Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Využití GPU v programu Gromacs

Diplomová práce

Bc. Tomáš Krejsa

Školitel: Doc. RNDr. Milan Předota, Ph.D.

České Budějovice 2015

KREJSA, T., 2015: Využití GPU v programu Gromacs. [Utilization of GPU in Gromacs. Mgr. Thesis, in Czech.] – 50 p., Faculty of Science, University of South Bohemia,
České Budějovice, Czech Republic.

Anotace:
Appropriate GPU, other hardware and software is chosen to utilize GPU in Gromacs.
Gromacs is one of the fastest molecular dynamics software engine available today. It is aimed at highly efficient use of a hardware. Computing on GPU is supported from Gromacs 4.6. Gromacs performance with GPU is tested against CPU only. Moreover, two different systems and also some options in mdrun command are tested. Additionaly, a small program for reading xtc binary file is created. The main purpose of this program is easier analysis of trajectories.

Prohlašuji, že svoji diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

České Budějovice, 14. 12. 2015.

Děkuji doc. RNDr. Milanu Předotovi, Ph.D. za ochotu, trpělivost, cenné rady, připomínky a za čas, který mi věnoval při vedení diplomové práce.

Obsah

1. Úvod

Motivací pro vytvoření této diplomové práce byla příležitost zrychlit výpočty simulací v programu pro molekulární simulace Gromacs, protože tento program podporuje grafické karty od společnosti NVIDIA při svých výpočtech. Snaha prokázat efektivitu výpočtů na GPU tu již na fakultě byla a v klastru MP je jedna ne plně využitá grafická karta GeForce 560 Ti. Je tedy snaha o využití zakoupených grafických karet na výpočtech molekulárního modelování, které fyzici denně využívají. Tato diplomová práce potvrdí či vyvrátí efektivitu využití GPU i při takto obecných a složitých výpočtech. Dále bude vytvořena aplikace pro čtení *.xtc* souboru, která bude sloužit pro dodatečné analýzy prováděné fyziky.

2. Volba grafických karet

Úkol byl vybrat nejlepší variantu grafické karty pro výpočty na grafických kartách a výpočty simulací v programu Gromacs. Proběhl tedy průzkum trhu s aktuálně nabízenými grafickými kartami. Grafická karta byla vybrána a navrhnuta k zakoupení do klastru MP. Pro tuto diplomovou práci byla zakoupena pouze jedna grafická karta. Později, až se ukáže efektivita výpočtů na grafických kartách v programu Gromacs, budou zakoupeny další grafické karty dle finančních možností.

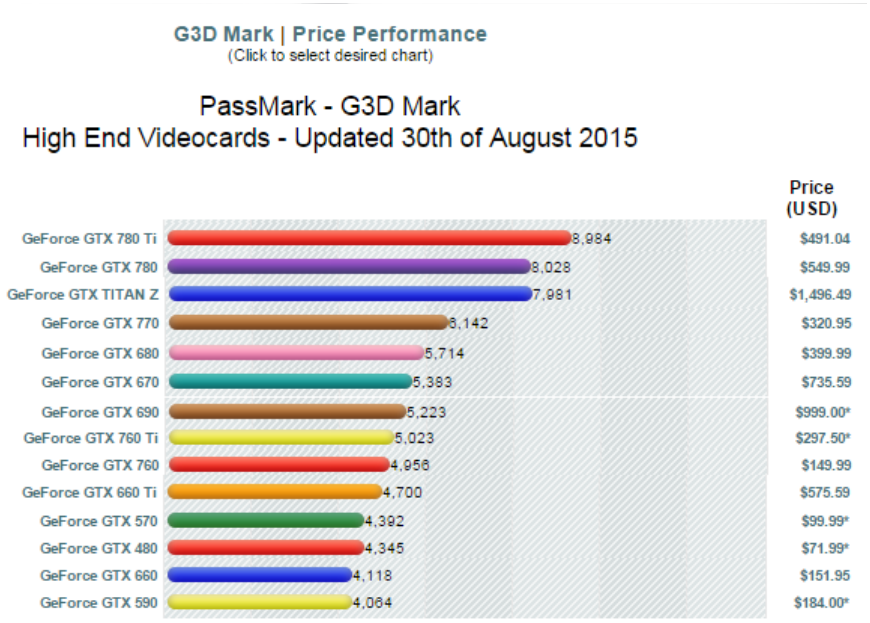
2.1. Výběr grafické karty

Pokud má být excelentní nativní GPU podpora programu Gromacs, je vyžadována alespoň CUDA development kit verze 4.0 od společnosti NVIDIA. Nezbytností je také GPU s NVIDIA compute capability 2.0. To jsou grafické karty s architekturou Fermi nebo Kepler. Z toho tedy vyplývá, že Gromacs, ve kterém fyzici provádí své simulace modelů, nativně podporuje pouze grafické karty od společnosti NVIDIA. Proto je vybírán GPU od společnosti NVIDIA, která má sloužit právě k těmto Gromacs výpočtům a urychlit je tak. [7]

U vybíraných GPU byly sledovány parametry:

- Počet CUDA jader
- Base Clock (MHz)
- Boost Clock (MHz)
- Memory Speed
- Memory Bandwidth (GB/sec)
- Standard Memory Config – velikost paměti (GB)
- Memory Interface Width
- Other Supported Technologies
- Bus Support
- Height, Length, Width
- Graphics Card Power (W) a Minimum Recommended System Power (W)
- Supplementary Power Connectors

GPU byla vybrána porovnáváním benchmarků ze stránek https://compubench.com/, http://www.videocardbenchmark.net/, http://www.geforce.co.uk/hardware/desktop-gpus/geforce-gtx-780/performance. Ze stránky http://www.geforce.co.uk/hardware/desktop-gpus/geforce-gtx-780/performance do obrázku 1 byly vybrány pouze některé GPU od společnosti NVIDIA a to pro porovnání výkonnosti tehdy dostupných grafických karet. Úplně byla vynechána například série grafických karet 900, která je již dnes na vedoucích příčkách těchto benchmarků, protože tato série grafických karet nebyla v roce 2014 ještě vydána. [2, 3, 8]



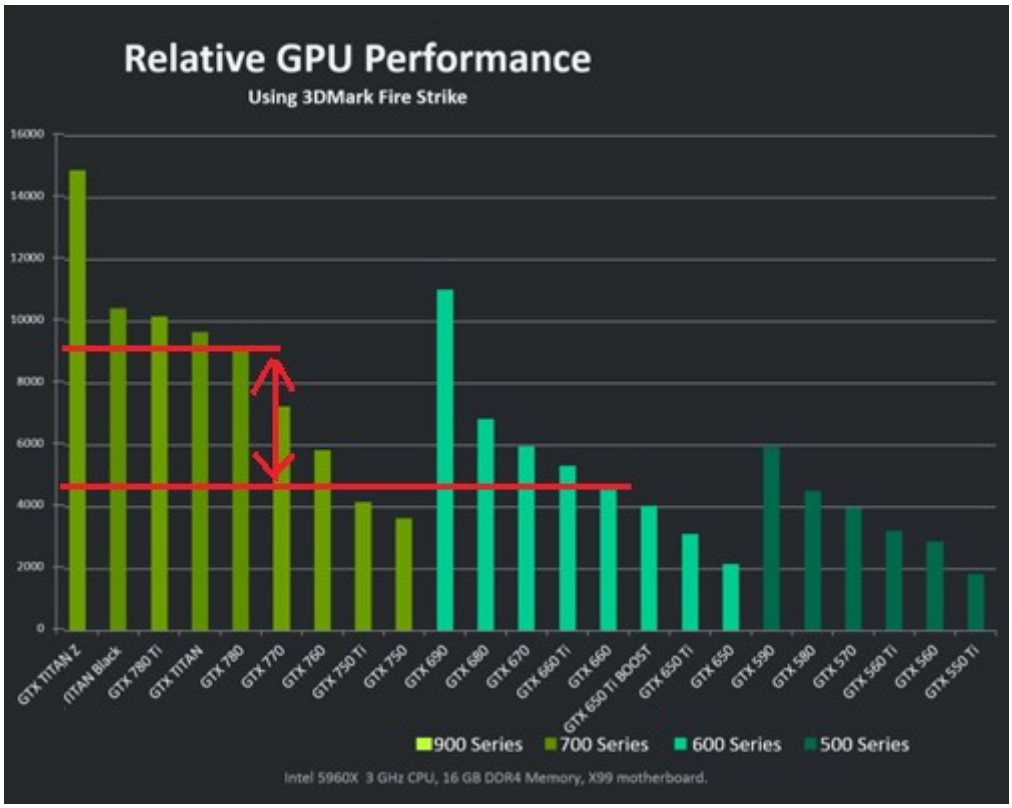
Obrázek 1: Přehled vybraných NVIDIA GPU [2]

Zásadním benchmarkem byl https://compubench.com/ (Tabulka 1). K porovnání výkonnosti grafických karet byl použit test Physics: Particle Simulation. Tento test se již blíží simulacím, které fyzici provádějí. [8]

CompuBenchmark		Particle Simulation – 64k (mInteraction/s	Poznámka
GPU	NVIDIA GeForce GTX TITAN Z	1129.268	Windows, OpenCL
	NVIDIA GeForce GTX 780 TI	673.247	OS X, OpenCL
	NVIDIA GeForce GTX 780	612.33	OS X, OpenCL
	NVIDIA GeForce GTX 690	452.554	Windows, OpenCL
	NVIDIA GeForce GTX 770	440.967	OS X, OpenCL
	NVIDIA GeForce GTX 680	431.992	OS X, OpenCL
	NVIDIA GeForce GTX 750 Ti	390.719	OS X, OpenCL
	NVIDIA GeForce GTX 670	371.024	OS X, OpenCL
	NVIDIA GeForce GTX 760	354.66	OS X, OpenCL
	NVIDIA GeForce GTX 750	348.908	OS X, OpenCL
	NVIDIA GeForce GTX 660	336.042	OS X, OpenCL
	NVIDIA GeForce GTX 660 Ti	324.6	OS X, OpenCL
	NVIDIA GeForce GTX 590	289.713	Windows, OpenCL
	GeForce GTX 650 Ti BOOST	281.974	OS X, OpenCL
	GeForce GTX 650 Ti	209.439	OS X, OpenCL
	GeForce GTX 650	150.3	OS X, OpenCL

Tabulka 1: Výběr GPU z compubench.com [8]

Třetím benchmarkem byl na stránce výrobce grafických karet. Na tomto benchmarku je vidět jak si vedou grafické karty GeForce mezi sebou a i v rámci různých sérií (http://www.geforce.co.uk/hardware/desktop-gpus/geforce-gtx-780/performance). [3]



Obrázek 2: Srovnání NVIDIA GPU serií [3]

Byly vybrány dvě grafické karty a tedy i dvě možnosti zakoupení. Jedna varianta GPU byla vybírána z hlediska co možná největší kompatibility se stávajícími MP počítači a druhá varianta jako nejlepší pro nás dostupná grafická karta. Možnost zakoupit grafickou kartu GeForce GTX 660 by byla výhodná z hlediska kompatibility stávajícího HW v klastrovně MP (grafické kartě by postačoval stávající zdroj, který má výkon 400 W a 1 konektor 6-pin pro PCI-E). Avšak výkon, jak ukazují benchmarky, by byl mnohem menší než u GeForce GTX 780, která byla vybrána jako druhá varianta grafické karty. Žádná výkonnější grafická karta než je GeForce GTX 660, která je na obrázku 2, není kompatibilní se stávajícími zdroji v MP klastrovně.
Rozpočty byly tedy zhotoveny pro obě varianty grafických karet. Protože finance stačili i na zakoupení GeForce GTX 780 a její výkon byl vítán, bylo rozhodnuto zakoupit právě ji.

Zde je porovnání vítězů dvou grafických karet:

CPU Intel Core i7-4790K	GeForce GTX 780	GeForce GTX 660
Počet CUDA jader	2304	960
Base Clock (MHz)	1019	980
Boost Clock (MHz)	1071	1033
Memory Speed	6.0 Gbps	6.0 Gbps
Standard Memory Config – velikost paměti	3072 MB	2048 MB
Memory Bandwidth (GB/sec)	288.4	144.2
Memory Interface Width	384-bit	192-bit
Other Supported Technologies	CUDA	CUDA
Bus Support	PCI Express 3.0	PCI Express 3.0
Height	4.376 inches	4.376 inches
Length	10.5 inches	9.5 inches
Width	Dual-slot	Dual-slot
Graphics Card Power (W)	250 W	140 W
Minimum Recommended System Power (W	600 W	450 W
Supplementary Power Connectors	Two 8-pin	One 6-pin

Tabulka 2: Porovnání GeForce GTX 780 a GeForce GTX 660 [1]

Cílem tedy bude porovnat efektivitu zakoupené grafické karty GeForce GTX 780 a to jak oproti profesionálním grafickým kartám na MetaCentru, tak oproti starší grafické kartě na počítači MP-8. Samozřejmě bude také porovnán výkon s využitím GPU oproti výkonu samotného CPU.

2.2. Výběr hardwaru

Grafická karta GeForce 780, která byla vybrána, má sběrnici PCI-E 3.0, ale stávající počítače v klastrovně MP mají pouze PCI-E 1.1. Z toho důvodu by stávající hardware nevyužil potenciál grafické karty, která byla vybrána. PCI sběrnice by se stala úzkým hrdlem komunikace s GPU. Navíc grafická karta se do stávajících strojů MP nevejde a je počítáno zakoupit i výkonnější zdroj pro novou grafickou kartu. Nakonec tedy byl navrhnut a zakoupen celý počítač, který odpovídá možnostem naší nové GPU. Návrh počítače byl z velké části inspirován PC sestavou „Pro Gaming 5“, která byla nabízena na stránkách czc.cz. V této sestavě byla nabízena právě grafická karta GeForce GTX 780. Protože sestava již byla dobře vyvážená, byla změna pouze ve zdroji, HDD a paměti RAM.

