerical computations.

In the developed code there was no need to use third-party libraries to create an entire designed parallel model. It concerns to both the commercial and also to freely available software libraries. However, the use of those libraries is also possible. Since the most recent revision of C++ programming language now allows a number of required activities regarded to parallel computing, also with respect to the considerable speed of the resulting compiled code and of course with respect to the generality of such a programming language even for other purposes of software development, respectively, the mentioned programming languge was chosen as the main abstract instrument for the implementation of the respective algorithms.

During the course of the code development, various types of problems arose. The first problem which arose was the need to effectively solve the problem of general contact in an algorithmic way other than than with a native algorithm which exhibits the quadratic complexity of the algorithm. For the numerical solution of contact of bodies, the data structure called kd-tree is used. It is based on a range-searching query processed in the mentioned data structure to obtain the nodes of all the FEs that get too close to the investigated FE. Whether a given set of FE nodes gets too close to the examined FE indicates the volume of the block shape surrounding the given FE. The surrounding block shape is called a bounding box and the bounding box then indicates the range search query. One searching task is of logarithmic complexity as it is common with binary tree searching algorithms. The range-searching process is then further parallelized in terms of the set of FEs.

The second problem surfaced at a later stage of the code development. The problem relates to the redistribution of the numerical data depending on the current state of the numerical simulation over the network. In the context of the second problem, a new term Macro Entity Interaction Multigraph (MEIM) was invented. It relates to the previously dealt with problem of macro body multi-contact representing closed units of the FE meshes. The non-oriented multigraph structure is further simplified and analyzed to find the connected subgraphs representing the bulk of the data units intended to move together within the range of the considered computer network. An effort is made to provide the best balanced workload for individual workstations. The standard DFS algorithm is applied to find connected subgraphs. Multigraph assembly is based on the same principle as a detailed contact search within one FE, with the difference where the respective bounding box represents each contained body (macro entity) in the entire model.

For the numerical tests, T. Belytschko's C^0 triangular shell FE containing one integration node only, is applied. It was preferred for the low computational demands for its numerical integration, and also for the amount of data required for its definition. As an another type of nonlinear behavior of structures, including the already mentioned general contact problem, the corotational formulation for the respective triangular shell FE is considered as a source of the geometrically nonlinear behavior of structures in the range of small strains but large rotations. Consideration of geomerically non-linear behavior for such an FE is so much easier due to its geometrical definition using the first-order polynomial. The simulation results can then be further validated using the commercial software analytical tool Ansys LS-Dyna, where the respective FE is also implemented.

For the purpose of validating and also the purpose of results presentation, the Paraview software tool is used as it is a popular instrument widespread among the scientific community but also in an engineering practice. The specific result data export is executed into the files whose format is defined within the VTK.

The FEXP solver designed to test the functionality mentioned above is divided into two main parts. The first part concerns computations possible to run on a local workstation only. It uses the cores of the currently installed processor for its configurable parallel computation. The computation can also be performed in the sequential manner of an instruction execution primarily for the debug purpose of numerical computations and for performance comparison tests. It is represented by one executable only. The second part concerns the possibility of solving an appropriate numerical model in the scope of the computer network with local type of parallelization in each of the connected workstations. Such a type of solver is referred to as a hybrid-parallel FEXP solver in the text of thesis. The standard client-server software architecture is applied in the hybrid-parallel FEXP solver. The TCP/IP protocol of the transport/network layer is chosen with conjunction of Berkeley like network socket type in the Windows OS environment.

Many new features of C++ programming language ver. 14 are considered throughout the entire FEXP solver software solution. The most important improvements in the programming language are the possibility of using portable platform-independent native threads for parallel computations and the specific way of handling dynamically allocated memory through the so-called smart pointers similarly as in open source high performance 3D graphics toolkit OpenSceneGraph (OSG), e.g. it is used for the topology mapping of a civil and machinery structures in the already mentioned software tools RFEM and RSTAB developed by Dlubal Software company. A new way of dealing with the dynamically allocated memory then increases the program safety and eliminates the problem of memory leaks. Throughout the entire source code, an emphasis is placed on the general design of the relevant functionality using templates, primarily due to later scalability for further functionality and program maintenance.

Finally, the FEXP Solver Manager was designed to simplify manipulation with the FEXP solver. It gives the ability to set up the solver, edit the solver's input data, run the solver and it also enables the monitoring of computation and the controlling of its behaviour through a graphical user interface (graphical UI or GUI). The functionality of the FEXP Solver Manager is programmed in the C# programming language using the Microsoft .NET Framework 4.6. An emphasis is placed on a modern object oriented and functional type of programming and on the asynchronous execution of a number of complex processes. It is primarily for the control of ongoing processes and for securing the responsivity of the graphical UI. The FEXP Solver Manager is ready to manage the work of both the FEXP solver focused on a single workstation as well as a hybrid-parallel type of the FEXP solver.

Given the long lasting and challenging development of such a distributed software application like the FEXP solver, only a few simpler numerical testing models have been considered. The reasons are the specifics of a multi-process execution control where completely different types of problems occur often with a random character that is natural for the such type of applications compared to common sequential running applications. The

complicated random nature of program errors and the generally time-consuming nature of numerical simulations involved cause considerable complications in the time consumption for the development of the respective application.

