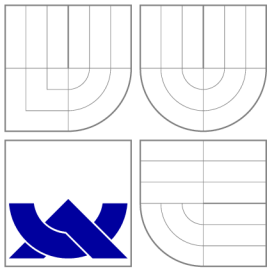
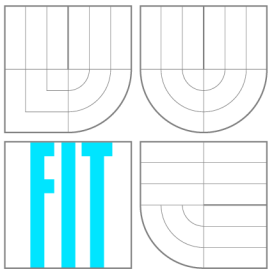


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# MINIATURNÍ VÝPOČETNÍ CLUSTER SLOŽENÝ Z MIKROKONTROLÉRŮ

A MINI-CLUSTER BASED ON MICROCONTROLLER COMPUTING NODES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

BOLESLAV ŠÍDLO

VEDOUCÍ PRÁCE
SUPERVISOR

ing. MICHAL BIDLO, Ph.D.

BRNO 2016

*Zadání bakalářské práce/18261/2015/xsidlo02*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačových systémů Akademický rok 2015/2016

# Zadání bakalářské práce

Řešitel: **Šídlo Boleslav**
Obor: Informační technologie
Téma: **Miniaturní výpočetní cluster složený z mikrokontrolerů**
**A Mini-Cluster Based on Microcontroller Computing Nodes**
Kategorie: Vestavěné systémy

Pokyny:

1. Nastudujte problematiku programování a možnosti vzájemného propojování a komunikace mikrokontrolérů. Proveďte rešerši v této oblasti a vytipujte vhodné modely mikrokontrolérů či vývojových kitů s ohledem na požadované komunikační rozhraní a periferní výbavu.
2. Zvolte vhodnou platformu (založenou na mikrokontroléru určité rodiny z rešerše provedené v bodu 1) a navrhněte výpočetní platformu založenou na kooperaci paralelně pracujících výpočetních uzlů.
3. Realizujte prototyp výpočetní platformy s využitím alespoň čtyř výpočetních uzlů reprezentovaných mikrokontroléry.
4. S ohledem na výpočetní možnosti navržené platformy zvolte a implementujte vhodnou úlohu s cílem paralelizace jejího řešení na této platformě. Srovnejte vlastnosti této implementace se sekvenčním řešením.
5. Zhodnoťte dosažené výsledky, diskutujte výhody a nevýhody navržené platformy a potenciální možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího projektu.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání, demonstrace funkčního prototypu alespoň jednoho výpočetního uzlu z bodu 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bidlo Michal, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
*vedoucí ústavu*

# Abstrakt

Cílem této práce je prozkoumat možnosti využití miniaturního výpočetního clusteru, složeného z jednoduchých mikrokontrolérů, pro paralelní výpočty. Práce zkoumá chování tohoto výpočetního clusteru při řešení různých typů úloh, popisuje jeho možnosti a omezení. Pokusy byly prováděny na výpočetním clusteru tvořeném čtyřmi vývojovými deskami, které byly osazeny 8–bitovými čipy a komunikovaly přes I2C rozhraní. Výsledkem pokusných měření je srovnání rychlosti výpočtu při použití jednoho mikrokontroléru a při použití clusteru. Experimentálně bylo zjištěno, že v případě aplikací, které nevyžadující velký objem přenášených dat, lze dosáhnout výrazného urychlení. Dále se potvrdil předpoklad, že takto jednoduché mikrokontroléry nejsou vhodné pro výpočty s desetinnými čísly, které vyžadují velkou přesnost.



# Abstract

The objective of this bachelor thesis is to investigate a low-cost computing cluster, composed of microcontrollers-based nodes, for parallel computing tasks. The work deals with the behaviour and limitations of the platform in various situations. Experiments were performed using 4 development boards equipped with 8-bit microcontrollers. I2C serial interface was used for the communication between the nodes. The experiments were devoted to the comparison of computing times of a sequential algorithm (running on a single minrocontroller only) and the parallel version using the cluster. The results showed that the cluster can speed-up the computation of applications that does not require a high communication overhead. Moreover, the microcontrollers applied showed as unsuitable for floating-point computing if a high accuracy of the results is required.



# Klíčová slova

Mikrokontrolér, cluster, paralelní výpočty.

# Keywords

Microcontroller, cluster, parallel computing.


# Citace

Boleslav Šídlo: Miniaturní výpočetní cluster složený
z mikrokontrolérů, bakalářská práce, Brno, FIT VUT v Brně, 2016

# Miniaturní výpočetní cluster složený z mikrokontrolérů

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doktora Michala Bidla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Boleslav Šídlo
13. května 2016

## Poděkování

Chtěl bych velice poděkovat vedoucímu své bakalářské práce za veškerý věnovaný čas, poskytnuté cenné rady a inspirující podněty.

# Obsah

# Seznam obrázků

# Seznam tabulek

# Kapitola 1

# Úvod

V současné době dochází ke stále většímu rozmachu výpočetních systémů. Stále více různých typů zařízení v sobě obsahuje řídicí počítačovou jednotku, přičemž začíná být stále více důležité, aby tato zařízení spolu dokázala komunikovat a spolupracovat, což následně vede k nutnosti zkoumání chování těchto zařízení při paralelních výpočtech.

Byly například prováděny experimenty s paralelním zapojením 8 výpočetních platforem Raspberry Pi [13] do clusteru. Každá jedna tato platforma poskytuje sama o sobě vysoký výpočetní výkon a při spojení do clusteru se její možnosti dále rozšiřují. Otázkou pak je, jestli je možné (při zapojení do clusteru) dosáhnout podobných výsledků i při použití mnohem jednodušších a levnějších platforem.

Cílem této práce je prozkoumat možnosti velice jednoduchých a levných 8 bitových mikrokontrolérů pro použití při paralelních výpočtech, stanovit podmínky, za kterých má takováto paralelizace smysl, upozornit na omezení a nedostatky tohoto řešení a nastínit možnosti dalšího rozvoje tohoto tématu.

# Kapitola 2

# Vestavěné systémy na bázi mikrokontrolérů

V této kapitole je shrnuta základní problematika týkající se vestavěných systémů. Jelikož se jedná o velice rozsáhlé téma, je zde uvedeno pouze stručné shrnutí.

## 2.1 Vestavěné systémy

Pro potřeby této práce budeme pod pojmem *vestavěný systém* uvažovat následující definici: [26]

„*Vestavěné systémy (VS) jsou systémy, ve kterých je zpracování dat vestavěno/vloženo do většího systému a ve kterém není zpracování dat viditelné uživateli prostřednictvím např. PC počítače. Anglicky se vestavěné systémy označují jako* ***embedded system***“.