Parametry klíčových PC komponent:

Zdroj Fortron Aurum 92+ 650:

Zdroj Fortron Aurum 92+ 650	
Parametr	Hodnota
Výkon	650 W
Formát	ATX
Výbava a funkce	Odpojitelné kabely, Aktivní PFC, Tepelná regulace otáček, Síťový vypínač
Certifikace	80 PLUS Platinum
Účinnost	92,00%
Konektory	
Konektory pro základní desku	ATX 24-pin, EPS 8-pin
Počet PCI Express 8-pin	4x
Počet Seriál ATA 15-pin	9x
Počet Molex HDD 4-pin	4x
Počet Molex FDD 4-pin	1x

Tabulka 3: Zdroj Fortron Autum 92+ 650

CPU Intel Core i7-4790K:

CPU Intel Core i7-4790K	
Parametr	Hodnota
Řada procesoru	Intel Core i7
Socket	Intel Socket 1150
Počet jader procesoru	4x
Frekvence procesoru	4000 MHz (4 GHz)
Maximální frekvence	4400 MHz (4,4 GHz)
Pokročilé parametry	
Funkce	Automatické přetaktování, Virtualizace, HyperThreading, Chladič v balení
TDP procesoru	88 W
Výrobní technologie	22 nm
L3 cache	8 MB
Integrovaná grafická karta	
Typ integrované grafické karty	Intel HD Graphics 4600
Frekvence	1250 Hz

Tabulka 4: CPU Intel Core i7-4790K

HDD – velikost pevného disku 1 TB.

Základní deska GIGABYTE GA-Z97X-Gaming 5:

Základní deska GIGABYTE GA-Z97X-Gaming 5	
Parametr	Hodnota
Dual BIOS	ano
DVI výstup	ano
Externí porty USB 2.0	4
Externí porty USB 3.0	4
Formát	ATX
HDMI výstup	ano
Chipset	Intel Z97
Max. velikost paměti [GB]	32
Patice (socket procesoru)	1150
PCI Express x1	3
PCI Express x16	3
PCI slot	1
PCI-Express 3.0	ano
Počet konektorů M.2	1
Počet paměťových slotů	4
Počet portů RJ-45	1
Počet portů SATA Express	1
Počet portů USB 2.0	8
Počet portů USB 3.0	6
Podporované frekvence paměti [MHz]	1333, 1600, 1800, 1866, 2000, 2133, 2200, 2400, 2500, 2600, 2666, 2800, 2933, 3000, 3100, 3200
RAID	0, 10, 1, 5
Režim zapojení pamětí	Dual-channel
Řadič M.2	ano
Řadič RAID	ano
Řadič SATA Express	ano
Řadič USB 3.0	ano
SATA III konektory (6 Gb/s)	6
Síťová karta - rychlost [Mbps]	10/100/1000
Šířka [mm]	225
Výška [mm]	305
Typ paměti	DDR3
Typ síťové karty	Qualcomm Atheros Killer E2201
Typ zvukové karty	Realtek ALC1150
UEFI BIOS	ano
VGA výstup	ano
Stránky výrobce	www.gigabyte.com
Stránky o produktu	ɔ://www.gigabyte.cz/products/page/mb/ga-z97x-gaming_5rev_

Tabulka 5: Základní deska GIGABYTE GA-Z97X-Gaming 5

Paměť RAM Corsair 8GB KIT DDR3 1600MHz CL8 Red Vengeance:

Paměť RAM Corsair 8GB KIT DDR3	
Parametr	Hodnota
Typ paměti	DDR3
Kapacita paměti	8 GB
Počet modulů v balení	2 ks
Kit vhodný pro	Dual-channel
Frekvence a časování	
Frekvence paměti	1600 MHz
Časování	CL8
Pokročilé parametry	
Napětí	1,5 V
Další vlastnosti	Pasivní chladič, XMP

Tabulka 6: Paměť RAM Corsair 8GB KIT DDR3 1600MHz CL8

Vzhledem k extra nákladům na HW, byla k HW zakoupena pouze jedna grafická karta GeForce GTX 780. Celkový rozpočet kompletního PC + GeForce GTX 780 byl 30 680,-.

2.3. Instalace softwaru

2.3.1. Instalace operačního systému

Po zakoupení počítače bylo postupně zkušebně instalováno několik operačních systémů linux. Z vyzkoušených operačních systémů byl pouze OpenSUSE 13.2 bezproblémový a je také podporovaný programem Gromacs. OpenSUSE 13.2 byl tedy vybrán do klastru MP, kde již tyto operační systémy na stávajících počítačích jsou nainstalovány.

Nejdříve byl instalován Debian 7, kde při instalaci operačního systému nebyly nalezeny ovladače na síťovou kartu.

Druhým operačním systémem byl OpenSuse 13.1, u kterého instalace proběhla v pořádku. Ovšem kvůli černé obrazovce se nebylo možné zalogovat. Černá obrazovka se objevovala jak v terminálovém, tak i grafickém okně. Problém s černou obrazovkou byl vyřešen až u operačního systému OpenSuse 13.2.

Dalším byl Debian 8, který uprostřed instalace nahlásil chybu a instalace „zamrzla“.

OpenSuse 13.2 byl nainstalován jako poslední a konečný operační systém. Problém s černou obrazovkou při přihlašování do operačního systému byl vyřešen. Integrovanou kartu bylo potřeba zakázat v BIOSu. Po restartu počítače si dedikovanou grafickou kartu počítač již správně našel.

2.3.2. Instalace GPU driveru

Systém OpenSUSE 13.2 nainstaloval své ovladače grafické karty s názvem nouveau, které jsou bezplatně dostupné. Je tedy potřeba nainstalovat ovladače od výrobce. Ze stránek GeForce (http://www.geforce.com/drivers) byly staženy ovladače pro grafickou kartu GTX 780. Stažené

ovladače byly nainstalovány pomocí grafického módu, který ovladače poskytovaly. [26]

```
$ /sbin/init 3     #prepnuti se na konzolovou obrazovku
$ sh /home/suseuser/Download/NVIDIA-Linux-x86_64-346.47.run       #spuštění grafického módu instalace
```

Nyní se zobrazil průvodce, kterého instalátor poskytuje. Pomocí něho byly ovladače nainstalovány. Dalším krokem bylo vrátit se do grafického módu operačního systému

```
$ /sbin/init 5     #zapnutí grafického módu OS
```

Ověření, že ovladače od NVIDIA jsou správně nainstalovány:

```
$ /sbin/lspci -nnk | grep -i vga -A3
01:00.0 VGA compatible controller [0300]: NVIDIA Corporation GK110 [GeForce GTX 780] [10de:1004] (rev a1)
        Subsystem: Gigabyte Technology Co., Ltd Device [1458:3624]
        Kernel driver in use: nvidia
        Kernel modules: nouveau, nvidia
```

2.3.3. Instalace Cuda

Na následující webové stránce byly nalezeny prerekvizity pro instalaci Cuda 7.0 a návod na instalaci http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/index.html#verify-you-have-cuda-enabled-system. [23]
Dle návodu je potřeba zkontrolovat prerekvizity před samotnou instalací CUDA Toolkit [23]:

1. Ověření, že GPU je kompatibilní s CUDA

Výpis informací o grafické kartě následujícím příkazem:

```
$ /sbin/lspci | grep -i nvidia
01:00.0 VGA compatible controller: NVIDIA Corporation GK110 [GeForce GTX 780] (rev a1)
01:00.1 Audio device: NVIDIA Corporation GK110 HDMI Audio (rev a1)
```

Na stránkách http://developer.nvidia.com/cuda-gpus jsou uvedeny grafické karty podporující CUDA a GeForce GTX 780 tam také byla nalezena. CUDA Toolkit tedy podporuje GPU v novém PC.

2. Ověření, že CUDA podporuje tuto verzi linuxu

Podporu OpenSUSE 13.2 s CUDA Toolkit zjistíme na následující stránce http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/index.html#verify-you-have-cuda-enabled-system. V tabulce byl nalezen náš operační systém i s požadavky [23]:

Distribuce	Kernel	GCC	GLIBC	ICC	PGI	XLC
OpenSUSE 13.2	3.16.6	4.8.3	2.19	15.0.0	>=14.9	NO

Je-li potřeba, aktuálně používaný OS zjistíme příkazem:

```
$ uname -m && cat /etc/*release
x86_64 #x86_64 indikuje 64-bitový systém
NAME=openSUSE
```

```
VERSION="13.2 (Harlequin)"
VERSION_ID="13.2"
PRETTY_NAME="openSUSE 13.2 (Harlequin) (x86_64)"
ID=opensuse
ANSI_COLOR="0;32"
CPE_NAME="cpe:/o:opensuse:opensuse:13.2"
BUG_REPORT_URL="https://bugs.opensuse.org"
HOME_URL="https://opensuse.org/"
ID_LIKE="suse"
openSUSE 13.2 (x86_64)
VERSION = 13.2
CODENAME = Harlequin
# /etc/SuSE-release is deprecated and will be removed in the future, use /etc/os-release instead
```

3. Ověření, zda je v systému nainstalovaný gcc kompilátor

```
$ gcc --version
gcc (SUSE Linux) 4.8.3 20140627 [gcc-4_8-branch revision 212064]
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

4. Stažení NVIDIA CUDA Toolkit

Na stránkách INVIDIA byl stažen repozitář CUDA 7.0 a následně z něj CUDA nainstalována. Link na repozitář k OS OpenSUSE 13.2:
http://developer.download.nvidia.com/compute/cuda/7_0/Prod/local_installers/rpmdeb/cuda-repo-opensuse132-7-0-local-7.0-28.x86_64.rpm

5. Akce, které je nutné provést po instalaci CUDA Toolkit

Po dokončení instalace CUDA Toolkit je nutné nastavit proměnné pro prostředí CUDA. Aby byly proměnné nastavené trvale, byly zapsány na konec souboru */etc/profile*.

```
$ vim /etc/profile
export PATH=/usr/local/cuda-7.0/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-7.0/lib64:$LD_LIBRARY_PATH
```

Nyní jsou nastaveny proměnné a nastal čas ověřit, že je CUDA Toolkit schopná komunikovat s grafickou kartou. To lze vyzkoušet zkompilováním a spuštěním hotových příkladů, na kterých bude ověřena správná funkčnost. [23]
Příklady v adresáři */usr/local/cuda-7.0/samples/* jsou pouze read-only a tak budou příklady nainstalovány s právy zapisování. O pohodlnou instalaci příkladů se postará skript [23]:

```
$ cuda-install-samples-7.0.sh /home/krejsa/gromacsVypocty/
```

V adresáři */home/krejsa/gromacsVypocty/* byl vytvořen nový adresář *NVIDIA_CUDA-7.0_Samples*. [23]

Ověření, že systém ovladač nalezne:

```
$ cat /proc/driver/nvidia/version
```

```
NVRM version: NVIDIA UNIX x86_64 Kernel Module  346.47  Thu Feb 19 18:56:03 PST 2015
GCC version:  gcc version 4.8.3 20140627 [gcc-4_8-branch revision 212064] (SUSE Linux)
```

Následuje kompilace příkladu:
Verze CUDA Toolkitu může být ověřena příkazem *nvcc -V.*
Příkaz *nvcc* volá GCC kompilátor na zdrojové kódy určené pro *host* (část programu vykonávaná na CPU) a NVIDIA PTX kompilátor volá na zdrojové kódy určené pro *kernel* (GPU). [23]

Kompilace příkladů:

```
$ cd /home/krejsa/gromacsvypocty/NVIDIA_CUDA-7.0_Samples
$ make
```

Spuštění binárního souboru příkladu *deviceQuery*:

```
$ cd /home/krejsa/gromacsvypocty/NVIDIA_CUDA-7.0_Samples/1_Utilities/deviceQuery
$ ./deviceQuery           #spuštění binárního souboru
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 780"
  CUDA Driver Version / Runtime Version          7.0 / 7.0
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 3072 MBytes (3220897792 bytes)
  (12) Multiprocessors, (192) CUDA Cores/MP:     2304 CUDA Cores
  GPU Max Clock rate:                            1072 MHz (1.07 GHz)
  Memory Clock rate:                             3004 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 1572864 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536),
3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
```

```
 Alignment requirement for Surfaces:            Yes
 Device has ECC support:                        Disabled
 Device supports Unified Addressing (UVA):      Yes
 Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
 Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simu
ltaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 7.0, CUDA Runtime Versi
on = 7.0, NumDevs = 1, Device0 = GeForce GTX 780
Result = PASS
```

V tomto příkladu je možné se dozvědět mnoho užitečných informací, např. velikost konstantní paměti, sdílené paměti v bloku nebo počet registrů. [23]

Druhým příkladem je spuštění bandwidthTest a ujištění se, že systém a zařízení kompatibilní s CUDA (v tomto případě grafická karta) správně komunikují. [23]

```
$ cd /home/krejsa/gromacsvypocty/NVIDIA_CUDA-7.0_Samples/1_Utilities/bandwidthTest
$ ./bandwidthTest
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce GTX 780
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     12098.8

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     12422.3

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     223532.0

Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary
when GPU Boost is enabled.
```

Podle webových stánek, např. http://www.trentonsystems.com/, bandwidth PCI-E 3.0 (bandwidth = množství dat, která mohou být přenesena za určitý čas) je 15.754 GB/s (126.032 Gbit/s) v jednom směru. V tomto testu se ověřila rychlost jednosměrného transferu. Z hostu do GPU je rychlost 11.8152 GB/s resp. z GPU do hostu 12.1312 GB/s. Z toho vyplývá, že systém s GPU komunikuje správně. [9, 10]

2.3.4. Instalace Open MPI

Stažení a instalace Open MPI 1.8.5:

```
$ cd /opt
$ wget http://www.open-mpi.org/software/ompi/v1.8/downloads/openmpi-1.8.5.tar.gz
$ tar -xvf openmpi-1.8.5.tar.gz
$ cd openmpi-1.8.5
$ ./configure --prefix=/opt/openmpi/1.8.5
$ make all install
```

Vždy před použitím Open MPI je nutné exportovat proměnné ručně nebo pomocí vlastního skriptu, který bude ukázán později. Exportované proměnné jsou tyto:

```
$ export PATH=$PATH:/opt/openmpi/1.8.5/bin
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/openmpi/1.8.5/lib64
```

2.3.5. Instalace ICC kompilátoru

Vzhledem k dosavadním výborným zkušenostem s rychlostí výpočtů kompilovaných pomocí Intel C Compiler (ICC), byl tento kompilátor instalován. Pro získání ICC kompilátoru bylo nutné se zaregistrovat na stránkách https://software.intel.com a poté stáhnout *parallel_studio_xe_2015_update3.tgz* do adresáře */opt*. [5]
Stažený soubor byl rozbalen a nainstalován pomocí *install_GUI.sh*:

```
$ cd /opt
$ tar -zxvf parallel_studio_xe_2015_update3.tgz
$ cd parallel_studio_xe_2015_update3
$ ./install_GUI.sh
```

Příkazem *./install_GUI.sh* byl spuštěn průvodce instalací, pomocí něhož bylo nainstalováno celé Parallel studio XE 2015 do adresáře */opt/intel/*, které obsahuje ICC kompilátor.
Pro používání ICC kompilátoru je také nutné spustit soubor, který nastaví proměnné kompilátoru. *Iccvars.sh* soubor lze umístit i do ~/.bashrc nebo jiného systémového login souboru. V tomto případě bylo nastavení proměnných vloženo na konec souboru */etc/profile* [5]:

```
source /opt/intel/bin/iccvars.sh intel64
```

2.3.6. Instalace CMake

CMake je software, který umožňuje automatizaci překladu programu v různých operačních systémech. Jeho výstupem mohou být projekty pro Eclipse, Microsoft Visual Studio, MinGW nebo makefile.txt pro Unixový program make a další. Gromacs používá CMake build systém. Zároveň překlad programu v průvodci instalací Gromacsu je popsán pomocí CMake. Proto bude nainstalován následujícím postupem [17, 18]:

```
$ cd /opt
$ wget http://www.cmake.org/files/v3.3/cmake-3.3.0-rc3-Linux-x86_64.tar.gz
$ tar -xvf cmake-3.3.0-rc3-Linux-x86_64.tar.gz
```

Vložením řádku export *PATH* na konec souboru */etc/profile*, se nastaví cesta na CMake:

```
$ vim /etc/profile
export PATH=$PATH:/opt/cmake-3.3.0-rc3-Linux-x86_64/bin
```

2.4. Instalace Gromacsu a jeho spouštění

2.4.1. Kompilace a příprava Gromacsu

Ke kompilaci Gromacs-5.0.5 je zapotřebí nový CMake a Open MPI, jejichž instalace byly v předchozí kapitole popsány.
Než se začne samotný Gromacs instalovat, je nutné jej stáhnout na stránkách http://www.gromacs.org/. Samotná instalace se řídí webovými stránkami programu Gromacs (http://www.gromacs.org/Documentation/Installation_Instructions_5.0#configuring-with-cmake), popisem instalace programu Gromacs na MP1 – MP8 doktora Fibicha a dalšími nashromážděnými informacemi [7]:

```
#Install Gromacs (pomocí gcc kompilátoru)
$ cd /opt
$ wget ftp://ftp.gromacs.org/pub/gromacs/gromacs-5.0.5.tar.gz
$ tar -xvf gromacs-5.0.5.tar.gz
$ cd gromacs-5.0.5

$ mkdir build
$ cd build

###
# single precision + GPU
###
$ cmake .. -DGMX_GPU=ON -DGMX_BUILD_OWN_FFTW=ON
-DREGRESSIONTEST_DOWNLOAD=ON -DCMAKE_INSTALL_PREFIX=/opt/gromacs-
5.0.5/bin
$ make
$ make check
$ make install

# mpi
export PATH=/opt/openmpi/1.8.5/bin:$PATH
export LD_LIBRARY_PATH=/opt/openmpi/1.8.5/lib64:$LD_LIBRARY_PATH
$ cmake .. -DGMX_GPU=ON -DGMX_BUILD_OWN_FFTW=ON -DGMX_MPI=ON
-DREGRESSIONTEST_DOWNLOAD=ON -DCMAKE_INSTALL_PREFIX=/opt/gromacs-
5.0.5/bin
$ make
$ make check
$ make install

###
# double precision – GPU double precision není podporován
```

```
###
$ cmake .. -DGMX_BUILD_OWN_FFTW=ON -DGMX_GPU=OFF -DGMX_MPI=OFF
-DGMX_DOUBLE=ON -DREGRESSIONTEST_DOWNLOAD=ON
-DCMAKE_INSTALL_PREFIX=/opt/gromacs-5.0.5/bin_double
$ make
$ make check
$ make install

# mpi
$ cmake .. -DGMX_BUILD_OWN_FFTW=ON -DGMX_GPU=OFF -DGMX_DOUBLE=ON
-DGMX_MPI=ON -DREGRESSIONTEST_DOWNLOAD=ON
-DCMAKE_INSTALL_PREFIX=/opt/gromacs-5.0.5/bin_double
$ make
$ make check
$ make install

###
# Gromacs nastavení prostředí a testy 5.0.5 navíc
###
source /opt/gromacs-5.0.5/bin/bin/GMXRC
export PATH=/opt/openmpi/1.8.5/bin:$PATH
export LD_LIBRARY_PATH=/opt/openmpi/1.8.5/lib64:$LD_LIBRARY_PATH
# testy
$ cd /opt
$ wget --no-check-certificate http://gerrit.gromacs.org/download/regressiontests-5.0.5.tar.gz
$ mv index.html regress.tar.gz
$ tar -xvf regress.tar.gz
$ chmod -R a+wrx /opt/regressiontests-5.0.5
$ cd regressiontests-5.0.5
$ ./gmxtest.pl all -np 2

#####################################
#Install Gromacs (pomocí icc kompilátoru)
export PATH=/opt/openmpi/1.8.5/bin:$PATH
export LD_LIBRARY_PATH=/opt/openmpi/1.8.5/lib64:$LD_LIBRARY_PATH
###
#single precision + GPU
###
$ cmake .. -DCMAKE_C_COMPILER=icc -DCMAKE_CXX_COMPILER=icpc
-DGMX_MPI=OFF -DGMX_GPU=ON -DGMX_DOUBLE=OFF
-DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON
-DCMAKE_INSTALL_PREFIX=/opt/gromacs-5.0.5/bin_icc
$ make
$ make check
$ make install

#mpi
$ cmake .. -DCMAKE_C_COMPILER=icc -DCMAKE_CXX_COMPILER=icpc
-DGMX_MPI=ON -DGMX_GPU=ON -DGMX_DOUBLE=OFF
-DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON
```

```
-DCMAKE_INSTALL_PREFIX=/opt/gromacs-5.0.5/bin_icc
$ make
$ make check
$ make install

###
#double precision – GPU double precision není podporován
###
$ cmake .. -DCMAKE_C_COMPILER=icc -DCMAKE_CXX_COMPILER=icpc
-DGMX_GPU=OFF -DGMX_DOUBLE=ON -DGMX_BUILD_OWN_FFTW=ON
-DREGRESSIONTEST_DOWNLOAD=ON -DCMAKE_INSTALL_PREFIX=/opt/gromacs-
5.0.5/bin_icc_double
$ make
$ make check
$ make install

#mpi
$ cmake .. -DCMAKE_C_COMPILER=icc -DCMAKE_CXX_COMPILER=icpc
-DGMX_MPI=ON -DGMX_GPU=OFF -DGMX_DOUBLE=ON
-DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON
-DCMAKE_INSTALL_PREFIX=/opt/gromacs-5.0.5/bin_icc_double
$ make
$ make check
$ make install

###
#uklizení buildu Gromacsu
$ cd /opt/gromacs-5.0.5/build/
$ rm -R *
```

Použité přepínače CMake [6, 7]:
Pro změnu C a C++ kompilátoru se použijí příznaky:
-DCMAKE_C_COMPILER
-DCMAKE_CXX_COMPILER

Pro použití intelovského kompilátoru bude použit [6]:
-DCMAKE_C_COMPILER=icc
-DCMAKE_CXX_COMPILER=icpc

Pro GNU kompilátor [6]:
-DCMAKE_C_COMPILER=gcc
-DCMAKE_CXX_COMPILER=g++

Pro instalaci MPI do Gromacsu je nutné zapnout *GMX_MPI* [6]:
-DGMX_MPI=ON

Defaultně je Gromacs nastaven na single precision. Pro double precision musí tedy být nastaven příznak *GMX_DOUBLE* [6, 7]:
-DGMX_DOUBLE=ON

Gromacs podporuje GPU výpočty pouze se single precision. Proto je lépe u single precision kompilaci příznak *GMX_GPU* zapnout a při double precision je nutné *GMX_GPU* vypnout [7]:
-DGMX_GPU=OFF

Povolení instalaci Gromacsu automaticky stáhnout a vytvořit build FFTW ze zdroje se uděluje:
-DGMX_BUILD_OWN_FFTW=ON
Tato volba je doporučována Gromacs týmem kvůli výkonu simulací. Defaultně totiž Gromacs používá „mixed“ floating point precision, u kterého se používá single precision ve FFTW. Defaultní FFTW balíček je ale double precision a při linkování FFTW s Gromacsem mohou být použity špatné volby kompilátoru. [6, 7]

Regression test slouží pro ověření buildu Gromacsu. Nejjednodušším způsobem je udělat build s volbou -*DREGRESSIONTEST_DOWNLOAD* a spustit *make check* [6, 7]:
-DREGRESSIONTEST_DOWNLOAD=ON

Adresář pro instalaci se může změnit příznakem *DCMAKE_INSTALL_PREFIX*. Např.:
-DCMAKE_INSTALL_PREFIX=/opt/gromacs-5.0.5/bin_icc_double. [6]

2.4.2. Příprava cest Gromacsu

Program Gromacs se připraví pro použití načtením příslušného *GMXRC* souboru dle toho, která verze programu má být použita.
Pro zavedení různých verzí Gromacs do systému byly vytvořeny skripty *singlegccgromacs.sh*, *doublegccgromacs.sh*, *singleiccgromacs.sh*, *doubleiccgromacs.sh*, *singlegccgromacsicc.sh*, *singleiccgromacsgcc.sh*, *doublegccgromacsicc.sh*, *doubleiccgromacsgcc.sh* v adresáři */home/krejsa/scriptsmoje/*:

```
#Gromacs verze kompilovaná gcc kompilátorem, se single precision, podporou GPU a gcc Open MPI
$ vim /home/krejsa/scriptsmoje/singlegccgromacs.sh
source /opt/gromacs-5.0.5/bin/bin/GMXRC
source /home/krejsa/scriptsmoje/exportOpenMPI.sh
```

```
#Gromacs verze kompilovaná gcc kompilátorem s double precision a gcc Open MPI
$ vim /home/krejsa/scriptsmoje/doublegccgromacs.sh
source /opt/gromacs-5.0.5/bin_double/bin/GMXRC
source /home/krejsa/scriptsmoje/exportOpenMPI.sh
```

```
#Gromacs verze kompilovaná icc kompilátorem, single precision, podporou GPU a icc Open MPI
$ vim /home/krejsa/scriptsmoje/singleiccgromacs.sh
source /opt/gromacs-5.0.5/bin_icc/bin/GMXRC
source /home/krejsa/scriptsmoje/exportIccOpenMPI.sh
```

```
#Gromacs verze kompilovaná icc kompilátorem s double precision a icc Open MPI
```

```
$ vim /home/krejsa/scriptsmoje/doubleiccgromacs.sh
source /opt/gromacs-5.0.5/bin_icc_double/bin/GMXRC
source /home/krejsa/scriptsmoje/exportIccOpenMPI.sh
```

```
#Gromacs verze kompilovaná gcc kompilátorem, single precision a icc Open MPI
$ vim /home/krejsa/scriptsmoje/singlegccgromacsicc.sh
source /opt/gromacs-5.0.5/bin/bin/GMXRC
source /home/krejsa/scriptsmoje/exportIccOpenMPI.sh
```

```
#Gromacs verze kompilovaná icc kompilátorem, single precision a gcc Open MPI
$ vim /home/krejsa/scriptsmoje/singleiccgromacsgcc.sh
source /opt/gromacs-5.0.5/bin_icc/bin/GMXRC
source /home/krejsa/scriptsmoje/exportOpenMPI.sh
```

```
#Gromacs verze kompilovaná gcc kompilátorem s double precision a icc Open MPI
$ vim /home/krejsa/scriptsmoje/doublegccgromacsicc.sh
source /opt/gromacs-5.0.5/bin_double/bin/GMXRC
source /home/krejsa/scriptsmoje/exportIccOpenMPI.sh
```

```
#Gromacs verze kompilovaná icc kompilátorem s double precision a gcc Open MPI
$ vim /home/krejsa/scriptsmoje/doubleiccgromacsgcc.sh
source /opt/gromacs-5.0.5/bin_icc_double/bin/GMXRC
source /home/krejsa/scriptsmoje/exportOpenMPI.sh
```

Následující řádky slouží k exportování cest Open MPI do proměnných prostředí. Exporty budou zapsány do souboru */etc/profile*, aby se vykonaly hned při startu bashe/shellu.