8.1 Contributions of the Doctoral Thesis

First and foremost the scientific contribution is realted to a somewhat different view of the distribution of numerical computations in nonlinear dynamics of structures provided by the explicit integration of equations of motion compared to those commonly applied approaches in the field of the parallelization of explicit numerical computations primarily focused on powerful single workstations. This refers to the so-called "big data" in terms of cloud computing, where ordinary workstations connected to a computer network are used for the application of parallel algorithms.

With respect to the effectiveness of such a solution, it was necessary to come up with a parallel algorithm competitive with those normally applied on single powerful workstations. Therefore, a specific type of domain decomposition of the FE mesh was applied. It considers those parts of the FE mesh representing the entire integral parts of the numerical model meaning single bodies (connected subgraphs of the entire FE mesh). Those parts of the FE mesh can interact with the surrounding structures by means of the contact forces only. Direct contact interactions of the individual bodies must be solved together in one of the workstations within the considered computer network. The disadvantage of such an approach is the state, where all the bodies are in such interaction that it is no longer possible to use the given domain decomposition method and it is necessary to move all the data from the workstations within the computer network to a single one.

For the distribution of the respective model data on the network, a specific algorithm has been proposed. However, it does not consider further metadata relating to the performance of individual workstations of the heterogeneous computer cluster as well as the speed of individual connections within the computer network. It is just one of the parts providing a greater scope for further scientific research. It mainly applies to models containing a large number of interacting bodies numerically dealt with within large computer networks.

Another benefit is the parallel solution of a contact problem. Here it relates to the application of the kd-tree data structure used for such purposes. The contact search itself can then be further parallelized. It then also allows further scientific research. It also applies to the implementation of other types of searching algorithms and their subsequent performance analysis, performance tunning and improvements.

The designed architecture of the FEXP solver simply allows the implementation of new algorithms which are not only related to the themes mentioned above. Thanks to the clearly defined interface within the FEXP solver, it is possible to change the content of the individual parts under consideration. It allows the focus to purely be on the details of the specific problem solution without the need to re-develop the entire software solution. Compared to other similar software systems, the FEXP solver has been primarily designed

from the outset for the purpose of implementing parallel algorithms. Therfore, it should be able to flexibly respond to the new approaches in the field of massive parallel computing.

8.2 Future Work

Referring to all of the previously mentioned topics, further development of the FEXP solution can be expected. It concerns both the implementation of new algorithms and also other hardware and software technologies. In the not too distant future, it can be assumed that the development of the FEXP solver will most likely get the capability to numerically solve even other physical phenomenas rather than just the problems related to the mechanics of solids.

However, with a view to the near future, it is necessary to continue with the implementation of the above-mentioned topics. Those have been either only partially solved or they have not already been implemented in the FEXP solver yet. Thus, the author of the dissertation thesis suggests to explore the following:

- Apply all the data structures capable of handling spatial data related to the general contact handling. This concerns both the *octree* data structure and the data structures primarily geared to spatial databases. These are concerned mainly with the various mutations of the data structure of the so-called *R-tree*, which are still the focus of current scientific research.

- Apply the NVIDIA CUDA technology in terms of the GPGPU to some parts of the computations. The first algorithms for the parallel implementation on GPUs should be algorithms related to the detection of body contacts in a complex structural environment. It applies to the algorithms mentioned in the previous item. The framework for connecting the CUDA technology to the FEXP solver is now in the final stage of completion.

- Apply some of the advanced heuristic algorithms to the MEIM analysis. Here, as a suitable candidate, a group of so-a called meta-heuristic algorithms appears. For that purpose, it may be necessary to consider additional information related to both the communication speed within the computer network and the hardware type of the respective workstations of which they are made up.

- Apply into the FEXP solver the Boost.Asio library for the network socket communication as a portable platform-independent technology. Preparations have already been made in this context.

- Apply into the FEXP solver some type of service oriented technology for netwoirk communication. As a first suitable candidate, the Microsoft .NET/WCF technology appears. The given SOA technologies should simplify and also make more robust and secure an inter-process communication within a computer network. Such an

SOA solution is then probably the most suitable candidate for the possible further commercial implementation of the mentioned distributed numerical computations.

- Apply into the FEXP solver the further type of finite elements. In particular, the spatial types of FEs that are computationally demanding in terms of the amount of the required computational operations. Thus, the spatial types of the FEs represent a suitable environment requiring the application of parallel algorithms for performance intensification. In such context, the theoretical preparation for the application of the theory of large deformations has already been carried out as a further source of nonlinear behavior of structures including nonlinear contact.
- It would be interesting to apply further constitutive material laws. Plasticity in regime of small and large strains in terms of the spatial FE seems to be very interesting in such a context.

The above listing of the goals to scientific reasearch and their implementation within the scope of the FEXP solver contains a number of an advanced and often unexplored topics. The mentioned themes are seen as those at least partially prepared for realization in the foreseeable future.