Vzhledem k tomu, že *vestavěné systémy* se neustále vyvíjejí a neustále přibývá velké množství rozličných zařízení tohoto typu, není tato definice jediná a lze nalézt i mnoho dalších, přičemž žádná z nich nedokáže toto téma jednoznačně a komplexně shrnout.

Obecně lze *vestavěné systémy* popsat jako elektronická, případně elektromechanická zařízení, která mají velice úzce specifikovaný rozsah svého použití, nedisponují žádným, případně velice strohým uživatelským rozhraním a svoji svěřenou výpočetní úlohu vykonávají z hlediska spotřeby výpočetních zdrojů efektivněji, než by to dokázaly univerzální výpočetní stroje, jakými jsou například osobní počítače (PC).

V dnešní moderní době se s *vestavěnými systémy* setkáváme prakticky všude. Může se jednat o domácí spotřebiče, různé druhy počítačových periferií, nebo třeba parkovací automat a mnoho dalších zařízení. Přičemž současný trend je takový, že počet těchto zařízení neustále narůstá. Proto má smysl se dále věnovat výzkumu těchto systémů a to na všech úrovních.

Jádrem každého *vestavěného systému* je řídicí jednotka, která má typicky na starost veškeré výpočetní úlohy. Tuto jednotku obvykle tvoří mikrokontrolér.

## 2.2 Mikrokontroléry

Pojmem mikrokontrolér (MCU), nebo také jednočipový počítač, v sobě zahrnuje značné množství typů výpočetní techniky s velkými možnostmi využití a proto není snadné tento

pojem jednoduše obsáhnout. Jako obvyklá definice se uvádí, že mikrokontrolér je složen z CPU a periferií, které jsou všechny uloženy na jednom čipu [8]. Mikrokontrolérem může být velice jednoduchý počítačový čip disponující pouze tou nejzákladnější funkcionalitou a používaný na ty nejjednodušší úlohy. Ale také to může být velice výkonný a komplexní prostředek umožňující řízení složitých aplikací. [26] Z hlediska dostupných výpočetních zdrojů lze najít velice „skromné“ čipy s operační pamětí v řádu jednotek bytů a minimální funkcionalitou, jejichž cena nepřasahuje $0,5\,\$$ [21], ale také mimořádně výkonné čipy, které disponují značnou výpočetní silou a mnoha rozšiřujícími moduly [3].

Ačkoliv jsou mikrokontroléry značně různorodé, mají společnou jednu vlastnost a tou je efektivita využití zdrojů. Nejedná se o naprosto univerzální zařízení, které potenciálně zvládne úplně všechny typy úloh, ale naopak je každý mikrokontrolér specializovaný výpočetní stroj, který je často určen k vykonávání jediné činnosti po celou dobu své životnosti. Tato specializace umožňuje, aby byly mikrokontroléry vysoce efektivní nejen z hlediska rychlosti výpočtu dané úlohy, ale také rozměrů, příkonu a ceny. Výběr toho nejvhodnějšího mikrokontroléru pro danou aplikaci pak představuje netriviální úlohu a k jejímu řešení je třeba znát důkladně nejen tuto aplikaci, ale také možnosti různých mikrokontrolérů [22].

### 2.2.1 Technické aspekty mikrokontrolérů

V případě rozboru technických aspektů mikrokontroléru je obvyklé popsat použitou architekturu z hlediska tří nejvýznamnějších vlastností: organizace paměti, realizace instrukční sady CPU a dostupné periferie.

#### Organizace paměti

Stejně jako u ostatních počítačů, vychází schéma mikrokontroléru ze dvou základních architektur. Jedná se o architekturu *von Neumann* a architekturu *Harvard* [19]. Hlavním rozdílem těchto dvou architektur je způsob práce s pamětí.

V případě **Von Neumann** architektury je použit jeden společný paměťový prostor pro aplikační data, tak pro samotný program. Toto řešení má výhodu snadnější fyzické realizace počítačového čipu, protože není potřeba vytvářet adresové piny pro obě paměti zvlášť.

V případě **Harvard** architektury dochází k důslednému oddělení pamětí aplikačních dat a samotného programu. Počítač pracuje se dvěma naprosto oddělenými paměťovými prostory, kde každý má svoji vlastní adresaci. Výhoda tohoto přístupu spočívá v tom, že oddělené paměti mohou být implementovány naprosto odlišně. To poté vede k optimalizaci parametrů (například kapacita, rychlost) obou pamětí z hlediska předpokládaného použití.

Zatímco v případě klasických univerzálních osobních počítačů se téměř bez výhrad prosadila architektura *von Neumann*, v případě mikrokontrolérů není situace vůbec jednoznačná. V současné době se vyrábí velké množství různých druhů mikrokontrolérů a lze mezi nimi najít zástupce obou dvou architektur [26].

#### Instrukční sada CPU

Instrukční sada centrální výpočetní jednotky může být dvojího druhu [24].

**CISC**, neboli *Complex Instruction Set Computer*, je, jak už její název napovídá velice rozsáhlá a komplexní instrukční sada, která obsahuje velkou řadu různých operací, často včetně pokročilých matematických funkcí jako například odmocnina.

**RISC**, neboli *Reduced Instruction Set Computer*, naopak obsahuje pouze omezenou sadu těch nejnutnějších instrukcí a veškeré pokročilejší operace musí být realizovány sofwarově pomocí těchto instrukcí.

Na první pohled by se mohlo zdát, že neexistuje rozumný důvod pro výrobu RISC CPU. Opak je ale pravdou. Implementace CISC architektury je totiž mnohem náročnější na výrobu. Další její nevýhodou je, že komplikované instrukce se obvykle vykonávají několikrát déle, než ty jednoduché. A vzhledem k tomu, že v dnešní době jsou již málokteré aplikace psány přímo ve strojovém kódu, nemusí se tyto instrukce nikdy použít, protože dostupné kompilátory nemusí být schopné s nimi náležitě pracovat. To poté může vést k tomu, že veškeré prostředky věnované implementaci takovéto komplexní instrukce přijdou vniveč.

Naproti tomu technologie *RISC* je obvykle využívána velice efektivně, neboť prakticky všechny implementované instrukce jsou využívány. Dále je u *RISC* obvyklé, že všechny instrukce jsou zhruba stejně složité, což v kombinaci s vhodným zřetězením operací (*pipelining*) vede vysoké efektivitě využití CPU.

Z těchto důvodů má technologie *RISC* své nezastupitelné místo při výrobě počítačových čipů a v případě mikrokontrolérů naprosto dominuje.

### Periferie mikrokontroléru

Přítomnost periferních obvodů je to, co mikrokontrolér odlišuje od ostatních počítačových čipů. Existuje velice široká škála nejrůznějších periferních modulů, kterými může být mikrokontrolér vybaven. Zde jsou ve stručnosti vyjmenovány ty nejobvyklejší z nich [8] :

- **Analogově–digitální převodník**
  Slouží pro převod hodnoty analogového signálu do digitální číselné reprezentace. Typicky se používá pro zpracování a uložení naměřených údajů z připojených analogových senzorů.