```
#exportování knihoven a binárních souborů Open MPI do proměnných PATH
a LD_LIBRARY_PATH
$ vim /etc/profile
export PATH=$PATH:/opt/openmpi/1.8.5/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/openmpi/1.8.5/lib64
$ source ~/.bashrc #znovu načte bash bez nutnosti se odhlašovat => zavede provedené změny
```

Jednoduché volání různých variant programu Gromacs bude umožněno pomocí aliasů v souboru *.bashrc*:

```
$ cd
$ vim .bashrc
#na konec souboru .bashrc zapíšeme následující řádky
alias singlegromacs='source /home/krejsa/scriptsmoje/singlegccgromacs.sh'
alias doublegromacs='source /home/krejsa/scriptsmoje/doublegccgromacs.sh'
alias singleiccgromacs='source /home/krejsa/scriptsmoje/singleiccgromacs.sh' #OpenMPI icc,
Gromacs icc
```

```
alias doubleiccgromacs='source /home/krejsa/scriptsmoje/doubleiccgromacs.sh' #OpenMPI icc, Gromacs icc
alias singlegromacsicc='source /home/krejsa/scriptsmoje/singlegccgromacsicc.sh' #openMPI icc, Gromacs gcc
alias doublegromacsicc='source /home/krejsa/scriptsmoje/doublegccgromacsicc.sh' #OpenMPI icc, Gromacs gcc
alias singleiccgromacsgcc='source /home/krejsa/scriptsmoje/singleiccgromacsgcc.sh' #OpenMPI gcc, Gromacs icc
alias doubleiccgromacsgcc='source /home/krejsa/scriptsmoje/doubleiccgromacsgcc.sh' #OpenMPI gcc, Gromacs icc
$ source ~/.bashrc #znovu načte bash bez nutnosti se odhlašovat => zavede provedené změny
```

Díky aliasům je možné snadno volat předpřipravené skripty. Text napsaný za klauzulí *alias* je název, který se bude volat v bashi. Například bude zavolán Gromacs s gcc kompilátorem a single precision. Bude tedy volán skript *singlegccgromacs.sh*:

```
$ singlegromacs #volání Gromacs verze single precision, gcc, GPU
```

Nyní je tato varianta programu Gromacs připravena k použití. Obdobně se postupuje při volbě jiné nainstalované verze programu.

Tyto napsané skripty jsou nyní vázané aliasem pouze na uživatele *krejsa*, ovšem v případě potřeby, např. použitím příkazu *source* a uvedení celé cesty skriptu, je lze také použít jiným uživatelem.

3. Seznámení s programem Gromacs

Gromacs je softwarový balík pro molekulární modelování. Jako velice dobrý zdroj, návod a popis fungování programu Gromacs poslouží bakalářská práce Hany Barvíkové, „Počítačové modelování interakcí molekul s minerálními povrchy“. [12]
Tato kapitola vás stručně seznámí s tím, jak vytvořit spustitelný soubor pro simulaci systému, jak ho spustit a jaké jsou možnosti při spouštění simulace s použitím GPU.

V souvislosti s využíváním GPU v Gromacs se objevují některé výrazy [15, 24]:
uzel (node) – je jeden počítač.

MPI – pomocí MPI je v Gromacsu zajištěna komunikace mezi uzly.

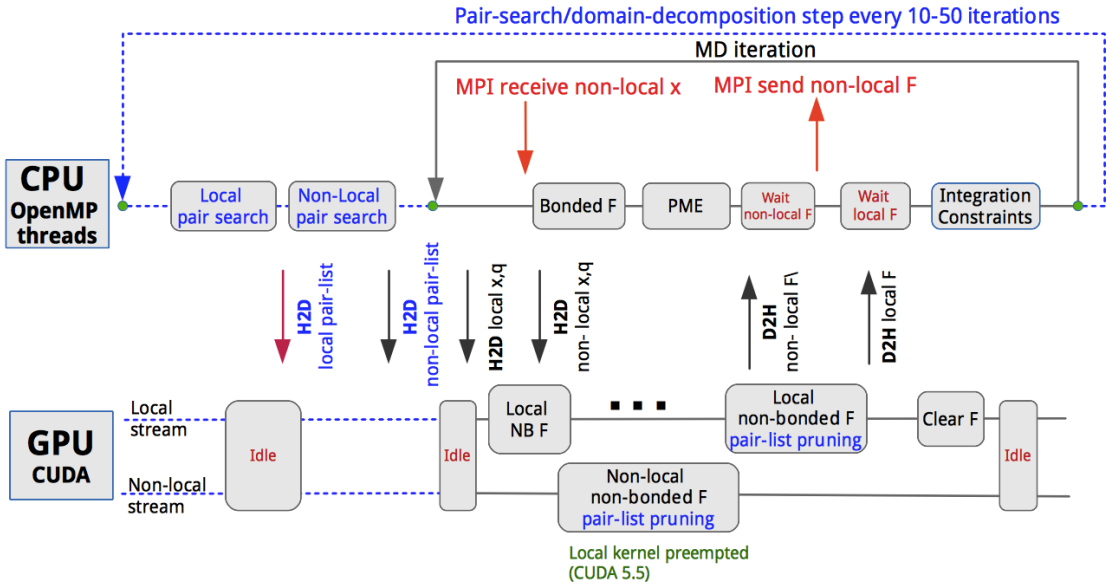
rank – v MPI, rank je hardware sloužící pro paralelizaci mezi uzly. Tento hardware může být řízen uživatelem a může odpovídat např. jádru (jednotka, která vykonává instrukce), nebo skupině jader a GPU. Počet ranků je vhodné zvolit dle počtu GPU a mezi tyto ranky rozdělit jádra.

GPU – je vždy asociován s určitým uzlem a často s určitými jádry v tomto uzlu.

OpenMP – standardizovaná technika podporovaná mnoha kompilátory, která sdílí s několika jádry výpočetní zátěž.

SIMD (single instruction multiple data) – CPU jádra mají instrukce, které mohou vykonat velké množství floating-point instrukcí v jednom cyklu.

3.1. GPU paralelizace



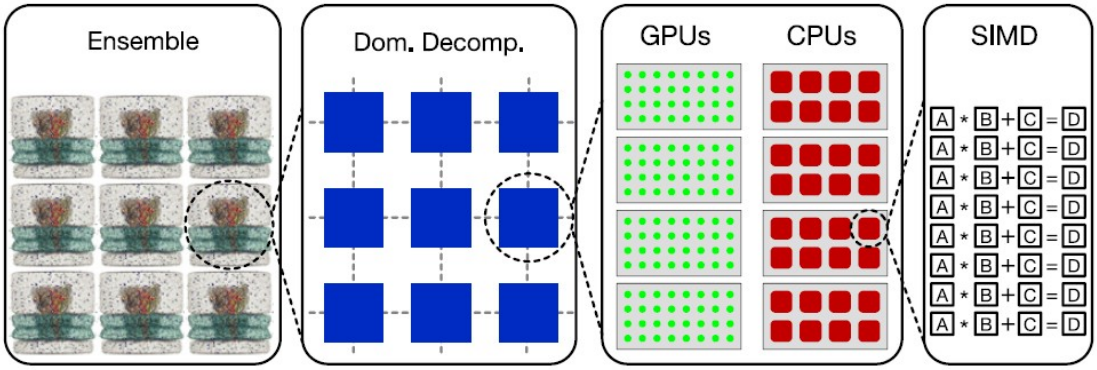
Obrázek 3: Heterogenní paralelizace [11]

Paralelizaci a komunikaci CPU s GPU popisuje obrázek 3. Pokud je k výpočtu použita více než jedna GPU, jedná se o výpočet s doménovou dekompozicí. Simulace bez GPU, nebo s jednou GPU nepoužívá doménovou dekompozici [11].

Popis heterogenní paralelizace
Nejprve CPU vyhledává páry atomů nebo molekul, které se pošlou také na GPU. GPU ještě čeká na přijetí souřadnic částic. Po přijetí párů a souřadnic začíná GPU počítat nevazebné síly (van der Waalsovy síly a část elektrostatických interakcí počítaných v reálném prostoru, tj. Do vzdálenosti interakčního cutoffu), které po jejich vypočtení pošle zpět na CPU. CPU mezitím počítá vazebné síly, reciprokou část elektrostatických interakcí počítaných metodou PME (Particle Mesh Ewald) pomocí algoritmu 3D FFT (rychlá Fourierovská transformace). Po přijetí nevazebných sil od GPU se na základě vypočtených sil spočítají nové polohy částic a cyklus opět začíná od začátku posláním nových poloh částic na GPU. Každých 10-50 iterací se znovu vyhledají páry v rámci seznamu nejbližších sousedů. [11, 24]

Dedikované PME ranky
Dedikované PME ranky počítají reciprokou část PME. PP (Particle-Particle) ranky pak počítají část elektrostatických interakcí v reálném prostoru. Zvětšením cutoffu krátkodosahových interakcí se zároveň zvětšují PME buňky a tím snižuje jejich počet. Gromacs může postupně přesunout výpočetní zátěž mezi PP a PME výpočet. To je realizováno formou automatizovaného statického vyrovnávání zátěže mezi CPU a GPU, nebo mezi PP a PME ranky. Statické vyrovnávání zátěže se provádí během prvních několika stovek až tisíců simulačních kroků.
Není-li žádný PME rank dedikován, nejprve se na všech MPI rancích počítá PP a pak PME. [24]

Doménová dekompozice (DD)
Gromacs používá DD k rozdělení systému do $N_{DD} = DD_x * DD_y * DD_z$ zpočátku stejně velkých buněk. Každá buňka je pak přiřazena jednomu MPI ranku. Jestliže je aktivní dynamické vyvažování zátěže (dynamic load balancing – DLB), velikost buněk DD se postupně přizpůsobuje. Implicitně Gromacs používá 1 GPU na 1 DD buňku. Každá GPU je mapovaná na PP rank. V rámci PP ranku, OpenMP vlákna mohou sdílet vytížení, nebo práce může být přesunuta na GPU. PP rank se také stará o vazebné interakce v rámci své domény. [15, 24]



Obrázek 4: Víceúrovňová paralelizace v Gromacs [16]

3.2. Spouštění Gromacsu

Za prvé, musí být zaveden program Gromacs do systému spuštěním *GMXRC* souboru. Cesty k *GMXRC* se liší dle verze Gromacsu (k zavedení programu lze použít i předpřipravený skript z kapitoly 2.4.2).

```
$ source /opt/gromacs-5.0.5/bin/bin/GMXRC
```

Na příkladu systému vody bude ilustrován postup

- vytvoření spustitelného souboru *w1.tpr,*
- jeho spuštění a možnosti, které souvisí se spouštěním simulací systémů na GPU.

Soubor *w1.tpr* se sestaví příkazem *grompp* [12]:

```
$ grompp -c w1.gro -p w1.top -f md.mdp -o w1.tpr
```

Simulace programu Gromacs se spouští příkazem *mdrun* [12]:

```
$ mdrun -nb gpu_cpu -ntmpi 2 -ntomp 8 -deffnm w1
```

Příznaky příkazu *mdrun* [12, 15, 24]:
-nb

- *gpu_cpu* – spustí výpočet lokálních nevazebných sil (non-bonded force) na GPU a nelokální nevazebné síly se spouští na CPU
- *gpu* – všechny nevazebné síly se počítají na GPU
- *cpu* – vynucuje použití pouze CPU

-maxwarn 10 – ignoruje až 10 varování

-ntomp X – číslo X specifikuje počet OpenMP vláken spouštěných na CPU v rámci jednoho ranku.

-ntmpi X – číslo X specifikuje počet thread-MPI vláken (a tedy i počet MPI ranků). Toto číslo odpovídá počtu fyzických GPU karet. Není-li tato hodnota specifikována, je automaticky nastavena dle počtu detekovaných GPU. Pokud je X>11, *mdrun* automaticky dedikuje PME ranky pro výpočet PME a to PP : PME v poměru 3 : 1 (např. je-li specifikováno 16 MPI ranků, jsou ranky rozděleny na 12 PP ranků a 4 PME ranky). [24]

-npme X – číslo X specifikuje počet ranků použitých pro PME výpočet. Uváděné nejefektivnější rozdělení ranků je právě v poměru 3 : 1 (PP : PME). Dedikování PME ranků lze předejít hodnotou X=-1. MPI rank = PP rank + PME rank. [24]

Gromacs při simulaci vytváří soubor *w1.log* a další soubory, do kterých zapisuje výsledky a informace o průběhu simulace. Právě v souboru *w1.log* můžeme vidět všechny záznamy o tom, jaké byly vstupní konfigurace, souhrnné výsledky, čas běhu simulace, počet přidělených vláken a počet namapovaných GPU, příkaz, kterým byla simulace spuštěna a mnoho dalšího.

3.2.1. Gram

Stroje Gram jsou součástí MetaCentra. Každý stroj má dostupných 16 vláken CPU a 4 GPU, které mohou být použity pro výpočet molekulárních simulací v programu Gromacs. Výpočty se spouští submitovacím skriptem, který zařadí úlohu do fronty pomocí příkazu *qsub* [19].
Příklad vytvořeného submitovacího skriptu:

```
#!/bin/sh
#PBS -q gpu
#PBS -l walltime=30:00
#PBS -l nodes=1:ppn=16:gpu=1:home_gram
#PBS -l mem=16gb
#PBS -l scratch=16gb
#PBS -j oe
hostname
cat $PBS_NODEFILE

# setting the automatical cleaning of the SCRATCH
# (unless set otherwise, the data will be deleted)
trap 'clean_scratch' TERM EXIT

# set the working directory
WORK=vypocty/gromacsvypocty/watergpu-cpu/w1
SOURCE=/storage/plzen1/home/krejst00

# input data copy
cp $SOURCE/$WORK/* $SCRATCHDIR  || exit 1

# starting computation in the working directory
cd $SCRATCHDIR

# adding the module
module add cuda-6.5
```

```
module add gromacs-5.0.5-gpu
#grompp -c w1.gro -p w1.top -f md.mdp -o w1.tpr

# execution
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w1
cp w1.log w1_GccGcc_5
# copy results from the scratch  (if there is something wrong, leave data in SCRATCHDIR and informes the user)
cp $SCRATCHDIR/* $SOURCE/$WORK/ || export CLEAN_SCRATCH=false
```

V submitovacím skriptu je nutné stanovit dobu výpočtu (*walltime*), počet strojů (*nodes*), vláken na jednom stroji (*ppn*) a GPU (*gpu*), které si program zarezervuje. Skript dále z uživatelského adresáře do prostoru na konkrétním stroji, kde se bude program vykonávat, nahraje soubory s konkrétním molekulárním systémem. Poté skript nahraje *cuda-6.5* a *gromacs-5.0.5-gpu* moduly a výpočet může začít příkazem *mdrun*. Jakmile simulace skončí, zkopírují se z dočasného prostoru všechny soubory zpět do uživatelského adresáře.

3.2.2. MP-9

Stroj MP-9 disponuje 8 CPU vlákny a 1 GPU. Zde se cesty k programu Gromacs a MPI načtou předpřipraveným skriptem, který byl připraven v kapitole 2.4.2.
Příprava a spouštění simulací molekulárních systémů se provádí tak, jak je naznačeno v této kapitole výše (3.2). Rozdíl je pouze v realizaci nastavení cest. Místo *source …/GMXRC* lze použít například skript *singleiccgromacs.sh*.

3.2.3. MP-8

Počítač MP-8 je nejslabší z testovaných strojů a má 4 CPU vlákna a 1 GPU. Načítání cest Gromacsu je realizováno pomocí příkazu *source* (např: *source /opt/gromacs-5.0.5/bin_gpu/bin/GMXRC*). Na tomto stroji jsou zvlášť nainstalovány single, double a gpu verze programu Gromacs.
Příklad submitovacího skriptu:

```
#!/bin/bash
#PBS -N w1
#PBS -l nodes=1:ppn=4:mp8
#PBS -j oe
cat $PBS_NODEFILE

WORK=computeFolder/watergpu-cpu/w1        #adresář s konkrétním molekulárním systémem
SOURCE=/home/krejsa          #domovský adresář
SCRATCH=/local/krejsa/$WORK
rm -r $SCRATCH
mkdir -p $SCRATCH    #dočasný adresář na stroji, kde se bude provádět simulace
cp -r $SOURCE/$WORK/* $SCRATCH
cd $SCRATCH

source /opt/gromacs-5.0.5/bin_gpu/bin/GMXRC          #cesta k GPU verzi Gromacsu na MP 8
mdrun -nb cpu -deffnm w1
cp -r $SCRATCH/* $SOURCE/$WORK
rm -r $SCRATCH
```

Skript nastaví počet strojů (*nodes*), vláken CPU (*ppn*) a název stroje (*mp8*), kde se simulace vykoná. Dále nastaví do proměnných cesty k adresářům (*WORK, SOURCE, SCRATCH*). Vytvoří dočasný adresář na stroji, kde se bude provádět simulace a zkopíruje tam konkrétní systém. Příkazem *source skript* nastaví cestu k GPU verzi programu Gromacs. *Mdrun* spustí simulaci a následně se celý systém zkopíruje zpět na původní místo v domovském adresáři.

3.3. Rozdíly oproti staré verzi Gromacs

Nová verze programu Gromacs 5.0.5 přináší některé změny oproti starší verzi. Změny byly například v *md.mdp* nebo *w1.top* vstupních souborech pro preprocessing programu *grompp*. Následující jsou jen ty, které byly řešeny v rámci diplomové práce.

3.3.1. md.mdp

V souboru *md.mdp* jsou některé změny názvů voleb.
Na některé změny v konfiguračním souboru *md.mdp* upozorní i sám program Gromacs při sestavování souboru *w1.tpr.* [4]

1) cutoff-scheme = Verlet
   - V používané vezi Gromacs 5.0.5 je cutoff-scheme = group zastaralý a v novějších verzích bude zcela nahrazen schématem Verlet. Proto ve všech zde užívaných *mdp* souborech je již použit Cutoff-scheme = Verlet. Pokud není toto schéma použito, program při sestavování *w1.tpr* souboru hlásí varování
2) nstxout-compressed
   - nstxout-compressed nahrazuje zastaralý nstxtcout. V případě použití této zastaralé volby, g*rompp p*ři sestavování *.tpr* souboru, ve verzi Gromacs 5.0.5, hodnotu z nstxtcout použije do nové nstxout-compressed, která starou volbu automaticky nahradí
3) optimize_fft
   - zastaralá volba optimize_fft je odstraněna a *grompp* ji ignoruje

3.3.2. w1.top

V nové verzi došlo k přejmenování některých vkládaných topologií (soubory s příponou *.itp*) a změně jejich cest. Tyto topologie se vkládají pomocí #*include* do souboru s příponou *.top*. [4]

1) #include "oplsaa.ff/forcefield.itp"
   - *#include "oplsaa.ff/forcefield.itp"* nahrazuje *#include "ffoplsaa.itp"* ze starších verzí Gromacsu
2) #include "oplsaa.ff/spce.itp"
   - *#include "oplsaa.ff/spce.itp"* nahrazuje *#include "spce.itp"* ze starších verzí Gromacsu

4. Testování GPU verze programu Gromacs

V programu Gromacs jsou testovány různé parametry v systému vody. Výpočty probíhají na čtyřech systémech vody – w1, w10, w100 a w1000. Systém w1 má 1000 molekul, w10 (10000 molekul), w100 (100000 molekul) a w1000 má milión molekul. Testované parametry jsou následující:

- porovnání rychlosti simulací programu Gromacs, který je zkompilovaný ICC, nebo GCC kompilátorem
- zrychlení simulací pouštěných s GPU oproti pouštěných pouze na CPU

- vliv velikosti cutoffu (oříznutí interakčního potenciálu, viz. [12]) na rychlost výpočtu simulace
- různý počet GPU a počet CPU vláken a s tím související vliv na rychlost výpočtu simulace.

Dále je testován systém argonu, který má nulový náboj. Jsou testovány tři velikosti systému argonu – 1k (1000 molekul), 10k (10000 molekul), 100k (100000 molekul).

4.1. Časy simulací

Časy simulací v rámci stejných vstupních parametrů se liší. Byl tedy proveden test, kde se ukázalo, jak moc se časy od sebe liší a zvoleny takové počty kroků simulací, aby doba simulace byla řádově 1-10 minut.

MP9		Adresar:	**waterCasRozdil**			
System	nsteps	Cas 1 (s)	Cas 2 (s)	Cas 3 (s)	Cas 4 (s)	Cas 5 (s)	Prumer
w1	1000000	291,9	290,8	291,5	291,3	291,6	291,4
w10	200000	462,1	459,7	458,1	458,5	461,7	460,0
w100	10000	296,8	292,0	292,6	291,7	291,7	293,0

Tabulka 7: Testování odchylek časů simulací

V menších systémech vody nedocházelo k větším časovým výkyvům. Provedeno bylo tedy pouze 5 simulací pro každý systém.

MP9		Adresar:	**waterCasRozdil**		
System	nsteps	Cas 1 (s)	Cas 2 (s)	Cas 3 (s)	Cas 4 (s)
w1000	1000	428,2	430,5	551,6	433,0
		Cas 5 (s)	Cas 6 (s)	Cas 7 (s)	Cas 8 (s)
		490,8	481,7	440,2	508,6
		Cas 9 (s)	Cas 10 (s)	Cas 11 (s)	Cas 12 (s)
		440,5	439,0	502,4	469,5
		Cas 13 (s)	Cas 14 (s)	Cas 15 (s)	Prumer
		535,6	531,4	494,4	478,5

Tabulka 8: Testování odchylek časů simulací w1000

Rozdílné výsledky v provedených 15 simulacích w1000 ukazují, že se časy liší o ±61,72 s. Průměr z těchto 15 simulací je 478,50 s. Je-li brán průměr pouze z prvních 5 simulací, je roven 466,817 s. Rozdíl mezi těmito dvěma průměry činí 4%. Proto simulace budou opakovány pětkrát v případě, že nebude docházet k větším časovým výkyvům. Odhad chybovosti výsledků uvedených v tabulkách je tedy cca 4%.
Ověřil jsem u simulace trvající téměř 6 hodin, že její čas byl přímo úměrný počtu kroků oproti výše uvedeným krátkým simulacím.

4.2. Test ICC vs. GCC

Na základě dobré zkušenosti s ICC kompilátorem, byl zrealizován také test porovnání ICC a GCC.

Z mnoha spuštěných testů bylo vypozorováno, že ICC verze programu Gromacs je efektivnější na CPU simulacích. GCC verzi je pak lépe použít na simulace s GPU. Ostatně tabulka 9 dokládá toto tvrzení.

4.3. Simulace s GPU vs. CPU

Systém vody byl testován na zrychlení simulací na GPU a CPU. Čtyři různé velikosti systémů vody byly testovány na MP 8, MP 9 a Gramu. Počet kroků *nsteps* bylo přizpůsobeno tak, aby délka simulace byla cca 5 minut. Každá velikost systému má stanovený počet kroků a je tedy možné porovnat i výkon různých strojů na simulaci se stejnými vstupními parametry.
Pro porovnání výkonu strojů byly simulace spouštěny s následujícím počtem GPU a CPU vláken:

- Výpočty na MP 9 v tabulce 9 byly spuštěny s 8 CPU vlákny a 1 GPU.
- Výpočty na Gramu v tabulce 10 byly spuštěny s 16 CPU vlákny a 1 GPU.
- Výpočty na MP 8 v tabulce 11 byly spuštěny s 4 CPU vlákny a 1 GPU.

MP9 Single precision		Adresar: **watergpu-cpu**			
Příkaz	CPU/GPU	Počet kroků	Čas (s)	Kompilátor	Poměr CPU/GPU
mdrun -nb cpu -deffnm w1	CPU	1000000	834,2	GCC	2,9
mdrun -nb gpu_cpu -deffnm w1	GPU	1000000	292,0	GCC	
mdrun -nb cpu -deffnm w1	CPU	1000000	848,8	ICC	2,9
mdrun -nb gpu_cpu -deffnm w1	GPU	1000000	292,8	ICC	
mdrun -nb cpu -deffnm w10	CPU	200000	1 704,0	ICC	3,6
mdrun -nb gpu_cpu -deffnm w10	GPU	200000	467,9	GCC	
mdrun -nb cpu -deffnm w100	CPU	10000	1 096,4	ICC	3,7
mdrun -nb gpu_cpu -deffnm w100	GPU	10000	300,3	GCC	
mdrun -nb cpu -deffnm w1000	CPU	1000	1 191,6	GCC	2,6
mdrun -nb gpu_cpu -deffnm w1000	GPU	1000	459,1	GCC	
mdrun -nb cpu -deffnm w1000	CPU	1000	1 122,7	ICC	2,2
mdrun -nb gpu_cpu -deffnm w1000	GPU	1000	508,6	ICC	

Tabulka 9: MP 9 – porovnání kompilátorů a GPU vs. CPU

MetaCentrum Gram Single precision		Adresar: **watergpu-cpu**			
Příkaz	CPU/GPU	Počet kroků	Čas (s)	Kompilátor	Poměr CPU/GPU
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w1	CPU	1000000	748,1	GCC	1,2
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm w1	GPU	1000000	615,1	GCC	
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w10	CPU	200000	1 042,6	GCC	1,2
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm w10	GPU	200000	899,1	GCC	
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w100	CPU	10000	513,1	GCC	1,2
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm w100	GPU	10000	419,2	GCC	
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w1000	CPU	1000	671,3	GCC	1,5
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm w1000	GPU	1000	440,5	GCC	

Tabulka 10: Gram – porovnání GPU vs. CPU

MP8 Single precision		Adresar:	**watergpu-cpu**		
Příkaz	CPU/GPU	Počet kroků	Čas (s)	Kompilátor	Poměr CPU/GPU
mdrun -nb cpu -deffnm w1	CPU	1000000	1 839,0	GCC	2,2
mdrun -nb gpu_cpu -deffnm w1	GPU	1000000	841,7	GCC	
mdrun -nb cpu -deffnm w100	CPU	10000	2 257,9	GCC	2,9
mdrun -nb gpu_cpu -deffnm w100	GPU	10000	766,4	GCC	
mdrun -nb cpu -deffnm w1000	CPU	1000	2 366,0	GCC	2,3
mdrun -nb gpu_cpu -deffnm w1000	GPU	1000	1 051,5	GCC	
mdrun -nb cpu -deffnm w10	CPU	200000	3 653,0	GCC	2,5
mdrun -nb gpu_cpu -deffnm w10	GPU	200000	1 440,8	GCC	

Tabulka 11: MP 8 – porovnání GPU vs. CPU

Tabulky 9, 10 a 11 poměřují výkon simulací GPU a CPU. Ze všech strojů největší zrychlení GPU oproti CPU je 3,7× na MP 9 u systému W100. Naopak nejmenší zrychlení je 1,2× na Gramu u systému W10.
Obecně lze tedy říci, že s použitím GPU dojde ke zrychlení simulace a vyplatí se ji proto použít.

4.4. Cutoff