List of Abbreviations

Number Sets

$\mathbb{N}$	Natural numbers set
$\mathbb{Z}$	Integer numbers set
$\mathbb{R}$	Real numbers set
$\mathbb{R}^{[\alpha,\omega]}$	Interval of a real numbers set, $\mathbb{R}^{[\alpha,\omega]} = \{x \in \mathbb{R} \mid \alpha \leq x \leq \omega\}$
$\mathbb{S}$	Unsigned integer numbers set, $\mathbb{S} \in [0, 2^{64}]$ for 64bit platform, $\mathbb{S} \in [0, 2^{32}]$ for 32bit platform
$\mathbb{F}_t$	Floating point numbers set with a precision of t
$\mathbb{F}_t^{[\alpha,\omega]}$	Interval of a floating point numbers set with a precision of t, $\mathbb{F}_t^{[\alpha,\omega]} = \{x \in \mathbb{F}_t \mid \alpha \leq x \leq \omega\}$
$\mathbb{F}_d$	Floating point numbers set with a precision of 15 digits, $\mathbb{F}_d \to \mathbb{F}_{15}^{\left[-1.7\cdot10^{308},1.7\cdot10^{308}\right]}$
$\mathbb{F}_d^{[\alpha,\omega]}$	Interval of a floating point numbers set, $F_d^{[\alpha,\omega]} \to \mathbb{F}_{15}^{[\alpha,\omega]} = \{x \in \mathbb{F}_{15} \mid \alpha \leq x \leq \omega\}$
$\mathbb{F}_d^+$	Posive values of a floating point numbers set including zero, $\mathbb{F}_d^+ \to \mathbb{F}_d^{\left[0,1.7\cdot10^{308}\right]}$

Common Mathematical Functions and Operators

[1]

$\mathbf{b}$	Vector $\mathbf{b}$
b_i	the i^{th} element of vector $\mathbf{b}$
$\|\|\mathbf{b}\|\|$	Norm of vector $\mathbf{b}$
$\mathbf{A}$	Matrix (tenzor) $\mathbf{A}$
a_{ij}	Element of matrix $\mathbf{A}$ at the i^{th} row, and the j^{th} column
$\mathbf{A}^{-1}$	Inverse matrix to matrix $\mathbf{A}$
$\mathbf{A}^T$	Transposed matrix to matrix $\mathbf{A}$
$\det \mathbf{A}$	Determinant of matrix $\mathbf{A}$
$\text{tr } \mathbf{A}$	Trace of matrix $\mathbf{A}$
$\text{lin } \mathbf{A}$	Linearization of tensor $\mathbf{A}$
$\partial_\alpha f(\mathbf{x})$	Partial derivative $\frac{\partial f(\mathbf{x})}{\partial x_\alpha}$
$\partial_x f(\mathbf{x}, y)$	Partial derivative $\frac{\partial f(x,y)}{\partial x}$
$O(x)$	The big O notation

[1] It contains the list of important or frequently used mathematical operators in the text of the dissertation thesis. Mathematical functions and operators are always explained in the text, thus the following list is included here primarily for the purpose of better orientation.

Miscellaneous Abbreviations [2]

CPU	Central Processing Unit
GPU	Graphics Processing Unit
GPGPU	General-Purpose computing on Graphics Processing Units
OpenMP	Open Multi-Processing
MPI	Message Passing Interface
Open MPI	Open Message Passing Interface
CUDA	Compute Unified Device Architecture
OpenCL	Open Computing Language
OS	Operating System
IPv4	Internet Protocol version 4
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
LAN	Local Area Network
VPN	Virtual Private Network
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language
UML	Unified Modeling Language
WCF	Windows Communication Foundation
MFC	Microsoft Foundation Class
CSV	Comma-Separated Values file format
PDM	Product Data Management software system
ERP	Enterprise Resource Planning software system
UI/GUI	User Interface/Graphical User Interface
IBPV	Initial Boundary Value Problem
FEM	Finite Element Method
FEA	Finite Element Analysis
FE	Finite Element
FETI	Finite Element Tearing and Interconnecting method
CFD	Computational Fluid Dynamics
CM	Computational Mechanics
CCM	Computational Contact Mechanics
OOFEM	Object Oriented Finite Element software tool
SIFEL	SImple Finite ELements software tool
FEXP	Finite [E]lement [E]XPlicit software tool
MEIM	Macro Entity Interaction Multigraph

[2] It contains the list of important or frequently used miscellaneous abbreviations in the text of the dissertation thesis. The abbreviations are always explained in the text, thus the following list is included here primarily for the purpose of better orientation.

Bibliography

[1] Alamatian J.: *A new formulation for fictitious mass of the Dynamic Relaxation method with kinetic damping*, Computers & Structures, Vol. 90-91, pp. 42–54, Elsevier, 2012.

[2] Aliaga J.I., Pérez J., Quintana-Ortí E.S.: *Systematic Fusion of CUDA Kernels for Iterative Sparse Linear System Solvers*, Euro-Par 2015: Parallel Processing, 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, pp. 675-686, Springer-Verlag Berlin Heidelberg, 2015.

[3] Aristotle: *Physics*, Kessinger Publishing, LLC, 2004.

[4] Bedford A.: *Hamiltons Principle in Continuum Mechanics*, John Wiley & Sons, 1986.

[5] Belytschko T., Hsieh B.J.: *Nonlinear Transient Finite Element Analysis with Convected Coordinates*, International Journal for Numerical Methods in Engineering, Vol. 7(3), pp. 255-271, John Wiley & Sons, 1973.

[6] Belytschko T., Lin J.I., Chen-Shyh Tsay: *Explicit algorithms for the nonlinear dynamics of shells*, Computer Methods in Applied Mechanics and Engineering, Vol. 42(2), pp. 225-251, Elsevier, 1984.

[7] Belytschko T., Liu W.K., Moran B., Elkhodary K.: *Nonlinear Finite Elements for Continua and Structures*, John Wiley & Sons, Second Edition, 2014.