- **Digitálně–analogový převodník**
  Umožňuje převést digitální číselnou reprezentaci proměnné uložené v paměti mikrokontroléru na analogovou hodnotu a přesměrovat ji na jeden z analogových výstupních pinů. Tento modul najde uplatnění při práci s připojenými zařízeními s analogovým ovládáním. Mikrokontrolér tak může fungovat například jako potenciometr.

- **Řadič přerušení**
  Řadič přerušení umožňuje spravovat různé typy přerušení, jak softwarová, tak hardwarová, která jsou generována připojenými zařízeními.

- **Časovač**
  Je modul, který je obvykle určen pro generování periodického impulzního signálu, případně k detekci změn vstupního signálu a odměřování časových úseků. Dále je možné časovač použít pro generování *PWM (Pulse Width Modulation)* signálu.

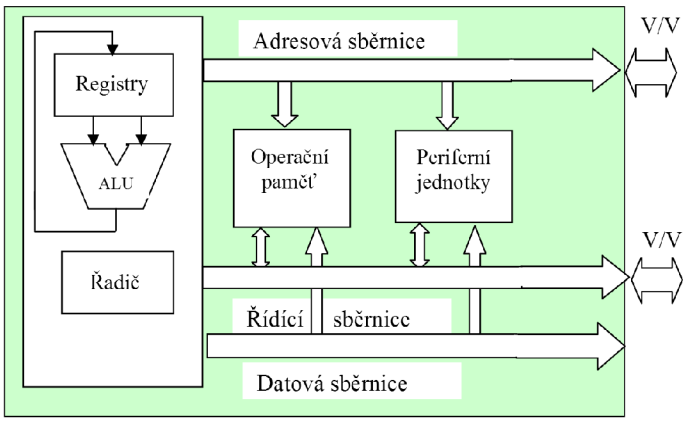
- **Watchdog**
  Hlídací zařízení, které lze použít ke kontrole, jestli nedošlo k zacyklení programu. Pakliže bylo zacyklení detekováno, *watchdog* resetuje běh programu.

- **Moduly pro sériová komunikační rozhraní**
  Sériových komunikačních rozhraní je více. Mezi ty, kterými jsou mikrokontroléry vybaveny nejčastěji, patří *SPI* [7], *I2C* [16], *TWI* [5] a u pokročilejších mikrokontrolérů i *USB* [9].

Tento seznam rozhodně není úplný. Různé druhy mikrokontrolérů mohou disponovat širokou škálou různých periferií a pro kompletní přehled nabízené funkcionality je třeba nastudovat technickou dokumentaci příslušného zařízení. Některé z výše zmíněných periferií budou podrobněji popsány v následujících kapitolách.

#### Obecné schéma mikrokontroléru

Na obrázku 2.1 je vidět obecná struktura jednočipového počítače. Registry, ALU a řadič tvoří centrální výpočetní jednotku, která prostřednictvím adresové, datové a řídicí sběrnice komunikuje s operační pamětí, periferiemi a vstup–výstupními zařízeními




Obrázek 2.1: Struktura mikrokontrolérů
(obrázek převzat z [26])

### 2.2.2 Obvyklé využití mikrokontrolérů

Mikrokontroléry nejčastěji najdou uplatnění při řešení problémů, u kterých je třeba vykonávat jednu konkrétní specializovanou úlohu a to často bez jakéhokoliv zásahu člověka. V případě mikrokontrolérů se nepředpokládá častá změna programu, ve většině případů se s ní dokonce nepočítá vůbec. Naopak se očekává co nejdelší možná životnost celého zařízení, při minimálních zásazích zvenčí.

Obvyklé uplatnění mikrokontrolérů je právě ve vestavěných systémech, které mají jasným a neměnným způsobem daný svůj účel. Vestavěné systémy jsou obvykle vyráběny ve velkých počtech a proto je třeba maximálním možným způsobem zefektivnit využití všech zdrojů. Do tohoto konceptu mikrokontroléry dokonale zapadají. Mikrokontrolérů se vyrábí nepřeberné množství různých druhů a téměř vždy je možné nalézt takový, který svými parametry vyhovuje pro danou úlohu a řeší ji efektivně [26].

### 2.2.3 Budoucnost mikrokontrolérů

V současné době je uplatnění mikrokontrolérů skutečně široké a lze očekávat, že jejich význam se bude neustále zvětšovat. Vestavěné systémy na bázi mikrokontrolérů totiž pronikají do stále většího množství oborů lidské činnosti.

Oproti klasickým univerzálním počítačům totiž mikrokontroléry disponují řadou nesporných výhod. Nemají sice tak velký výpočetní výkon, ale to obvykle vůbec ničemu nevadí, protože na mikrokontroléru standardně běží pouze jedna aplikace, na rozdíl od klasických

počítačů, kde bývá paralelně spuštěno několik aplikací s desítkami procesů a navíc ještě s operačním systémem. Hlavní síla mikrokontrolérů spočívá v malých rozměrech a nízkém příkonu, což z nich dělá ideální výpočetní jednotky pro mobilní zařízení napájená z baterií. Přičemž těchto mobilních zařízení neustále přibývá.

Na tomto místě je vhodné krátce zmínit *ARM* (*Advanced RISC Machine*) architekturu. Vestavěné systémy, založené na *ARM* architektuře mají totiž velký potenciál [12]. V posledních letech zažívají obrovský rozmach a to právě díky svým téměř ideálním vlastnostem. *ARM* zařízení v sobě totiž kombinují vysoký výpočetní výkon s relativně nízkým příkonem a nároky na chlazení.

# Kapitola 3

# Výpočetní cluster realizovaný pomocí mikrokontrolérů

## 3.1 Požadavky na výsledné zařízení

Ještě než bylo možno přistoupit k samotné fyzické realizaci výpočetního clusteru složeného z mikrokontrolérů, bylo potřeba si jasně stanovit kritéria a podmínky podle kterých bude vybírán příslušný hardware.

### 3.1.1 Požadavky na mikrokontrolér

Pro potřeby této práce budeme uvažovat cluster o čtyřech *MCU*, které mají být vybaveny co možná nejjednoduššími a nejlevnějšími mikrokontroléry. Vzhledem k tomu, že všechny uzly mají pracovat paralelně je nezbytné, aby vybraný mikrokontrolér byl vybaven vhodným komunikačním rozhraním. Také bylo velice důležité vybrat vhodnou architekturu, tedy takovou, která by byla co nejjednodušší a s co nemenšími nároky na spotřebu zdrojů.