Další test, tabulka 12, ukazuje, jak se mění časy simulací při použití 8 vláken CPU a jak při 8 vláknech CPU a 1 GPU. Simulace s GPU byly prováděny s GCC verzí Gromacsu a simulace se samotným CPU byly prováděny s ICC verzí. Tyto simulace ukazují vliv cutoffu na dobu výpočtu na CPU resp. GPU.

MP9

1 MPI, 8 OMP			Adresář:	**water-cutoffs**	
water	1		water	10	
nsteps	1000000		nsteps	200000	
	CPU	GPU		CPU	GPU
rlist, rcoulomb, rvdw	Čas (s)	Čas (s)	rlist, rcoulomb, rvdw	Čas (s)	Čas (s)
1.0	624,2	247,8	1.0	1 292,1	401,1
1.2	848,8	292,0	1.2	1 704,0	467,9
1.5	1 272,3	394,2	1.5	2 536,0	616,3

water	100		water	1000	
nsteps	10000		nsteps	1000	
	CPU	GPU		CPU	GPU
rlist, rcoulomb, rvdw	Čas (s)	Čas (s)	rlist, rcoulomb, rvdw	Čas (s)	Čas (s)
1.0	885,7	262,3	1.0	924,4	443,8
1.2	1 096,4	300,3	1.2	1 122,7	459,1
1.5	1 769,6	367,6	1.5	1 523,5	571,6

Tabulka 12: MP 9 – vliv velikosti cutoffu na rychlost simulace

Z Tabulky 12 lze vyčíst, že s vzrůstající velikostí cutoffu roste i délka výpočtů a to jak na GPU, tak na CPU. Délka výpočtu simulace s použitím GPU ovšem není tak citlivá na změnu cutoffu na rozdíl od simulace na CPU.

4.5. Gram #GPU vs. #CPU

Na klastru Gramu v MetaCentru byl testován vliv rychlosti simulací v závislosti na počtu využitých CPU vláken a počtu GPU. Gram disponuje až 16 CPU vlákny a 4 GPU na jednom stroji.

MetaCentrum Gram Single precision, GCC kompilátor		Adresář:	**water-ranks**		
Příkaz	Počet CPU	Počet GPU	CPU/GPU	Počet kroků	čas (s)
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm w1	16	1	GPU	1000000	615,8
mdrun -nb gpu_cpu -ntmpi 2 -ntomp 8 -deffnm w1	8x2	2	GPU	1000000	638,7
mdrun -nb gpu_cpu -ntmpi 4 -ntomp 4 -deffnm w1	4x4	4	GPU	1000000	692,5
mdrun -nb gpu_cpu -ntmpi 2 -ntomp 4 -deffnm w1	4x2	2	GPU	1000000	824,9
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 8 -deffnm w1	8	1	GPU	1000000	630,0
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 4 -deffnm w1	4	1	GPU	1000000	661,7
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w1	16	0	CPU	1000000	756,8
mdrun -nb cpu -ntmpi 1 -ntomp 8 -deffnm w1	8	0	CPU	1000000	1 134,9
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm w10	16	1	GPU	200000	900,1
mdrun -nb gpu_cpu -ntmpi 2 -ntomp 8 -deffnm w10	8x2	2	GPU	200000	531,5
mdrun -nb gpu_cpu -ntmpi 4 -ntomp 4 -deffnm w10	4x4	4	GPU	200000	618,5
mdrun -nb gpu_cpu -ntmpi 2 -ntomp 4 -deffnm w10	4x2	2	GPU	200000	790,8
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 8 -deffnm w10	8	1	GPU	200000	932,1
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 4 -deffnm w10	4	1	GPU	200000	1 009,3
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w10	16	0	CPU	200000	1 046,8
mdrun -nb cpu -ntmpi 1 -ntomp 8 -deffnm w10	8	0	CPU	200000	1 862,1
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm w100	16	1	GPU	10000	419,1
mdrun -nb gpu_cpu -ntmpi 2 -ntomp 8 -deffnm w100	8x2	2	GPU	10000	227,2
mdrun -nb gpu_cpu -ntmpi 4 -ntomp 4 -deffnm w100	4x4	4	GPU	10000	245,3
mdrun -nb gpu_cpu -ntmpi 2 -ntomp 4 -deffnm w100	4x2	2	GPU	10000	332,2
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 8 -deffnm w100	8	1	GPU	10000	436,5
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 4 -deffnm w100	4	1	GPU	10000	481,1
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w100	16	0	CPU	10000	517,9
mdrun -nb cpu -ntmpi 1 -ntomp 8 -deffnm w100	8	0	CPU	10000	933,3
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm w1000	16	1	GPU	1000	441,1
mdrun -nb gpu_cpu -ntmpi 2 -ntomp 8 -deffnm w1000	8x2	2	GPU	1000	320,5
mdrun -nb gpu_cpu -ntmpi 4 -ntomp 4 -deffnm w1000	4x4	4	GPU	1000	314,2
mdrun -nb gpu_cpu -ntmpi 2 -ntomp 4 -deffnm w1000	4x2	2	GPU	1000	425,4
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 8 -deffnm w1000	8	1	GPU	1000	455,6
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 4 -deffnm w1000	4	1	GPU	1000	552,2
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w1000	16	0	CPU	1000	636,1
mdrun -nb cpu -ntmpi 1 -ntomp 8 -deffnm w1000	8	0	CPU	1000	1 050,8

Tabulka 13: Gram – rychlost simulací v závislosti na počtu GPU a CPU vláken

V tabulce 13 jsou uvedeny časy čtyř různě velkých systémů vody testovaných s různými počty GPU a CPU vláken.

Téměř ve všech velikostech systémů lze pozorovat, že výpočty s GPU jsou rychlejší než použití samotného CPU. Jedinou výjimkou je systém vody s tisíci molekulami, u kterého je s použitím 16 CPU vlákny simulace rychlejší než s použitím 8 CPU vláken a 2 GPU.
Na systému vody w1 je efektivní využít 16 CPU vláken a 1 GPU. Ovšem se zvětšující se velikostí sytému je lepší využít větší počet GPU. Je to způsobené tím, že na větších systémech vody CPU dlouho čeká na GPU, která musí nejprve své výpočty dokončit (dobu čekání na GPU lze vyčíst ze souboru *.log*). V důsledku zvýšení počtu GPU se tedy sníží čekací doba CPU a celkový čas simulace se tak zkrátí. U největšího testovaného systému w1000 byla naměřena nejrychlejší simulace za použití 16 CPU vlákny a 4 GPU.

Naopak pokud jsou k dispozici 4 simulace, které je nutné spočítat, je nejefektivnější v podílu vůči strojům na všech systémech vody spustit 4 simulace zároveň v konfiguraci se 4 CPU vlákny a 1 GPU. Pokud ovšem jsou k dispozici pouze 2 simulace, je u systémů w10, w100 a w1000 nejefektivnější spustit 2 simulace zároveň, každá s 8 CPU vlákny a 2 GPU. Na systému w1 by byly nejrychleji dokončeny 2 simulace spuštěné zároveň, každá s 8 CPU vlákny a 1 GPU.

4.6. Argon

Jak velké může být zrychlení simulace s GPU oproti CPU měl ukázat systém argonu. Argon totiž má nulový náboj a očekává se od něj obrovské zrychlení. Výpočet argonu byl proveden na třech velikostech – 1k, 10k a 100k. Počet kroků *nsteps* je opět zafixovaný na hodnotách odpovídajících přibližně 5 minutám pro lepší porovnání výkonu strojů.
Pro porovnání výkonu strojů byly simulace spouštěny s následujícím počtem GPU a CPU vláken:

- Výpočty na MP 9 v tabulce 14 byly spuštěny s 8 CPU vlákny a 1 GPU.
- Výpočty na Gramu v tabulce 15 byly spuštěny s 16 CPU vlákny a 1 GPU. Pouze žlutě vyznačené záznamy odpovídají nejrychlejší kombinaci počtů CPU vláken a více než jedné GPU.
- Výpočty na MP 8 v tabulce 16 byly spuštěny s 4 CPU vlákny a 1 GPU.

MP9 Single precision		Adresar: **ar**			
Příkaz	CPU/GPU	Počet kroků	Čas (s)		Poměr CPU/GPU
mdrun -nb cpu -deffnm ar	CPU	3000000	206,4	1k	0,7
mdrun -nb gpu_cpu -deffnm ar	GPU	3000000	285,1	1k	
mdrun -nb cpu -deffnm ar	CPU	700000	372,6	10k	1,4
mdrun -nb gpu_cpu -deffnm ar	GPU	700000	262,6	10k	
mdrun -nb cpu -deffnm ar	CPU	70000	387,1	100k	1,7
mdrun -nb gpu_cpu -deffnm ar	GPU	70000	231,7	100k	

Tabulka 14: Test argonu na MP 9

Nejmenší systém argonu na MP 9 je rychlejší spouštět s 16 vlákny CPU, ale se zvětšujícím se systémem je efektivnější použít GPU.

MetaCentrum Gram Single precision Příkaz	CPU/GPU	Počet kroků	čas (s)		Poměr CPU/GPU
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm ar	CPU	3000000	374,8	1k	0,7
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm ar	GPU	3000000	531,0	1k	
mdrun -nb gpu -ntmpi 2 -ntomp 4 -deffnm ar	GPU	3000000	399,1	1k	
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm ar	CPU	700000	322,2	10k	0,7
mdrun -nb gpu_cpu -ntmpi 1 -ntomp 16 -deffnm ar	GPU	700000	461,3	10k	
mdrun -nb gpu -ntmpi 4 -ntomp 4 -deffnm ar	GPU	700000	213,8	10k	
mdrun -nb cpu -deffnm ar	CPU	70000	265,1	100k	0,7
mdrun -nb gpu_cpu -deffnm ar	GPU	70000	357,7	100k	
mdrun -nb gpu_cpu -ntmpi 4 -ntomp 4 -deffnm ar	GPU	70000	108,2	100k	

Tabulka 15: Test argonu na Gramu

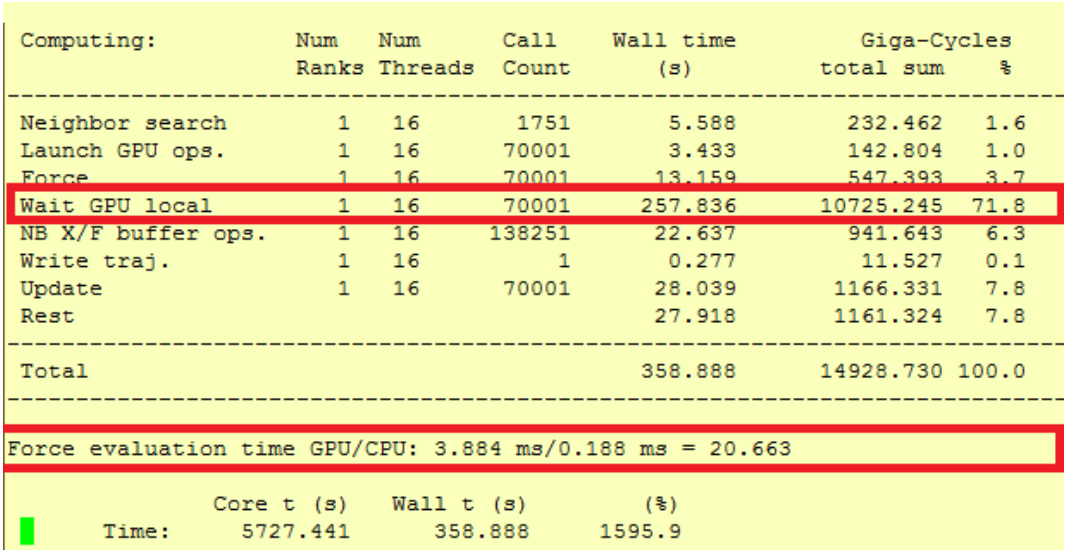
Poměry výkonu CPU/GPU v tabulce 15 jsou uváděny pouze pro simulace s 1 GPU a 16 CPU vlákny.

MP8 Single precision Příkaz	CPU/GPU	Počet kroků	Čas (s)		Poměr CPU/GPU
mdrun -nb cpu -deffnm ar	CPU	3000000	421,5	1k	0,8
mdrun -nb gpu_cpu -deffnm ar	GPU	3000000	523,1	1k	
mdrun -nb cpu -deffnm ar	CPU	700000	818,1	10k	1,3
mdrun -nb gpu_cpu -deffnm ar	GPU	700000	653,0	10k	
mdrun -nb cpu -deffnm ar	CPU	70000	845,7	100k	1,4
mdrun -nb gpu_cpu -deffnm ar	GPU	70000	592,7	100k	

Tabulka 16: Test argonu na MP 8

Se zvětšujícím se systémem argonu efektivita GPU na MP 8 také roste. Nejmenší systém je rychlejší spouštět s 4 vlákny CPU, ale na argonu 10k je zrychlení na GPU 1.3× a na 100k je zrychlení 1.4×.

Dle výsledků a záznamů z *ar.log* souborů lze pozorovat dlouhé čekání na GPU (Obrázek 5). Zhruba polovinu z celkového času výpočtu CPU tráví čekáním na GPU. Dle materiálu z workshopu Parallelization schemes & GPU Acceleration (http://www.gromacs.org/@api/deki/files/213/=gromacs_parallelization_acceleration.pdf) lze se této situace vyvarovat použitím méně CPU vláken nebo použití rychlejší GPU. Spouštění simulace na méně CPU vláknech celkově výpočet nezrychlí. Použít rychlejší GPU ne vždy lze. Gram poskytuje pomalejší, ale 4 GPU. [11]