[8] Belytschko T., Stolarski H., Carpenter N.: *A C^0 triangular plate element with one-point quadrature*, International Journal for Numerical Methods in Engineering, Vol. 20(5), pp. 787–802, John Wiley & Sons, 1984.

[9] Bentley J.L.: *Multidimensional binary search trees used for associative searching*, Communications of the ACM, Vol. 18(9), pp. 509–517, 1975.

[10] Berg M. de, Cheong O., Kreveld M. van, Overmars M.: *Computational Geometry: Algorithms and Applications*, Springer, Third Edition, 2008.

[11] Berger M.J., Bokhari S.H.: *A partitioning strategy for nonuniform problems on multiprocessors*, IEEE Transactions on Computers, Vol. C-36, No. 5, pp. 570–580, 1987.

[12] Blewett R., Clymer A.: *Pro Asynchronous Programming with .NET*, Apress, 2015.

[13] Bondy J.A., Murty U.S.R.: *Graph Theory With Applications*, Elsevier Science Ltd/North-Holland, 1976.

[14] Borst R.D., Crisfield M.A., Remmers J.J.C, Verhoosel C.V.: *Nonlinear Finite Element Analysis of Solids and Structures*, John Wiley & Sons, Second Edition, 2012.

[15] Borwein J., Zhu Q.: *Techniques of Variational Analysis (CMS Books in Mathematics)*, Springer, 2005.

[16] Bouknight W.J., Denenberg S.A., McIntyre D.E., Randall J.M, Sameh A.H., Slotnick D.L.: *The Illiac IV System*, Proceedings of the IEEE, Vol. 60, No. 4, 1972.

[17] Brenner S., Scott R.: *The Mathematical Theory of Finite Element Methods (Texts in Applied Mathematics)*, Springer, Third Edition, 2007.

[18] Brio M.: *Numerical Time-Dependent Partial Differential Equations for Scientists and Engineers*, Academic Press, 2010.

[19] Buluç A., Gilbert J.R., Budak C.: *Solving path problems on the GPU*, Parallel Computing, Elsevier, 2010.

[20] Courant R., Fredrichs K.O., Lewy H.: *On the partial difference equations of mathematical physics*, IBM Journal of Research and Development, 11 (2): 215–234, 1967.

[21] Day A.S.: *An introduction to dynamic relaxation*, The Engineers, Vol. 219, pp. 218-221, 1965.

[22] Divecchio M.C.: *The Design of the Parallel Arithmetic Unit in PEPE*, University of Pennsylvania Department of Computer and Information Science, Technical Report No. MSCIS-78-22, 1978.

[23] Dostál Z., Horák D., Kučera R.: *Total FETI - an easier implementable variant of the FETI method for numerical solution of elliptic PDE*, Communications in Numerical Methods in Engineering, Vol. 22, No. 6, pp. 1155-1162, 2006.

[24] Du P., Weber R., Luszczek P., Tomov S., Peterson G., Dongarra J.: *From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming*, Parallel Computing, Vol. 38(8), pp. 391–407, Elsevier, 2012.

[25] Dunigan T.H.: *Performance of the Intel iPSC/860 and Ncube 6400 hypercubes*, Parallel Computing, Elsevier, Volume 17, Pages 1285-1302, 1991.

[26] Dziubak T., Matulewski J.: *An object-oriented implementation of a solver of the time-dependent Schrödinger equation using the CUDA technology*, Computer Physics Communications, Vol. 183(3), pp. 800-812, Elsevier, 2012.

[27] Even S.: *Graph Algorithms*, Cambridge University Press, Second Edition, 2011.

[28] Eyerman S., Eeckhout L.: *Modeling critical sections in Amdahl's law and its implications for multicore design*, ISCA '10 Proceedings of the 37th annual international symposium on Computer architecture, pp. 362-370, 2010.

[29] Farhat C., Roux F.X.: *A method of finite element tearing and interconnecting and its parallel solution algorithm*, International Journal for Numerical Methods in Engineering, Vol. 32, No. 6, pp. 1205-1227, John Wiley & Sons, 1991.

[30] Ferland K.: *Discrete Mathematics An Introduction to Proofs and Combinatorics*, Houghton Mifflin Company, 2009.

[31] Ferrando N., Gosálvez M.A., Cerdá J., Gadea R., Sato K.: *Octree-based, GPU implementation of a continuous cellular automaton for the simulation of complex, evolving surfaces*, Computer Physics Communications, Vol. 182(3), pp.628-640, Elsevier, 2011.

[32] Gamow G.: *Thirty Years that Shook Physics: The Story of Quantum Theory*, Dover Publications, Revised ed. edition, 1985.

[33] Gaster B., Howes L., Kaeli D.R., Mistry P., Schaa D.: *Heterogeneous Computing with OpenCL*, Morgan Kaufmann, 2011.

[34] Goldstein H., Poole Ch., Safko J.: *Classical Mechanics: Pearson New International Edition*, Pearson Education, Third Edition, 2001.

[35] Gorobets A.V., Trias F.X., Oliva A.: *A parallel MPI + OpenMP + OpenCL algorithm for hybrid supercomputations of incompressible flows*, Computers & Fluids, Vol. 88, pp. 764–772, Elsevier, 2013.

[36] Gropp W., Lusk E., Skjellum A.: *Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation)*, The MIT Press, Second edition, 1999.

[37] Gruber P., Zeman J., Kruis J., Šejnoha M.: *Homogenization of composites with interfacial debonding using duality-based solver and micromechanics*, Computer Assisted Mechanics and Engineering Sciences, 16(1): 59-76, 2009.