Na základě těchto požadavků byla stanovena prioritní kritéria pro výběr vhodného mikrokontroléru:

- cena,
- jednoduchost architektury,
- dostupná komunikační rozhraní,
- dostupnost vhodné vývojové platformy.

Z hlediska jednoduchosti a minimální ceny je nejvhodnější 8–bitová architektura. Co se týče běžně používaných komunikačních rozhraní, tak mezi ta která jsou velice jednoduchá, ale zároveň dostatečně výkonná patří rozhraní *SPI*, *I2C* a *TWI* [1].

Při výběru vhodného mikrokontroléru bylo potřeba zohlednit i další kritéria, která sice už neměla takovou důležitost, jako ta výše zmíněná, ale též jim byl přikládán nezanedbatelný význam. Konkrétně se jedná o tyto:

- kapacita paměti pro uložení programu,

[1] *SPI* a *TWI* rozhraní jsou v podstatě jedno a totéž, pouze s jiným názvem. Podrobněji budou tato rozhraní rozebrána v pozdější části této kapitoly

- kapacita operační paměti.

Kapacita paměti pro uložení programu je důležité kritérium, které ovlivňuje, jak složitý algoritmus bude možné na daném mikrokontroléru implementovat a provozovat. U této kategorie byl stanoven dolní limit 20 KB. Tato kapacita by měla postačovat pro otestování i poměrně komplexních algoritmů.

Kapacita operační paměti limituje množství dat, která může vykonávaný program při svém běhu zpracovávat. Vzhledem k tomu, že většina jednoduchých mikrokontrolérů není určena pro práci s velkým objemem dat, tak kapacita jejich operační paměti se obvykle pohybuje v řádu 100 bytů až jednotek KB. Pro potřeby této práce byla jako dolní mez kapacity operační paměti stanovena hodnota 2 KB.

Co se týče vybavenosti periferiemi, nebyl stanoven žádný požadovaný limit ani standard, jedná se spíše o vedlejší pomocné kritérium, protože pro účely této práce nejsou žádné speciální periferie (kromě výše zmíněných komunikačních modulů) potřeba. Vybavenost dalšími periferiemi je spíše výhodou pro další potenciální rozvoj tohoto tématu.

Existuje několik rodin mikrokontrolérů, které připadají v úvahu při zohledňování výše zmíněných kritérií. Patří mezi ně 8–bitové PIC čipy od firmy Microchip [20], 8–bitové AVR čipy od firmy ATMEL [6], nebo 8–bitová rodina 78K od firmy RENESAS [23].

### 3.1.2 Požadavky na vývojovou platformu

Při výběru vhodné platformy byla zohledněna především následující prioritní kritéria:

- vybavenost platformy vhodným mikrokontrolérem,
- dostatek vstup–výstupních zařízení pro komunikaci mezi uzly clusteru,
- nízká cena.

Dostatek vstup–výstupních zařízení v tomto případě prakticky znamená dostatek pinů na vývojové desce, které jsou vhodně propojeny s komunikačními periferiemi mikrokontroléru. Vzhledem k tomu že většina levných a dostupných vývojových platforem je vybavena podobným komunikačním rozhraním a pohybují se ve stejné cenové relaci, největší důležitost mělo kritérium ohledně vhodného mikrokontroléru. Kritéria, která při výběru vhodné platformy hrála méně důležitou roli, ale též bylo potřeba je důkladně zvážit, potom byla následující:

- **Kvalita softwarové podpory ze strany výrobce**
- **Nízký příkon**
- **Malé rozměry**

Kvalita softwarové podpory ze strany výrobce se ukázala jako faktor, ve kterém se dostupné vývojové platformy mohou značně lišit.

## 3.2 Zvolený mikrokontrolér

Na základě provedených rešerší a průzkumu dostupných zařízení byl vybrán mikrokontrolér ATmega32U4 od firmy Atmel [4]. Jedná se o mikrokontrolér, který patří do rodiny

8–bitových *AVR* čipů [6]. Navzdory nízké pořizovací ceně, která se v současné době pohybuje okolo 3,5 $, a relativně jednoduché
8–bitové architektuře, se jedná o dostatečně výkonný výpočetní stroj, který splňuje všechna požadovaná kritéria, zejména co se týče dostatku vhodných komunikačních periferií. Nejdůležitější vlastnosti tohoto mikrokontroléru jsou shrnuty v tabulce 3.1.

| **ATmega32u4** | |
|---:|---|
| architektura | 8-bit AVR |
| maximální taktovací frekvence | 16 MHz |
| kapacita flash paměti | 32 KB |
| kapacita EEPROM | 1024 B |
| počet pinů | 44 |
| maximální počet I/O pinů | 26 |
| USB | 1 modul |
| SPI | 2 moduly |
| TWI (I2C) | 1 modul |
| počet ADC kanálů | 12 |
| Analogový komparátor | 1 modul |
| Časovač | 4 moduly |
| Watchdog | ano |

Tabulka 3.1: Nejdůležitější vlastnosti mikrokontroléru Atmel ATmega32u4

### 3.2.1 Výpočetní možnosti zvoleného mikrokontroléru

Jak již bylo zmíněno výše, zvolený mikrokontrolér je založen na 8–bitové *AVR* architektuře. Tato architektura je vybavena instrukční sadou typu *RISC*, tedy pouze základními instrukcemi.

### 3.2.2 Komunikační rozhraní zvoleného mikrokontroléru

V této podkapitole budou podrobně popsána jednotlivá komunikační rozhraní, která zvolený mikrokontrolér nabízí a která mohou být použita pro komunikaci mezi jednotlivými uzly clusteru.

#### SPI

SPI–*Serial Peripheral Interface* (sériové periferní rozhraní), též známé jako *four–wire–interface*, je díky své jednoduchosti a efektivitě jedno z nejrozšířenějších rozhraní v oblasti mikrokontrolérů. Je velice často používáno také na straně různých elektronických zařízení, jako například LCD displeje, digitální potenciometry, nebo například různé druhy senzorů [8].

V případě *SPI* se jedná o rozhraní typu *master–slave*, tedy že existuje jedno centrální řídící zařízení *master* a minimálně jedno zařízení typu *slave*. Rozhraní *SPI* obvykle využívá zapojení do topologie typu hvězda, kde zařízení typu *master* je středem této topologie. Komunikace mezi zařízeními je typu *plný duplex*. To znamená že v jeden okamžik mohou být data přenášena oběma směry, přičemž vždy komunikuje zařízení typu *master* s jedním z připojených zařízení typu *slave*.