```
Computing:          Num   Num      Call    Wall time         Giga-Cycles
                    Ranks Threads  Count      (s)         total sum    %
-----------------------------------------------------------------------------
Neighbor search        1   16       1751       5.588        232.462   1.6
Launch GPU ops.        1   16      70001       3.433        142.804   1.0
Force                  1   16      70001      13.159        547.393   3.7
Wait GPU local         1   16      70001     257.836      10725.245  71.8
NB X/F buffer ops.     1   16     138251      22.637        941.643   6.3
Write traj.            1   16          1       0.277         11.527   0.1
Update                 1   16      70001      28.039       1166.331   7.8
Rest                                          27.918       1161.324   7.8
-----------------------------------------------------------------------------
Total                                        358.888      14928.730 100.0
-----------------------------------------------------------------------------

Force evaluation time GPU/CPU: 3.884 ms/0.188 ms = 20.663

               Core t (s)   Wall t (s)        (%)
       Time:     5727.441      358.888     1595.9
```

Obrázek 5: Záznam ze souboru ar.log

Protože na výpočet s jednou GPU se čeká, na Gramu je proveden test na systému argonu při využití více grafických karet.

V tabulce 15 žlutě zvýrazněné výsledky jsou simulace argonu na Gramu s počtem CPU vláken a více GPU, které odpovídají nejrychlejší simulaci argonu v dané velikosti systému. Ostatní výsledky v tabulce jsou s 16 CPU vlákny, nebo 16 CPU vlákny a 1 GPU.
V systému 1k argon byl za použití GPU, naměřen nejrychlejší čas s 8 CPU vlákny a 4 GPU. Nelokální nevazebné síly byly počítány na GPU. Ovšem stále nejrychlejšího času dosahovala simulace bez použití GPU.
Systém 10k argon dosahoval nejrychlejšího času s 16 CPU vlákny a 4 GPU. Nelokální nevazebné síly byly také počítány na GPU. Nyní již bylo zrychlení 1.5× větší oproti testu na 16 CPU vláknech (Tabulka 15).
Systém 100k argon dosahoval nejrychlejšího času také s 16 CPU vlákny a 4 GPU. Nelokální nevazebné síly byly ale počítány na CPU. Zrychlení bylo 2.5× větší než s 16 CPU vlákny (Tabulka 15).

5. Vývoj softwaru pro čtení xtc trajektorií

Formát xtc je komprimovaný binární soubor, který obsahuje souřadnice všech atomů v čase a velikosti boxu. Protože je soubor komprimovaný, zabírá méně místa na disku a je tedy hojně užívaný. Jedná se o zásadní soubor pro strukturní analýzu simulací v rámci post-procesingu. V typické produkční simulaci dosahuje jeho velikost jednotek GB. Program Gromacs obsahuje nástroje pro základní strukturní analýzy pracující s xtc formátem, ale vedoucí práce potřebuje provádět specifičtější analýzy, které nejsou dostupné. Pro tento účel dosud převáděl binární xtc trajektorie na textové gro trajektorie pomocí standardního nástroje trjconv a textové trajektorie dále analyzoval vlastními programy. Tento postup však vyžaduje objemné xtc trajektorie konvertovat pouze pro účely těchto analýz na několikanásobně větší gro trajektorie, což je náročné na diskový prostor. Požadavkem vedoucího práce proto bylo vytvořit program, který bude číst xtc trajektorie, a do kterého bude moci vkládat vlastní kód pro jejich analýzu. Původním záměrem bylo tento program napsat pro GPU. Vzhledem k většímu než plánovanému času věnovanému ostatním částem práce byla nakonec realizována jenom hlavní část programu – načtení xtc trajektorie. Proto jsem vytvořil program *tread-xtc.out*. Vytvoření programu

pro čtení *.xtc* souborů lze díky existenci oddělených knihoven pro čtení a zápis *.xtc*, *.edr* a *.trr* souborů, která je distribuovaná pod GNU veřejnou licencí. Knihovna je založena na obslužných rutinách xdr. [12, 21, 25]

Zprvu byl záměr použít ke čtení *.xtc* souborů přímo Gromacs knihovny, které využívají programy Gromacs (např. gmx_rdf). Po vložení hlavičkových souborů do souboru *main.c* a zkompilování programu s linkem -lgmx.$(CPU) neproběhla kompilace úspěšně. Použití starší verze Gromacs 4.5.3 vedlo na stejnou chybu. Při kompilaci nebyl totiž nalezen link -lgmx.$(CPU). Bez použití tohoto linku a dalších jiných pokusech se objevovala chybová hláška „nedefinovaný odkaz na funkci“. Nakonec ale byly nalezeny výše zmíněné knihovny xdrfile-1.1.4, které byly funkční. [20, 25]

5.1. Instalace balíčku

Ze stránek programu Gromacs (http://www.gromacs.org/Developer_Zone/Programming_Guide/XTC_Library) jsem stáhl balíček knihoven, který jsem rozbalil a nainstaloval s právy uživatele root: [21]

```
$ cd /home/krejsa/gromacsvypocty/programovani/rxfile/instalace-xdr
$ wget ftp://ftp.gromacs.org/pub/contrib/xdrfile-1.1.4.tar.gz
$ tar -xvf xdrfile-1.1.4.tar.gz
$ cd xdrfile-1.1.4
$ ./configure
$ make
$ make install
```

Po provedení výše uvedených kroků, instalace balíčku xdrfile vytvořila například soubory *libxdrfile.a* a *libxdrfile.la* nutné pro kompilování mnou vytvořeného programu. Tyto soubory se nachází v adresáři /usr/local/lib64.

5.2. Čtení xtc

Příprava pro vytvoření programu *tread-xtc.out* pro čtení *.xtc* souboru začala vytvořením adresářů, ve kterých se program bude nacházet. Hlavním adresářem je *tomasxtc*, ve kterém jsou adresáře *include a src*. Do adresářů *include* jsem zkopíroval soubory xdrfile.h a xdrfile_xtc.h. Do *src* pak xdrfile.c a xdrfile_xtc.c. Tyto soubory jsem zkopíroval ze stejnojmenných adresářů ze stažené knihovny xdrfile-1.1.4. Soubor obsahující celý program pro čtení *.xtc* jsem vytvořil v adresáři *src* a jmenuje se *main.c*. [20, 22]

Soubor *main.c*:

```
#include <stdio.h>
#include <string.h>
#include "../include/xdrfile_xtc.h"
#include "../include/xdrfile.h"

/*
MAIN METHOD
*/
int main (int argc, char *argv[]){
	printf("Hello!\n");
```

```
	XDRFILE * Fxtc;
	FILE * Fgro;
	int result_xtc;
	char * soubor;
	char * souborW;
	int i=0;
	int j=0;
	int natoms;
	int stepI;
	float time;
	float pres;
	matrix box;
	//check number of arguments
	if (argc != 3){
		printf("You didn't write the names of .xtc and .gro file down. Count of argc = %d\n",
argc);
		return -9;
	}else{
		soubor = argv[1];
		souborW = argv[2];
	}
	printf("xtc file is %s\n", soubor);
	printf("gro file is %s\n", souborW);
	result_xtc = read_xtc_natoms(soubor, &natoms);
	Fxtc = xdrfile_open(soubor, "rb");
	//check if files are open
	if (NULL == Fxtc)
		return exdrFILENOTFOUND;
	   if (NULL == (Fgro=fopen(souborW, "w")))
		return exdrFILENOTFOUND;
	rvec *x;
	//x array alocation
	x = (rvec *)calloc(natoms, sizeof(x[0]));
	if (x==NULL) {
		printf("Error allocating memory!\n"); //print an error message
		return 1; //return with failure
	}
	printf("Processing ...\n");
	while(1){
		//read one frame from xtc file
		result_xtc = read_xtc(Fxtc, natoms, &stepI, &time,
		    box, x, &pres);
		//here is the space for analysis – before the data are writen to the file
		//break the while cycle at the end of xtc file
		if (result_xtc == 0){
		}else{
			printf("End of .xtc file\n");
			break;
		}
		//write data to Fgro file
```

```
            fprintf(Fgro, "steps: %d, time: %f, precision: %f\n", stepI,time, pres);
            fprintf(Fgro, "%d\n", natoms);
            //write coordinates to Fgro file
            for (j=0; j<(natoms); j++){
                  fprintf(Fgro, "%d\t%.3f\t%.3f\t%.3f\n", j+1, *(x[(j)]+0), *(x[(j)]+1), *(x[(j)]
+2));
            }
            //write box size to Fgro file
            fprintf(Fgro, "%f\t%f\t%f\n",box[0][0],box[1][1],box[2][2]);
            //fprintf(Fgro, "\n");
      }
      //close the files
          xdrfile_close(Fxtc);
      fclose(Fgro);
      return 0;
}
```

Informace k proměnným, které se objevily v programu:
box – je výstupní proměnnou a je to pole typu float o velikosti 3x3. Proměnná nese záznam o velikosti boxu a nenulové hodnoty jsou pouze na hlavní diagonále.

natoms – je vstupní proměnná pro čtení framů. Hodnota je získána z funkce *read_xtc_natoms*. Hodnota udává počet atomů

pres – v případě čtení *.xtc* souboru je to výstupní proměnná. Hodnota udává přesnost s jakou jsou souřadnice atomů zaznamenány.

x – je výstupní proměnná. Je to pole typu float o velikosti natoms × 3. Toto pole nese polohy všech atomů v daném čase (resp. v daném framu). Do této proměnné jsou již zapisovány dekomprimované souřadnice s požadovanou přesností (v nm).

time – je výstupní proměnná. Hodnota uvádí čas simulace, kdy byl frame pořízen.

stepI – je výstupní proměnnou. Hodnota uvádí krok simulace, ve kterém byl frame pořízen.

Takto vytvořený *main.c* soubor jsem zkompiloval příkazem *make* (příprava makefilu je v kapitole 5.3). *Make* vytvoří spustitelný soubor *tread-xtc.out*, který přijímá dva argumenty. První argument je název *.xtc* souboru a druhým je název souboru, do kterého bude výstup programu *tread-xtc.out* uložen. Soubor uvedený v druhém argumentu je textového formátu.

Takto vytvořený program *main.c* má za úkol otevřít soubor trajektorie, číst postupně jednotlivé framy a uložit do příslušného textového souboru čas, krok a k nim odpovídající pole x a velikost boxu. Celý program je přikládán kompletně se zdrojovými kódy, aby ho bylo možné upravovat k požadovaným fyzikálním analýzám. Předpokládané místo pro vkládání analýzy jsem ve zdrojovém kódu programu vyznačil komentářem.

Soubor *.xtc* se podařilo číst pouze sekvenčně z toho důvodu, že je komprimovaný. Trajektorie do tohoto souboru jsou zapisovány pomocí algoritmu redukování přesností. Souřadnice částic (v nm) jsou násobeny typicky 1000 (pokud není uvedeno jinak) a následně zaokrouhlovány na celá čísla (integer). Poté jsou aplikovány další triky, jako například využití prostorové lokality (atomy ve framu, které jsou v posloupnosti za sebou, jsou také obvykle blízko

v prostoru. Uváděn je příklad molekuly vody).

5.3. Příprava souboru makefile

Pro zkompilování programu *tread-xtc.out* jsem vytvořil makefile, který je umístěný v adresáři *src*. Samotný obsah makefilu byl převzat z webové stránky http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html a následně jsem upravil konkrétní data a cesty k souborům. Aby bylo možné stažené knihovny využít, bylo nutné do příkazu pro kompilaci uvést cestu k souboru *libxdrfile.a*. Cesta k tomuto souboru byla uložena do proměnné *LIBS*. [22]

Soubor *makefile*:

```
#
# 'make depend' uses makedepend to automatically generate dependencies
#               (dependencies are added to end of Makefile)
# 'make'        build executable file 'tread-xtc.out'
# 'make clean'  removes all .o and executable files
#

# define the C compiler to use
CC = gcc

# define any compile-time flags
CFLAGS = -Wall -g

# define any directories containing header files other than /usr/include
#
INCLUDES = -I../include

# define library paths in addition to /usr/lib
#   if I wanted to include libraries not in /usr/lib I'd specify
#   their path using -Lpath, something like:
LFLAGS = -L../lib

# define any libraries to link into executable:
#   if I want to link in libraries (libx.so or libx.a) I use the -llibname
#   option, something like (this will link in libmylib.so and libm.so:
LIBS = /usr/local/lib64/libxdrfile.a

# define the C source files
SRCS = main.c xdrfile.c xdrfile_xtc.c

# define the C object files
#
# This uses Suffix Replacement within a macro:
#   $(name:string1=string2)
#         For each word in 'name' replace 'string1' with 'string2'
# Below we are replacing the suffix .c of all words in the macro SRCS
# with the .o suffix
```

```
#
OBJS = $(SRCS:.c=.o)

# define the executable file
MAIN = tread-xtc.out

#
# The following part of the makefile is generic; it can be used to
# build any executable just by changing the definitions above and by
# deleting dependencies appended to the file from 'make depend'
#

.PHONY: depend clean

all:    $(MAIN)
        @echo  Simple compiler named tread-xtc.out has been compiled

$(MAIN): $(OBJS)
        $(CC) $(CFLAGS) $(INCLUDES) -o $(MAIN) $(OBJS) $(LFLAGS) $(LIBS)

# this is a suffix replacement rule for building .o's from .c's
# it uses automatic variables $<: the name of the prerequisite of
# the rule(a .c file) and $@: the name of the target of the rule (a .o file)
# (see the gnu make manual section about automatic variables)
.c.o:
        $(CC) $(CFLAGS) $(INCLUDES) -c $<  -o $@

clean:
        $(RM) *.o *~ $(MAIN)