[38] Guan Q.: *Parallel Algorithms for Geographic Processing*, Dissertation, University of California, Santa Barbara, 2008.

[39] Hallquist J.O.: *LS-DYNA Theoretical Manual*, Livermore: Livermore Software Technology Corporation, 1998.

[40] Halphen B., Nguyen Q.S: *Sur les matériaux standards généralisés*, Journal de Mécanique, Vol. 14, pp. 39–63, 1975.

[41] Har J., Tamma K.: *Advances in Computational Dynamics of Particles, Materials and Structures*, John Wiley & Sons, 2012.

[42] Hart J. M.: *Windows System Programming (Addison-Wesley Microsoft Technology Series)*, Addison-Wesley Professional, 2015.

[43] Hertz H.: *Über die Berührung fester elastischer Körper*, Journal für die Reine und Angewandte Mathematik, Vol. 29, pp. 156–171, 1882.

[44] Hillis W.D.: *The Connection Machine (Mit Press Series in Artificial Intelligence)*, The MIT Press, 1986.

[45] Hlaváček I., Haslinger J., Nečas J., Lovíšek J.: *Solution of Variational Inequalities in Mechanics (Applied Mathematical Sciences)*, Springer, 1988.

[46] Holuša L., Kratochvíl J., Křížek M., Marek I. Ženíšek A.: *Miloš Zlámal, zakladatel matematické teorie metody konečných prvků*, VUTIUM, 2006.

[47] Hopcroft J., Tarjan R.: *Algorithm 447: Efficient algorithms for graph manipulation*, Communications of the ACM, Vol. 16(6), pp. 372-378, ACM, 1973.

[48] Hwang K., Dongarra J., Fox G.C.: *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Morgan Kaufmann, 2011.

[49] Chapman S.: *Fortran 95/2003 for Scientists and Engineers*, McGraw-Hill Education, Third Edition, 2007.

[50] Chaves E.W.V: *Notes on Continuum Mechanics (Lecture Notes on Numerical Methods in Engineering and Sciences)*, Springer, 2013.

[51] Chen W., Beister M., Kyriakou Y., Kachelries M.: *High performance median filtering using commodity graphics hardware*, Nuclear Science Symposium Conference Record (NSS/MIC) IEEE, pp. 4142-4147, 2009.

[52] Christensen J., Bastien Ch.: *Nonlinear Optimization of Vehicle Safety Structures: Modeling of Structures Subjected to Large Deformations*, Butterworth-Heinemann, 2015.

[53] Idelsohn S.R., Marti J., Becker P., Oñate E.: *Analysis of multifluid flows with large time steps using the particle finite element method*, International Journal for Numerical Methods in Fluids, Vol. 75, pp. 621–644, John Wiley & Sons, 2014.

[54] Irvine K.R.: *Assembly Language for Intel-Based Computers*, Prentice Hall, 2007.

[55] Jacobsen D.A., Senocak I.: *Multi-level parallelism for incompressible flow computations on GPU clusters*, Parallel Computing, Vol. 39, pp. 1-20, Elsevier, 2013.

[56] Johnsen S.F., Taylor Z.A., Clarkson M.J., et al.: *NiftySim: A GPU-based nonlinear finite element package for simulation of soft tissue biomechanics*, International Journal of Computer Assisted Radiology and Surgery, Vol. 10(7), pp. 1077–1095, Springer, 2015.

[57] Johnson M.W., Amin M.H.S., Gildert S., Lanting T., Hamze F., Dickson N., Harris R., Berkley A.J., Johansson J., Bunyk P., Chapple E.M., Enderud C., Hilton J.P., Karimi K., Ladizinsky E., Ladizinsky N., Oh T., Perminov I., Rich C., Thom M.C., Tolkacheva E., Truncik C.J.S, Uchaikin S., Wang J., Wilson B., at al: *Quantum annealing with manufactured spins*, Nature, Volume 473, pp. 194-198, 2011.

[58] Kennedy J.M., Belytschko T., Lin J.I.: *Recent developments in explicit finite element techniques and their application to reactor structures*, Nuclear Engineering and Design, Vol. 97(1), pp. 1-24, Elsevier, 1986.

[59] Kernighan B.W., Lin S.: *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical Journal, Vol. 49, No. 2, pp. 291–307, 1970.

[60] Kernighan B.W., Ritchie D.M: *The C Programming Language*, Prentice-Hall, 1978.

[61] Khoei A.R.: *Extended Finite Element Method: Theory and Applications (Wiley Series in Computational Mechanics)*, John Wiley & Sons, 2015.

[62] Krause R., Rank E.: *A fast algorithm for point-location in a finite element mesh*, Computing, Vol. 57(1), pp. 49-62, Springer-Verlag, 1996.

[63] Kruis J.: *Domain Decomposition Methods for Distributed Computing (Saxe-Coburg Publications on Computational Engineering)*, Saxe-Coburg Publications, 2007.

[64] Kruis J., Bittnar Z.: *Reinforcement-matrix interaction modeled by FETI method*, Domain Decomposition Methods in Science and Engineering XVII, Lecture Notes in Computational Science and Engineering, Vol. 60, pp. 567-573, Springer Berlin Heidelberg, 2008.