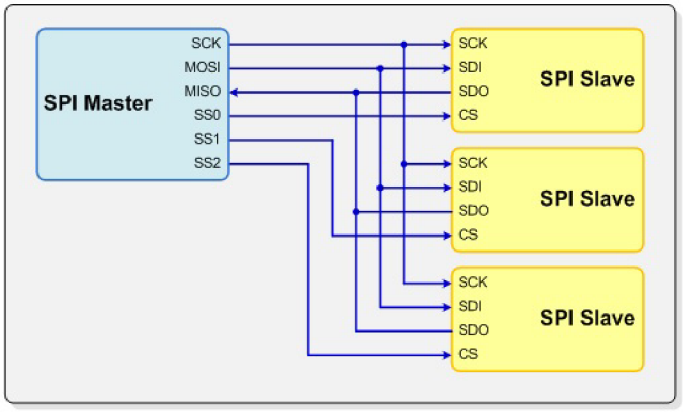
Pro samotné propojení potřebuje každé zařízení, bez ohledu na jeho typ, tři vodiče připojené do společné sběrnice. Konkrétně se jedná o vodiče:

- **SCLK**
  *Serial Clock*, vodič pro hodinový signál generovaný zařízením typu *master*, který slouží pro synchronizaci v průběhu vzorkování přenášených dat.
- **MOSI**
  *Master Out Slave In*, vodič pro datový signál, kterým jsou sériově přenášena data ve směru od zařízená typu *master* do zařízení typu *slave*.
- **MISO**
  *Master In Slave Out*, vodič pro datový signál, kterým jsou sériově přenášena data ve směru od zařízená typu *slave* do zařízení typu *master*.

Kromě těchto tří vodičů je dále zapotřebí jeden další vodič pro každé připojené zařízení typu *slave*. Tento vodič je označován jako:

- **SS**
  *Slave Select*, vodič pro signál, který je řízen zařízením typu *master* a slouží pro výběr právě jednoho zařízení typu *slave* se kterým chce *master* komunikovat.

Počet zařízení typu *slave*, která mohou být připojena k zařízení typu *master* je omezen počtem programovatelných digitálních pinů, kterými je *master* vybaven, neboť pro každé připojené zařízení typu *slave* potřebuje *master* jeden takový pin pro připojení příslušného vodiče *SS*. Samotné schéma možného zapojení *SPI* rozhraní je vidět na obrázku 3.1.




Obrázek 3.1: Schéma SPI komunikačního rozhraní
(obrázek převzat z [10])

Standardní průběh komunikace přes rozhraní *SPI* by pak vypadal tak, že na začátku má zařízení typu *master* všechny signály SS nastaveny na klidovou úroveň. Pokud chce *master* začít komunikovat s jedním ze zařízení typu *slave*, aktivuje příslušný signál SS, odpovídající *slave* toto zaznamená a je připraven komunikovat. Zařízení *master* a vybraný *slave* si prostřednictvím vodičů *MOSI* a *MISO* vymění potřebná data. Ostatní zařízení typu *slave*, která nebyla vybrána, tuto komunikaci ignorují. Poté co jsou všechna potřebná data přenesena, uvede *master* všechny *SS* signály do klidové úrovně a komunikace je skončena.

### TWI

TWI-*Two Wire Interface* (dvoudrátové rozhraní), je mimořádně jednoduché rozhraní pro sériový přenos dat. Ve své podstatě je *TWI* to samé jako jako sériová sběrnice *I2C* (I–squared–C), která byla vyvinuta společností Philips [15] a která má na tuto sběrnici chráněnu značku. Společnost Atmel používá u svých zařízení ten samý princip, pouze s tím rozdílem, že toto rozhraní přejmenovala na *TWI*.

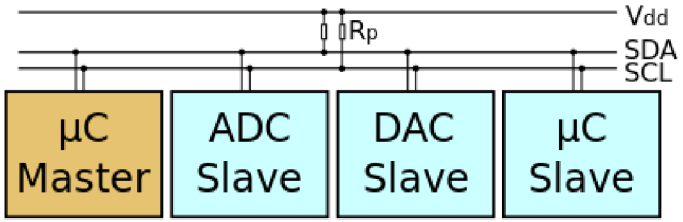
Podobně jako u *SPI* je možné rozhraní *TWI* použít pro komunikaci mikrokontroléru s různými druhy periferních zařízení, ale také jinými mikrokontroléry. A stejně jako u *SPI* je *TWI* rozhraní určeno pro komunikaci typu *master–slave*, kde zařízení typu *master* tvoří centrální řídicí prvek hvězdicové topologie. Důležitý rozdíl oproti *SPI* spočívá v tom, že rozhraní *TWI* umožňuje pouze *poloviční duplex*, to znamená, že v jednom okamžiku může vysílat pouze jedno zařízení a tedy v jednom okamžiku mohou data proudit pouze jedním směrem.

Jak už název sběrnice napovídá, k její realizaci jsou potřeba pouhé dva vodiče, což z ní činí jednu z nejjednodušších sběrnic, které lze ještě použít pro plnohodnotný přenos dat. Označení potřebných vodičů je pak následující:

- **SCL**
  *Serial Clock Line*, vodič pro hodinový signál generovaný zařízením typu *master*, který slouží pro synchronizaci v průběhu vzorkování přenášených dat.
- **SDA**
  *Serial Data Line*, vodič, který slouží pro přenos veškerých dat.

Při propojení zařízení prostřednictvím sběrnice *TWI* je každé zařízení jednoznačně identifikováno svojí číselnou adresou. Toto komunikační rozhraní používá 7–bitovou adresaci, takže prostřednictvím jedné sběrnice *TWI* je možno propojit jedno zařízení typu *master* a až 127 zařízení typu *slave*.

Pro fyzickou realizaci je potřeba, aby každý ze dvou vodičů byly přes pull–up rezistor připojen ke kladnému napětí, aby mohla být zajištěna hodnota napětí definovaná pro klidový stav. Samotné schéma zapojení sběrnice *TWI* je vidět na obrázku 3.2




Obrázek 3.2: Schéma TWI komunikačního rozhraní
(obrázek převzat z [27])

Samotná komunikace prostřednictvím sběrnice *TWI* probíhá (stručně řečeno a zobecněno) tím způsobem, že uzel *master* prostřednictvím *SDA* přenese adresu zařízení *slave* se kterým chce komunikovat a příznak požadované operace (čtení/zápis), příslušné zařízení potvrdí svoji připravenost pro komunikaci, která poté proběhne. Zařízení, která nebyla adresována, tuto komunikaci ignorují. V rámci sběrnice *TWI* je implementován mechanismus pro detekci kolize tak, aby bylo zabráněno tomu, že v jeden okamžik bude vysílat více zařízení.