depend: $(SRCS)
        makedepend $(INCLUDES) $^

# DO NOT DELETE THIS LINE -- make depend needs it
```

6. Závěr

Prvním krokem diplomové práce byl výběr vhodné grafické karty pro účely paralelních výpočtů v programu Gromacs a návrh kompletního hardwaru pro tuto GPU. Do tohoto nového stroje byl nainstalován operační systém OpenSUSE 13.2, potřebné ovladače grafické karty, CUDA, ICC kompilátor, program Gromacs a další. Nainstalované programy byly nastaveny a také byly napsány skripty pro nastavení cest různých verzí programu Gromacs. V diplomové práci bylo také popsáno, jak využít GPU k výpočtům v Gromacs a s jakými parametry a volbami se simulace spouští. Dále byly popsány některé rozdíly oproti starším verzím tohoto programu.

Důležitým bodem bylo testování zrychlení a výhodnosti GPU při molekulárním modelování. Bylo zjištěno, že simulace s GPU jsou 1.2 - 3.7× rychlejší oproti použití samotného CPU a je při nich vhodné použít GCC kompilátor. Naopak u simulací pouštěných pouze na CPU je vhodnější použít ICC kompilátor.

Na počítači Gram u malého systému vody byla pozorována jako nejrychlejší konfigurace 1 GPU a 16 CPU vláken. S rostoucí velikostí systému postupně bylo výhodnější přidat další GPU a na největším testovaném systému w1000 využít veškerých prostředků, kterými stroj disponoval, tedy spustit simulaci na 4 GPU a 16 CPU vláknech. Je-li požadavkem spustit na stroji více než jednu úlohu, bylo nejefektivnější využití stroje při rozdělení jeho prostředků mezi počítané úlohy a spustit dvě, nebo čtyři úlohy zároveň.

Další testy ukázaly, že změna délky cutoffu u simulací s GPU neměla velký vliv na dobu výpočtu simulace jako u CPU. Pro velké délky cutoffu se proto zvyšuje výhodnost GPU.

Na systému argonu pak byl ukázán případ, kdy GPU byla přetěžována a zvýšením počtu grafických karet se výpočet výrazně zrychlil. Čím byl větší systém argonu, tím bylo toto zrychlení výraznější (toto platí od velikosti systému 10k argon). Ovšem očekávané enormní zrychlení na tomto systému oproti CPU se nepotvrdilo.

Z celkové analýzy výpočtů na GPU a CPU provedené v diplomové práci, s ohledem na rychlost výpočtů i cenu počítače bez a s GPU, vyplývá, že nákup a použití GPU se při výpočtech v programu Gromacs vyplatí.

Nakonec byly nainstalovány knihovny pro čtení *.xtc* souborů. Tyto knihovny byly použity k vytvoření rozhraní s *.xtc* souborem. Pomocí tohoto rozhraní bude moci vedoucí práce dle konkrétních požadavků trajektorie dále analyzovat.

Výsledkem této práce je kromě sepsané diplomové práce plně funkční počítač s výpočetní GPU, pro který jsem navrhl komponenty, sestavil jej a nainstaloval vše potřebné pro jeho využití pro simulace v programu Gromacs s využitím GPU.

Bibliografie

[1] Desktop GPUs. GeForce.com [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://www.geforce.com/hardware/desktop-gpus

[2] Video Card Benchmarks. Videocardbenchmark.net [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://www.videocardbenchmark.net/

[3] GeForce GTX 780: Performance. Geforce.co.uk [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://www.geforce.co.uk/hardware/desktop-gpus/geforce-gtx-780/performance

[4] Gromacs. Gromacs.org [online]. 2015 [cit. 2015-12-09]. Dostupné z: www.gromacs.org

[5] Installing Intel Compilers on OpenSUSE. Software.intel.com [online]. 2011 [cit. 2015-12-09]. Dostupné z: https://software.intel.com/en-us/articles/installing-intel-compilers-on-opensuse

[6] GROMACS 4.6 Installation Guide. Dqfnet.ufpe.br [online]. 2013 [cit. 2015-12-09]. Dostupné z: http://dqfnet.ufpe.br/groups/geekstuff/wiki/2ebb2/GROMACS_46_Installation_Guide.html

[7] Installation guide for GROMACS 5.0.7. Gromacs.org [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://www.gromacs.org/Documentation/Installation_Instructions_5.0

[8] CompuBench 1.5 Desktop. Compubench.com [online]. 2015 [cit. 2015-12-09]. Dostupné z: https://compubench.com/

[9] PCI Express Interface. Trentonsystems.com [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://www.trentonsystems.com/applications/pci-express-interface/

[10] PCI Express 3.0: Build for speed. Tomshardware.com [online]. 2010 [cit. 2015-12-09]. Dostupné z: http://www.tomshardware.com/reviews/pci-express-3.0-pci-sig,2695-3.html

[11] LINDAHL, Erik a Szilárd PÁLL. Parallelization schemes & GPU Acceleration [online]. 2013, 57 [cit. 2015-12-09]. Dostupné z: http://www.gromacs.org/@api/deki/files/213/=gromacs_parallelization_acceleration.pdf

[12] BARVÍKOVÁ, Hana. Počítačové modelování interakcí molekul s minerálními povrchy. České Budějovice, 2012. Bakalářská práce. Jihočeská univerzita v Českých Budějovicích. Vedoucí práce RNDr. Milan Předota, Ph.D.

[13] Acceleration and parallelization. Gromacs.org [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://www.gromacs.org/Documentation/Acceleration_and_parallelization

[14] KUTZNER, Carsten, Szilárd PÁLL, Martin FECHNER, Ansgar ESZTERMANN, Bert L. DE GROOT a Helmut GRUBMÜLLER. Best bang for your buck: GPU nodes for GROMACS biomolecular simulations. J. Comput. Chem. [online]. 2015, 36 [cit. 2015-12-09]. DOI: 10.1002/jcc.24030. Dostupné z: http://onlinelibrary.wiley.com/doi/10.1002/jcc.24030/full

[15] Getting good performance from mdrun. Gromacs.org [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://manual.gromacs.org/documentation/5.1-rc1/user-guide/mdrun-

performance.html#gromacs-background-information

[16] ABRAHAM, Mark James, Teemu MURTOLA, Roland SCHULZ, Szilárd PÁLL, Jeremy C. SMITH, Berk HESS a Erik LINDAHL. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers [online]. 2015, 25 [cit. 2015-12-09]. DOI: 10.1016/j.softx.2015.06.001. Dostupné z: http://www.sciencedirect.com/science/article/pii/S2352711015000059

[17] CMake. Wikipedia.org [online]. 2015 [cit. 2015-12-09]. Dostupné z: https://cs.wikipedia.org/wiki/CMake

[18] CMake. Cmake.org/ [online]. 2015 [cit. 2015-12-09]. Dostupné z: https://cmake.org

[19] How to start computing - tutorial. Wiki.metacentrum.cz [online]. 2014 [cit. 2015-12-09]. Dostupné z: https://wiki.metacentrum.cz/wiki/How_to_start_computing_-_tutorial

[20] How to write a xtc file in c. Mailman-1.sys.kth.se [online]. 2006 [cit. 2015-12-09]. Dostupné z: https://mailman-1.sys.kth.se/pipermail/gromacs.org_gmx-developers/2006-August/001752.html

[21] XTC Library. Gromacs.org [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://www.gromacs.org/Developer_Zone/Programming_Guide/XTC_Library

[22] How to compile C++ propgram with xdrfile library? Mail-archive.com [online]. 2009 [cit. 2015-12-09]. Dostupné z: https://www.mail-archive.com/gmx-users@gromacs.org/msg21523.html

[23] NVIDIA CUDA Getting Started Guide for Linux. Docs.nvidia.com [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/index.html#verify-you-have-cuda-enabled-system

[24] KUTZNER, Carsten, Szilárd PÁLL, Martin FECHNER, Martin ESZTERMANN, Ansgar ESZTERMANN a Bert L. DE GROOT. Best Bang for Your Buck: GPU Nodes for GROMACS Biomolecular Simulations [online]. Journal of Computational Chemistry, 2015, 19 [cit. 2015-12-09]. DOI: 10.1002/jcc.24030. Dostupné z: https://www.mpibpc.mpg.de/15070156/Kutzner_2015_JCC.pdf

[25] Xtc file format. Manual.gromacs.org [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://manual.gromacs.org/online/xtc.html

[26] Drivers. Geforce.com [online]. 2015 [cit. 2015-12-09]. Dostupné z: http://www.geforce.com/drivers

Seznam tabulek

Seznam obrázků

Příloha

Parametry GeForce GTX 780

Parametry GeForce GTX 780 code name: GK110	Hodnota
GPU Engine Specs:	
CUDA Cores	2304
Base Clock (MHz)	863
Boost Clock (MHz)	900
Texture Fill Rate (billion/sec)	160,5
Memory Specs:	
Memory Speed	6,0 Gbps
Standard Memory Config	3072 MB
Memory Interface	GDDR5
Memory Interface Width	384-bit
Memory Bandwidth (GB/sec)	288,4
Support:	
Important Technologies	GPU Boost 2.0, PhysX, TXAA, NVIDIA G-SYNC-ready,
Other Supported Technologies	3D Vision, CUDA, Adaptive VSync, FXAA
OpenGL	4,3
Bus Support	PCI Express 3.0
Certified for Windows 7, Windows 8, Windows Vista, or Windows XP	Yes
3D Vision Ready	Yes
3D Gaming	Yes
Blu Ray 3D	Yes
3D Vision Live (Photos and Videos)	Yes
NVIDIA PhysX™ Technology	Yes
Microsoft DirectX	12 API with Feature Level 12_1
Display Support:	
Maximum Digital Resolution1	4096x2160
Maximum VGA Resolution	2048x1536
Standard Display Connectors	One Dual Link DVI-I, One Dual Link DVI-D, One HDMI, One
Multi Monitor	4 displays
HDCP	Yes
HDMI	Yes
Audio Input for HDMI	Internal
Graphics Card Dimensions:	
Height	4,376 inches
Length	10,5 inches
Width	Dual-slot
Thermal and Power Specs:	
Maximum GPU Temperature (in C)	95 C
Graphics Card Power (W)	250 W
Minimum Recommended System Power (W)	600 W
Supplementary Power Connectors	One 8-pin and one 6-pin

Submitovací skript pro Gram

```
#!/bin/sh
#PBS -q gpu
#PBS -l walltime=30:00
#PBS -l nodes=1:ppn=16:gpu=1:home_gram
#PBS -l mem=16gb
#PBS -l scratch=16gb
#PBS -j oe
hostname
cat $PBS_NODEFILE

# setting the automatical cleaning of the SCRATCH
# (unless set otherwise, the data will be deleted)
trap 'clean_scratch' TERM EXIT

# set the working directory
WORK=vypocty/gromacsvypocty/watergpu-cpu/w1
SOURCE=/storage/plzen1/home/krejst00

# input data copy
cp $SOURCE/$WORK/* $SCRATCHDIR  || exit 1

# starting computation in the working directory
cd $SCRATCHDIR

# adding the module
module add cuda-6.5
module add gromacs-5.0.5-gpu
#grompp -c w1.gro -p w1.top -f md.mdp -o w1.tpr

# execution
mdrun -nb cpu -ntmpi 1 -ntomp 16 -deffnm w1
cp w1.log w1_GccGcc_5
# copy results from the scratch  (if there is something wrong, leave data in SCRATCHDIR and
informes the user)
cp $SCRATCHDIR/* $SOURCE/$WORK/ || export CLEAN_SCRATCH=false
```

Submitovací skript pro MP-8

```
#!/bin/bash
#PBS -N w1
#PBS -l nodes=1:ppn=4:mp8
#PBS -j oe
cat $PBS_NODEFILE

WORK=computeFolder/watergpu-cpu/w1          #adresář s konkrétním molekulárním systémem
SOURCE=/home/krejsa             #domovský adresář
SCRATCH=/local/krejsa/$WORK
rm -r $SCRATCH
mkdir -p $SCRATCH    #dočasný adresář na stroji, kde se bude provádět simulace
```

```
cp -r $SOURCE/$WORK/* $SCRATCH
cd $SCRATCH

source /opt/gromacs-5.0.5/bin_gpu/bin/GMXRC          #cesta k GPU verzi Gromacsu na MP 8
mdrun -nb cpu -deffnm w1
cp -r $SCRATCH/* $SOURCE/$WORK
rm -r $SCRATCH
```

Čtení xtc souboru – main.c

```
#include <stdio.h>
#include <string.h>
#include "../include/xdrfile_xtc.h"
#include "../include/xdrfile.h"

/*
MAIN METHOD
*/
int main (int argc, char *argv[]){
	printf("Hello!\n");
	XDRFILE * Fxtc;
	FILE * Fgro;
	int result_xtc;
	char * soubor;
	char * souborW;
	int i=0;
	int j=0;
	int natoms;
	int stepI;
	float time;
	float pres;
	matrix box;
	//check number of arguments
	if (argc != 3){
		printf("You didn't write the names of .xtc and .gro file down. Count of argc = %d\n",
argc);
		return -9;
	}else{
		soubor = argv[1];
		souborW = argv[2];
	}
	printf("xtc file is %s\n", soubor);
	printf("gro file is %s\n", souborW);
	result_xtc = read_xtc_natoms(soubor, &natoms);
	Fxtc = xdrfile_open(soubor, "rb");
	//check if files are open
	if (NULL == Fxtc)
		return exdrFILENOTFOUND;
	   if (NULL == (Fgro=fopen(souborW, "w")))
		return exdrFILENOTFOUND;
```

```
    rvec *x;
    //x array alocation
    x = (rvec *)calloc(natoms, sizeof(x[0]));
    if (x==NULL) {
        printf("Error allocating memory!\n"); //print an error message
        return 1; //return with failure
    }
    printf("Processing ...\n");
    while(1){
        //read one frame from xtc file
        result_xtc = read_xtc(Fxtc, natoms, &stepI, &time,
            box, x, &pres);
        //here is the space for analysis – before the data are writen to the file
        //break the while cycle at the end of xtc file
        if (result_xtc == 0){
        }else{
            printf("End of .xtc file\n");
            break;
        }
        //write data to Fgro file
        fprintf(Fgro, "steps: %d, time: %f, precision: %f\n", stepI,time, pres);
        fprintf(Fgro, "%d\n", natoms);
        //write coordinates to Fgro file
        for (j=0; j<(natoms); j++){
            fprintf(Fgro, "%d\t%.3f\t%.3f\t%.3f\n", j+1, *(x[(j)]+0), *(x[(j)]+1), *(x[(j)]+2));
        }
        //write box size to Fgro file
        fprintf(Fgro, "%f\t%f\t%f\n",box[0][0],box[1][1],box[2][2]);
        //fprintf(Fgro, "\n");
    }
    //close the files
      xdrfile_close(Fxtc);
    fclose(Fgro);
    return 0;
}
```

Čtení xtc souboru – makefile

```
#
# 'make depend' uses makedepend to automatically generate dependencies
#               (dependencies are added to end of Makefile)
# 'make'        build executable file 'tread-xtc.out'
# 'make clean'  removes all .o and executable files
#

# define the C compiler to use
CC = gcc

# define any compile-time flags
CFLAGS = -Wall -g
```

```
# define any directories containing header files other than /usr/include
#
INCLUDES = -I../include

# define library paths in addition to /usr/lib
#   if I wanted to include libraries not in /usr/lib I'd specify
#   their path using -Lpath, something like:
LFLAGS = -L../lib

# define any libraries to link into executable:
#   if I want to link in libraries (libx.so or libx.a) I use the -llibname
#   option, something like (this will link in libmylib.so and libm.so:
LIBS = /usr/local/lib64/libxdrfile.a

# define the C source files
SRCS = main.c xdrfile.c xdrfile_xtc.c

# define the C object files
#
# This uses Suffix Replacement within a macro:
#   $(name:string1=string2)
#         For each word in 'name' replace 'string1' with 'string2'
# Below we are replacing the suffix .c of all words in the macro SRCS
# with the .o suffix
#
OBJS = $(SRCS:.c=.o)

# define the executable file
MAIN = tread-xtc.out

#
# The following part of the makefile is generic; it can be used to
# build any executable just by changing the definitions above and by
# deleting dependencies appended to the file from 'make depend'
#

.PHONY: depend clean

all:    $(MAIN)
        @echo  Simple compiler named tread-xtc.out has been compiled

$(MAIN): $(OBJS)
        $(CC) $(CFLAGS) $(INCLUDES) -o $(MAIN) $(OBJS) $(LFLAGS) $(LIBS)

# this is a suffix replacement rule for building .o's from .c's
# it uses automatic variables $<: the name of the prerequisite of
# the rule(a .c file) and $@: the name of the target of the rule (a .o file)
# (see the gnu make manual section about automatic variables)
.c.o:
```

```
	$(CC) $(CFLAGS) $(INCLUDES) -c $<  -o $@

clean:
	$(RM) *.o *~ $(MAIN)

depend: $(SRCS)
	makedepend $(INCLUDES) $^

# DO NOT DELETE THIS LINE -- make depend needs it
```