[65] Kruis J., Zeman J., Gruber P.: *Model of Imperfect Interfaces in Composite Materials and Its Numerical Solution by FETI Method*, Lecture Notes in Computational Science and Engineering, Vol. 91, pp. 337-344, 2013.

[66] Kurose J.F., Ross K.W.: *Computer Networking: A Top-Down Approach*, Pearson, Sixth Edition, 2012.

[67] Ladd T.D., Jelezko F., Laflamme R., Nakamura Y., Monroe Ch., O'Brien J.L: *Quantum Computing*, Nature, Volume 464, pp. 45-53, 2010.

[68] Lafontaine N.M., Rossi R., Cervera M. et al.: *Explicit mixed strain-displacement finite element for dynamic geometrically non-linear solid mechanics*, Computational Mechanics, Vol. 55, pp. 543–559, Springer Berlin Heidelberg, 2015.

[69] Lamport L.: *Computer: A History of the Information Machine (The Sloan Technology Series)*, Westview Press, Third Edition, 2013.

[70] Langr D., Tvrdík P., Dytrych T., Draayer J.P.: *Algorithm 947: Paraperm - Parallel Generation of Random Permutations with MPI*, ACM Trans. Math. Softw., Vol. 41(1), pp. 5:1-5:26, 2014.

[71] Langr D., Tvrdík P., Šimeček I., Dytrych T.: *Downsampling Algorithms for Large Sparse Matrices*, International Journal of Parallel Programming, Vol. 43(5), pp. 679–702, Springer, 2015.

[72] Laursen T.A.: *Computational Contact and Impact Mechanics: Fundamentals of Modeling Interfacial Phenomena in Nonlinear Finite Element Analysis*, Springer, 2002.

[73] Lea H.P., Cambierb J.-L., Cole L.K.: *GPU-based flow simulation with detailed chemical kinetics*, Computer Physics Communications, Vol. 184(3), pp. 596–606, Elsevier, 2013.

[74] LeBlanc T.J., Scott M.L., Brown C.M.: *Large-Scale Parallel Programming: Experience with the BBN Butterfly Parallel Processor*, Butterfly Project Report, Computer Science Department, University of Rochester, 1988.

[75] Lefebvre M., Guillen P., Le Gouez J.-M., Basdevant C.: *Optimizing 2D and 3D structured Euler CFD solvers on Graphical Processing Units*, Computers & Fluids, Vol. 70, pp. 136-147, Elsevier, 2012.

[76] Li R., Saad Y.: *GPU-accelerated preconditioned iterative linear solvers*, The Journal of Supercomputing, Vol. 63, pp. 443-466, Springer US, 2013.

[77] Lipschutz S., Lipson M.: *Schaum's Outline of Discrete Mathematics*, McGraw-Hill, Third Edition, 2007.

[78] Liu W., Schmidt B., Voss G., Müller-Wittig W.: *Accelerating molecular dynamics simulations using Graphics Processing Units with CUDA*, Computer Physics Communications, Vol. 179(9), pp. 634–641, Elsevier, 2008.

[79] Lucas É.: *Récreations Mathématiques (Vol. I)*, Paris:Gauthier-Villars, 1882.

[80] Manolopoulos Y., Nanopoulos A., Papadopoulos A.N., Theodoridis Y.: *R-Trees: Theory and Applications*, Advanced Information and Knowledge Processing, Springer, 2006.

[81] Matoušek J., Nešetřil J.: *Kapitoly z diskrétní matematiky*, Karolinum, 2010.

[82] Maugin G.A.: *Continuum Mechanics Through the Twentieth Century: A Concise Historical Perspective (Solid Mechanics and Its Applications)*, Springer, 2013.

[83] Maurer D.: *Energy Expressions in Explicit Finite-Element Methods*, 2003.

[84] Mehlhorn K.: *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, Monographs in Theoretical Computer Science. An EATCS Series, Springer, 1990.

[85] Melenka J.M., Babuška I.: *The partition of unity finite element method: Basic theory and applications*, Computer Methods in Applied Mechanics and Engineering, Vol. 139(1-4), pp. 289-314, Elsevier, 1996.

[86] Mielke A.: *A Mathematical Framework for Generalized Standard Materials in the Rate-Independent Case*, Lecture Notes in Applied and Computational Mechanics, Vol. 28, pp. 399-428, 2006.

[87] Nagel Ch., Glynn J.: *Professional C# 5.0 and .NET 4.5.1*, Wrox, 2014.

[88] Naumov M.: *Parallel Solution of Sparse Triangular Linear Systems in the Preconditioned Iterative Methods on the GPU*, NVIDIA Technical Report NVR-2011-001, NVIDIA, 2011.

[89] Neal M.O., Belytschko T.: *Explicit-explicit subcycling with non-integer time step ratios for structural dynamic systems*, Computers & Structures, Vol. 31(6), pp. 871-880, Elsevier, 1989.

[90] Nečas J.: *Les methodes directes en theorie des equations elliptiques*, Academia, Praha, and Masson et Cie, Editeurs, Paris, 1967.

[91] Němec I., Kolář V., Ševčík I., Vlk Z., Blaauwendraat J., Buček J., Teplý B., Novák D., Štembera V.: *Finite Element Analysis of Structures, Principles and Praxis*, Shaker Verlag, 2010.

[92] Ng S., Leung K.: *Induction of Quadratic Decision Trees using Genetic Algorithms and k-D Trees*, 2003.

[93] Oller S.: *Nonlinear Dynamics of Structures (Lecture Notes on Numerical Methods in Engineering and Sciences)*, Springer, 2015.