#### USB

USB–*Universal Serial Bus* je jedním z nerozšířenějších komunikačních rozhraní a to nejen v oblasti mikrokontrolérů, ale v celé oblasti výpočetní techniky [8]. Velká výhoda *USB* rozhraní spočívá, jak už vyplývá z názvu, v jeho univerzálnosti. Prostřednictvím *USB* může komunikovat obrovské množství různých druhů zařízení. Kromě mikrokontrolérů se jedná hlavně o nejrůznější počítačové periferie jako jsou externí paměťová zařízení , ovládací periferie, nebo třeba skenery. Hlavní výhoda tohoto komunikačního rozhraní je však to, že kromě přenosu dat, umožňuje také napájení připojené periferie. I když jsou možnosti tohoto druhu napájení omezené a nejsou schopné pokrýt energetické požadavky zařízení, která mají vyšší příkon (například tiskárny), pro velké množství energeticky nenáročných periferií (například flash disky, nebo externí pevné disky) je tento druh napájení naprosto postačující.

Sériové rozhraní *USB* pracuje s architekturou *master–slave*, kdy je možné k zařízení typu *master* připojit až 127 zařízení typu *slave*. V případě tohoto rozhraní platí, že *slave* nemůže začít přenášet data z vlastní iniciativy, vždy se řídí pokyny centrálního zařízení *master*. Nejobvyklejší použití tohoto rozhraní je takové, že v roli zařízení typu *master* vystupuje osobní počítač, které ovládá několik připojených periferií, zpravidla osazených mikrokontroléry. Použití mikrokontroléru, jakožto zařízení typu *master* není u rozhraní *USB* obvyklé. Stejně tak není obvyklé použití tohoto rozhraní pro komunikaci mezi dvěma a více mikrokontroléry, protože samotný komunikační standard rozhraní *USB* je velice komplikovaný a obsluha tohoto rozhraní a představuje nezanedbatelnou spotřebu výpočetních zdrojů.

#### Digitální programovatelné vstup–výstupní piny

Každý mikrokontrolér je vybaven určitým množstvím programovatelných vstup–výstupních pinů.

Tyto programovatelné vstup–výstupní piny jsou obvykle používány po kontrolu periferií, například pro signál SS (*Slave Select*) u *SPI* komunikačního rozhraní, nebo pokud to dovoluje technická realizace mikrokontroléru tak také pro napájení zařízení s velice nízkým příkonem (například LED dioda).

Také je možné využít tyto piny pro přenos regulérních aplikačních dat mezi dvěma a více mikrokontroléry. V takovém případě je ovšem potřeba si naimplementovat vlastní komunikační protokol. Tato možnost není obvykle používaná, protože takovýto způsob komunikace nemá žádnou hardwarovou podporu na straně mikrokontroléru, všechno je řešeno softwarově, takže z hlediska rychlosti není příliš efektivní.

## 3.3 Zvolená platforma

Jako vhodná vývojová platforma, která je osazena zvoleným mikrokontrolérem *ATmega32u4* a která poslouží jakožto jeden výpočetní uzel výsledného výpočetního clusteru, byla zvolena vývojová deska **Arduino Leonardo** od od firmy Atmel [2].

Arduino je open–source projekt, jehož cílem je vyvíjet jednoduché, levné víceúčelové platformy založené na mikrokontrolérech společnosti Atmel a to včetně plnohodnotné softwarové podpory, tedy vývojového prostředí, standardních knihoven a rozsáhlé dokumentace.

Vybraná deska Arduino Leonardo splňuje veškeré primární požadavky stanovené v podkapitole 3.1.2. Je vybavená vhodným mikrokontrolérem, má dostatek komunikačních rozhraní a je jednoduchá a levná. V současné době se její pořizovací cena pohybuje okolo

300 Kč. Stejně tak vyhovuje požadavkům na kvalitní podporu ze strany výrobce. Nejdůležitější parametry desky Arduino Leonardo jsou shrnuty v tabulce 3.2.

| **Arduino Leonardo** | |
|---:|---|
| mikrokontrolér | ATmega32u4 |
| operační napětí | 5 V |
| taktovací frekvence | 16 MHz |
| kapacita flash paměti | 32 KB |
| kapacita SRAM paměti | 2,5 KB |
| kapacita EEPROM paměti | 1 KB |
| počet I/O pinů | 20 |
| počet PWM kanálů | 7 |
| počet vstupních analogových pinů | 12 |

Tabulka 3.2: Nejdůležitější vlastnosti vývojové desky Arduino Leonardo [2]

### 3.3.1 Výpočetní možnosti zvolené platformy

Použitá taktovací frekvence 16 MHz představuje ve světě mikrokontrolérů standardní průměr a z hlediska rychlosti výpočtu by měla být naprosto postačující. Stejně tak paměť pro uložení programu (32 KB) poskytuje dostatek prostoru pro implementaci komplexních aplikací. Určitý limit pak představuje dostupná kapacita operační paměti, neboť 2,5 KB není prostor se kterým by mohla být testována aplikace pracující se skutečně velkým množstvím dat. Ovšem relativně omezená operační paměť je vlastnost prakticky všech velice levných mikrokontrolérů.

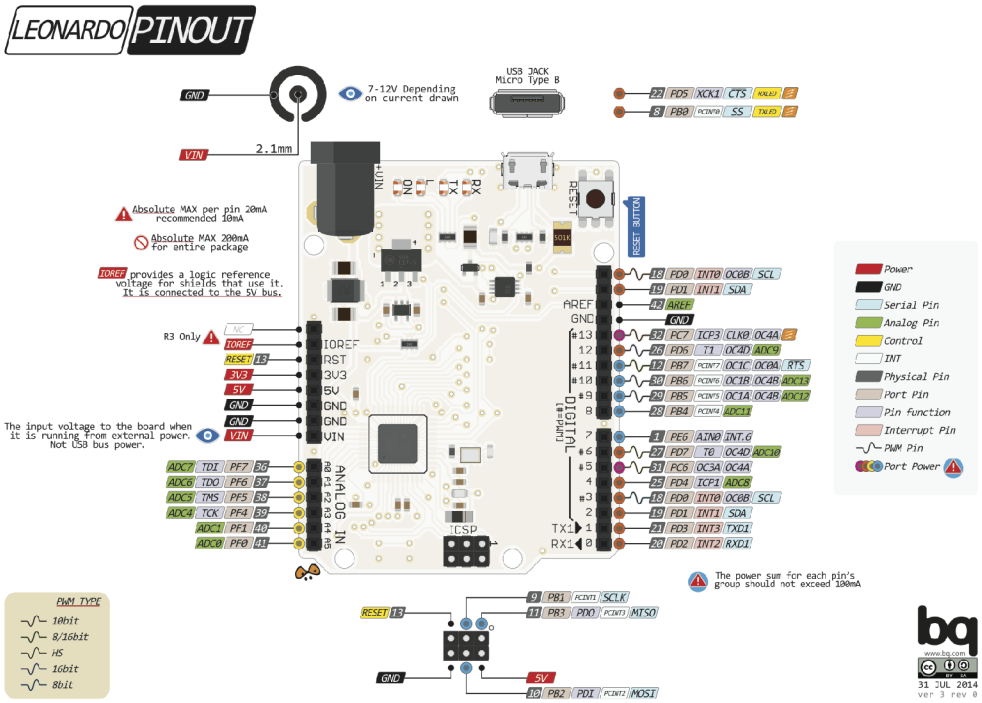
### 3.3.2 Komunikační rozhraní zvoleného platformy

Komunikační rozhraní platformy *Arduino Leonardo* využívá všech možností mikrokontroléru ATmega32u4. Na vývojové desce je dostupných celkem 32 pinů. Některé z nich jsou použity pouze pro rozvod napájení, ale většina z nich může zastávat několik rozdílných funkcí, podle vykonávaného programu, neboť jsou připojeny k různým komunikačním modulům zabudovaným do mikrokontroléru.

Kromě těchto 32 pinů je na vývojové desce zabudovaný *ICSP (In–Circuit Serial Programming)* konektor, který je primárně určen pro komunikaci prostřednictvím rozhraní *SPI*. Konektor *ISCP* je určen pro šest vodičů. Kromě standardních vodičů pro *SPI*, tedy *MOSI, MISO* a *SCLK*, jsou to 2 vodiče pro případné napájení připojené periferie a jeden vodič pro signál reset.

Vývojová platforma *Arduino Leonardo* je také vybavena *micro-USB* konektorem, který slouží k několika různým účelům. Může poskytovat napájení pro tuto platformu, dále poskytuje rozhraní pro naprogramování mikrokontroléru. Tento *micro-USB* konektor je také součástí uživatelského rozhraní, protože právě prostřednictvím tohoto konektoru je vývojová platforma připojena k PC, kde je zobrazován průběh a výsledek prováděného výpočtu. Zobrazování je zrealizováno prostřednictvím jednoduché konzole, která je součástí standardního vývojového prostředí dodávaného výrobcem.

Samotné schéma komunikačního rozhraní, které nabízí vývojová platforma *Arduino Leonardo*, je znázorněno na obrázku 3.3.




Obrázek 3.3: Schéma komunikačního rozhraní vývojové desky Arduino Leonardo (obrázek převzat z [2])

## 3.4 Výsledný výpočetní cluster

Ve chvíli, kdy byly k dispozici požadované čtyři vhodné vývojové platformy, bylo potřeba vytvořit samotnou architekturu výsledného výpočetního clusteru a stanovit jeho základní vlastnosti, kterými jsou topologie propojení, komunikační rozhraní a dále způsob napájení.

### 3.4.1 Komunikační rozhraní výsledného clusteru

Rozvržení komunikačního rozhraní clusteru představuje nejdůležitější krok v jeho konstrukci, neboť se od něj odvíjejí další vlastnosti clusteru. Samotné komunikační rozhraní výpočetního clusteru lze rozdělit do dvou kategorií:

#### Komunikační rozhraní v rámci jednotlivých uzlů

Toto komunikační rozhraní určuje, jakým způsobem budou mezi sebou komunikovat jednotlivé uzly clusteru. V tomto případě je možné reálně použít tři možnosti: rozhraní *SPI*, rozhraní *TWI*, nebo implementovat vlastní komunikační rozhraní prostřednictvím programovatelných vstup–výstupních digitálních pinů.

Použití komunikačního rozhraní *SPI* přináší výhody rychlého komunikačního protokolu s hardwarovou podporou a *plně duplexním* přenosem. Jeho nevýhodou je v případě použité platformy relativně velký počet potřebných vodičů. Z výpočetního uzlu typu *master* by muselo vést celkem 9 vodičů, tedy 6 vodičů pro pokrytí konektoru *ISCP* plus 3 další vodiče pro signály *SS–Slave Select*. Další podstatnou nevýhodou použití tohoto rozhraní je to, že výrobce vývojové platformy *Arduino Leonardo* nepředpokládá použití tohoto zařízení

v režimu *slave* a tedy k tomuto režimu neexistuje softwarová podpora ve formě standardní knihovny. Bylo by tedy potřeba naimplementovat vlastní *SPI* komunikační knihovnu, která by (mimo jiné) manipulovala s řídicími registry mikrokontroléru, tedy na té nejnižší možné úrovni. Jistou nevýhodu by použití komunikačního rozhraní *SPI* také znamenalo z hlediska dalšího rozvoje celého clusteru, protože by to prakticky znemožnilo připojit ke každému uzlu vlastní *SPI* periferii.

Implementace vlastního komunikačního rozhraní za použití programovatelných vstup–výstupních digitálních pinů, by sice byla možná, ale nepřinášela by s sebou žádné praktické výhody. Zvolený mikrokontrolér nemá pro toto rozhraní žádnou hardwarovou podporu v podobě periferního modulu řídícího proces komunikace, vše by muselo být řízeno softwarově. Proto by toto rozhraní nebylo z hlediska rychlosti přenosu příliš efektivní. Rychlost by se dala zvýšit použitím paralelní komunikace za použití většího množství vodičů. Tím by ovšem počet potřebných vodičů značně narostl a to až na samotnou kapacitu vývojové platformy, protože kromě těch datových by bylo potřeba i několik řídicích vodičů. Takové komunikační rozhraní by pak stěží mohlo být pokládáno za jednoduché a efektivní.

Použití komunikačního rozhraní TWI s sebou přináší prakticky pouze samé výhody. Na jeho implementaci postačují pouhé dva vodiče, mikrokontrolér poskytuje hardwarovou podporu tohoto rozhraní prostřednictvím příslušného periferního modulu a výrobce vývojové platformy poskytuje softwarovou podporu tohoto rozhraní v podobě standardní knihovny, která umožňuje výpočetnímu uzlu pracovat jak v režimu *master*, tak v režimu *slave*. Vzhledem k těmto výhodám bylo *TWI* zvoleno jako hlavní komunikační rozhraní mezi jednotlivými uzly clusteru.