[94] Oñate E.: *Structural Analysis with the Finite Element Method. Linear Statics: Volume 1: Basis and Solids (Lecture Notes on Numerical Methods in Engineering and Sciences)*, Springer, 2009.

[95] Oñate E.: *Structural Analysis with the Finite Element Method. Linear Statics: Volume 2: Beams, Plates and Shells (Lecture Notes on Numerical Methods in Engineering and Sciences)*, Springer, 2013.

[96] Oyarzun G., Borrell R., Gorobets A., Oliva A.: *MPI-CUDA sparse matrix–vector multiplication for the conjugate gradient method with an approximate inverse preconditioner*, Computers & Fluids, Vol. 90, pp. 244-252, Elsevier, 2013.

[97] Parsons B., Wilson E.A.: *A Method for Determining the Surface Contact Stresses Resulting From Interference Fits*, Journal of Engineering for Industry, Vol. 92(1), pp. 208-218, 1970.

[98] Patzák B.: *OOFEM - an object-oriented simulation tool for advanced modeling of materials and structures*, Acta Polytechnica, 52(6):59–66, 2012.

[99] Patzák B., Rypl D., Kruis J.: *MuPIF – A distributed multi-physics integration tool*, Advances in Engineering Software, Vol. 60-61, pp. 89-97, Elsevier, 2012.

[100] Perrot G., Domas S., Couturier S.: *Fine-tuned high speed implementation of a gpu-based median filter*, Journal of Signal Processing Systems, pp. 1-6, 2011.

[101] Pickeringa B.P., Jacksona Ch.W., Scoglandb T.R.W., Wu-Chun Fengb, Roya Ch.J.: *Directive-based GPU programming for computational fluid dynamics*, Computers & Fluids, Vol. 114, pp. 242–253, Elsevier, 2015.

[102] Reddy J.N.: *An Introduction to Nonlinear Finite Element Analysis: with applications to heat transfer, fluid mechanics, and solid mechanics*, Oxford University Press, Second Edition, 2015.

[103] Reissner E.: *On transverse bending of plates, including the effect of transverse shear deformation*, International Journal of Solids and Structures, Vol. 11(5), pp. 569-573, Elsevier, 1975.

[104] Rek V., Němec I.: *Parallel Computing Procedure for Dynamic Relaxation Method on GPU Using NVIDIA's CUDA*, Applied Mechanics and Materials, Vol. 821, pp. 331-337, Trans Tech Publications, 2016.

[105] Rek V., Němec I.: *Parallel Computation on Multicore Processors Using Explicit Form of the Finite Element Method and C++ Standard Libraries*, Strojnícky casopis – Journal of Mechanical Engineering, Vol. 66(2), pp. 67-78, De Gruyter, 2016.

[106] Rodriguez J., Rio R., Cadou J.M., Troufflard J.: *Numerical study of dynamic relaxation with kinetic damping applied to inflatable fabric structures with extensions for 3D solid element and non-linear behavior*, Thin-Walled Structures, Vol. 49(11), pp. 1468-1474, Elsevier, 2011.

[107] Rodrígueza M., Blesab F., Barrio R.: *OpenCL parallel integration of ordinary differential equations: Applications in computational dynamics*, Computer Physics Communications, Vol. 192, pp. 228–236, Elsevier, 2015.

[108] Rosen K.H.: *Handbook of Discrete and Combinatorial Mathematics, Second Edition (Discrete Mathematics and Its Applications)*, CRC Press, 2000.

[109] Ross A.: *A Rudimentary History of Dynamics*, Modeling, Identification and Control, Vol. 30(4), pp. 223–235, Norwegian Society of Automatic Control, 2009.

[110] Říha L., Brzobohatý T., Markopoulos A., Jarošová M., Kozubek T.: *Implementation of hybrid total FETI (HTFETI) solver for multi-core architectures*, AIP Conference Proceedings, Vol. 1648, American Institute of Physics, 2015.

[111] Salvadorea F., Bernardinib M., Botti M.: *GPU accelerated flow solver for direct numerical simulation of turbulent flows*, Journal of Computational Physics, Vol. 235, pp. 129–142, Elsevier, 2013.

[112] Samet H.: *Foundations of Multidimensional and Metric Data Structures*, The Morgan Kaufmann Series in Computer Graphics, Morgan Kaufmann, 2006.

[113] Santos A., Teixeira J.M., Farias T., Teichrieb V., Kelner J.: *Understanding the Efficiency of kD-tree Ray-Traversal Techniques over a GPGPU Architecture*, International Journal of Parallel Programming, Vol. 40(3), pp. 331–352, Springer, 2011.

[114] Sedgewick R., Flajolet P.: *An Introduction to the Analysis of Algorithms*, Addison-Wesley Professional, 2013.

[115] Shreiner D., Sellers G., Kessenich J.M., Licea-Kane B.: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3*, Addison-Wesley Professional, Eighth Edition, 2013.

[116] Signorini A.: *Sopra alcune questioni di elastostatica*, Atti della Societa Italiana per il Progresso delle Scienze, 1933.

[117] Simon H.D.: *Parallel computational fluid dynamics: implementations and results*, Scientific and engineering computation, The MIT Press, Cambridge, MA, 1992.

[118] Skiena S.S.: *The Algorithm Design Manual*, Springer, 2008.

[119] Sosutha S., Mohana D.: *Heterogeneous Parallel Computing Using Cuda for Chemical Process*, Procedia Computer Science, Vol. 47, pp. 237-246, Elsevier, 2015.

[120] Souza Neto E.A. de, Peric D., Owen D.R.J.: *Computational Methods for Plasticity: Theory and Applications*, John Wiley & Sons, 2008.

[121] Stallings W.: *Operating Systems: Internals and Design Principles*, Prentice Hall, Sixth Edition, 2008.

[122] Storaasli O.O., Peebles S.W., Crockett T.W., Knott J.D., Adams L.: *The finite element machine: An experiment in parallel processing*, Technical Report, NASA Langley Research Center; Hampton, VA, United States, 1982.

[123] Strömberg N., Johansson L., Klarbring A.: *Derivation and analysis of a generalized standard model for contact, friction and wear*, International Journal of Solids and Structures, Vol. 33, pp. 1817-1836, 1996.

[124] Tabarrok C., Rimrott F.P.: *Variational Methods and Complementary Formulations in Dynamics (Solid Mechanics and Its Applications)*, Springer, 1994.

[125] Vallée C., Lerintiu C., Fortuné D., Atchonouglo K, Ban M.: *Representing a non-associated constitutive law by a bipotential issued from a Fitzpatrick sequence*, Archives of Mechanics, Vol. 61, pp. 325-340, 2009.

[126] Walshaw C., Cross M.: *JOSTLE: parallel multilevel graph-partitioning software—an overview*, Mesh Partitioning Techniques and Domain Decomposition Techniques, pp. 27–58, Civil-Comp Press, 2007.

[127] Wang Y.-X, Zhang L.-L., Liu W., Che Y.-G., Xu C.-F., Wang Z.-H., Zhuang Y.: *Efficient parallel implementation of large scale 3D structured grid CFD applications on the Tianhe-1A supercomputer*, Computers & Fluids, Vol. 80, pp. 244-250, Elsevier, 2013.

[128] Wei Z., Jang B., Jia Y: *A fast and interactive heat conduction simulator on GPUs*, Procedia Computer Science, Vol. 270, pp. 496-505, Elsevier, 2014.

[129] Weller R.: *New Geometric Data Structures for Collision Detection and Haptics*, Springer Series on Touch and Haptic Systems, Springer, 2013.

[130] Wiener N.: *Cybernetics: or control and communication in the animal and the machine*, The MIT Press, Cambridge, MA, 1948.

[131] Wilson E.A., Parsons B.: *Finite element analysis of elastic contact problems using differential displacement*, International Journal for Numerical Methods in Engineering, Vol. 2(3), pp. 387-395, John Wiley & Sons, 1970.

[132] Wriggers P.: *Computational Contact Mechanics*, Springer, Second Edition, 2006.

[133] Wriggers P.: *Nonlinear Finite Element Methods*, Springer, 2008.

[134] Wriggers P., Zavarise G.: *A formulation for frictionless contact problems using a weak form introduced by Nitsche*, Computational Mechanics, Vol. 41(3), pp. 407–420, Springer, 2008.

[135] Wright R.S., Haemel N., Sellers G., Lipchak B.: *OpenGL SuperBible: Comprehensive Tutorial and Reference*, Addison-Wesley Professional, Fifth Edition, 2010.

[136] Wu S.R., Gu L.: *Introduction to the Explicit Finite Element Method for Nonlinear Transient Dynamics*, John Wiley & Sons, 2012.

[137] Zeman J., Gruber P.: *Numerical approach to a rate-independent model of decohesion in laminated composites*, Programs and Algorithms of Numerical Mathematics, Proceedings of Seminar, pp. 239-250, Institute of Mathematics AS CR, 2010.

[138] Zhou K., Hou R.Q., Wang B., Guo B.: *Real-Time KD-Tree Construction on Graphics Hardware*, ACM Transactions on Graphics (TOG), Vol. 27, pp. 1-11, 2008.

[139] Zlámal M.: *On the finite element method*, Numerische Mathematik, Vol. 12(5), pp. 394-409, Springer, 1968.

List of Tables

List of Figures

Attachments

1. Files related to testing example of numerical simulation:
   - Structural models from RFEM program:
     a) *rfem_sphere_m_1.rf5*
     b) *rfem_sphere_m_2.rf5*
     c) *rfem_sphere_m_3.rf5*
     d) *rfem_sphere_m_4.rf5*
   - Input data files for FEXP solver:
     a) *sphere_m_1.fexin*
     b) *sphere_m_2.fexin*
     c) *sphere_m_3.fexin*
     d) *sphere_m_4.fexin*
     e) *sphere_default.fexin*
     f) *cnfg_fexp_solver.fexcfg*
   - Folder with result files for Paraview program: *RESULTS*.
   - Videos related to dynamic simulation:
     a) *FEXP_simulation_spheres_view_1.mp4*
     b) *FEXP_simulation_spheres_view_2.mp4*
     c) *FEXP_simulation_spheres_view_3.mp4*
     d) *FEXP_running_network_solver___localhost.mp4*
2. Compiled programs:
   - *FEXPSolver.exe*, *FEXPNetServer.exe*, *FEXPNetClient.exe*
   - *FEXPSolverManager.exe*, *FEXPSolverNetworkClientManager.exe*, *FEXPEnterpriseCommonLibrary.dll*
3. Software documentation of *FEXP Solver* and *FEXP Solver Manager*.