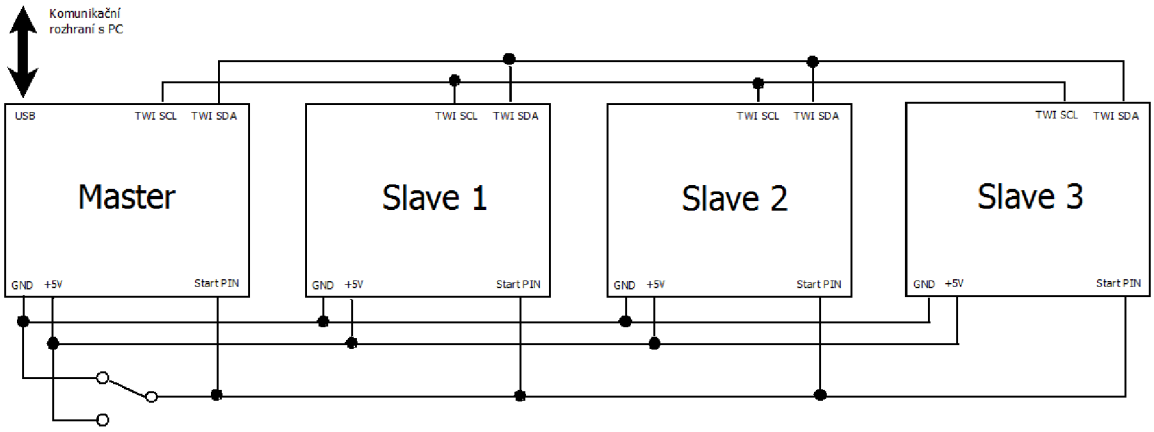
#### Komunikační rozhraní clusteru jako celku

Toto komunikační rozhraní určuje, jakým způsobem bude cluster komunikovat s okolím, tedy jakým způsobem bude předávat výsledky svých výpočtů, aby s nimi mohl uživatel dále pracovat. V tomto případě se jako jediné vhodné řešení ukazuje být *micro–USB* rozhraní, prostřednictvím kterého je cluster spojen s PC, kterému zasílá výsledná data. Toto rozhraní je také použito pro napájení celého clusteru, kdy jsou napájecí piny zařízení typu *master* propojeny s napájecími piny zařízení *slave*. Tímto také odpadá nutnost použití externího napájení pro každý výpočetní uzel zvlášť.

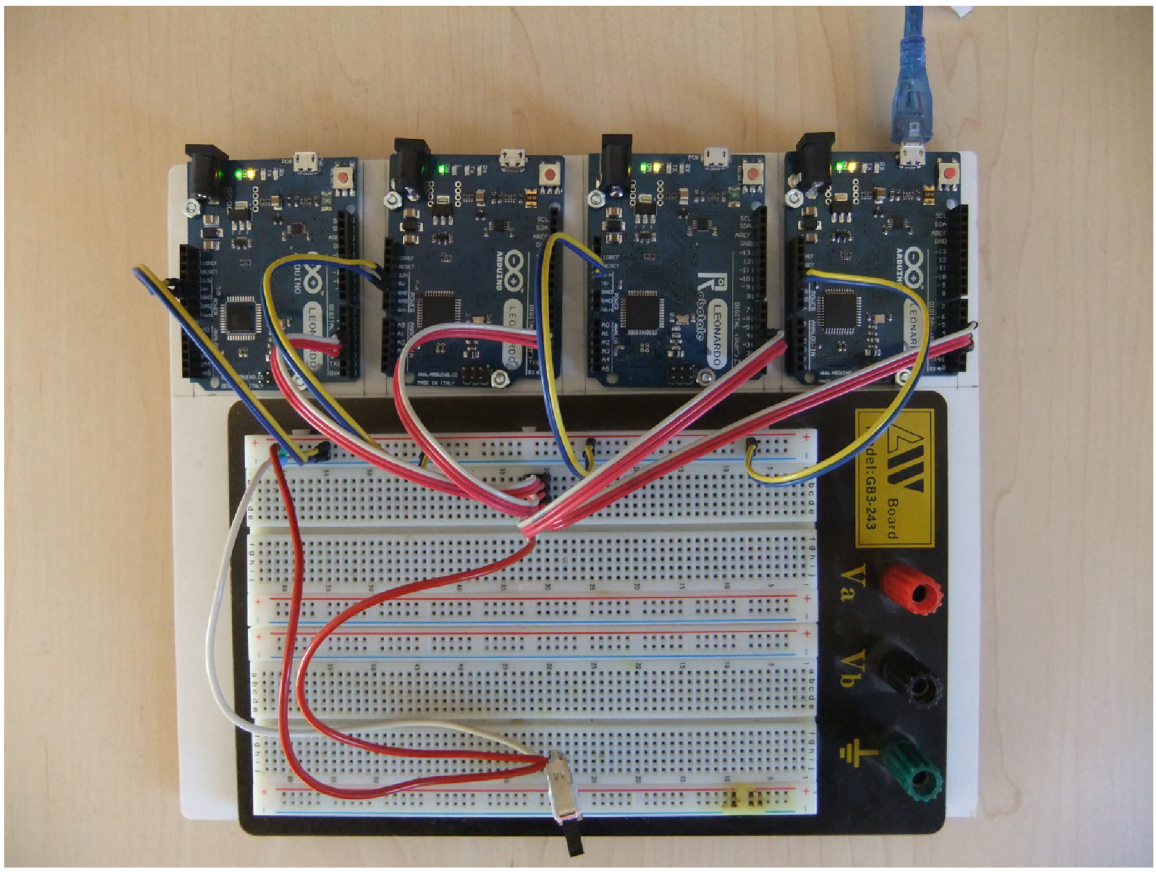
### 3.4.2 Schéma zapojení výsledného clusteru

Schéma navrženého clusteru je znázorněno na obrázku 3.4. Je zde vidět, že kromě napájení a komunikačního rozhraní *TWI* a komunikačního rozhraní *micro-USB* je na každém uzlu zapojen také takzvaný *Start pin*. Jedná se programovatelný digitální pin, který je u všech uzlů nastaven v režimu pro vstup. Funkce tohoto pinu je podrobně popsána v kapitole 4.

Pro samotné fyzické propojení všech potřebných vodičů bylo použito nepájivé pole, jak je vidět na obrázku 3.5.




Obrázek 3.4: Schéma zapojení výsledného výpočetního clusteru



Obrázek 3.5: Fotografie výsledného výpočetního clusteru

# Kapitola 4

# Aplikace na výpočetním clusteru

## 4.1 Stanovení kategorií testovacích aplikací

Při volbě testovacích aplikací bylo potřeba vzít v úvahu několik hledisek. Vzhledem k zaměření této práce na spolupráci více výpočetních uzlů, bylo tím nejdůležitějším aspektem možnosti paralelizace vybraných úloh.

Dále bylo potřeba zvolit testovací aplikace takovým způsobem, aby bylo možné posoudit chování clusteru při různých typech výpočtu. Pro zjednodušení byly uvažovány následující parametry aplikací:

- hlavní typ zpracovávaných dat,
- intenzita komunikace.

Hlavní typem zpracovávaných dat je určeno, jestli aplikace primárně pracuje s celočíselnou, nebo s floating–point aritmetikou.

Intenzitou komunikace je určen objem přenášených dat v průběhu paralelního výpočtu. Tento objem přenášených dat je relativní a to vzhledem k celkovému objemu dat určených ke zpracování. Přičemž za *nízkou intenzitu komunikace* je považována komunikace v aplikaci, při jejímž běhu představuje objem dat, přenášených v průběhu výpočtu, řádově jednotky procent z celkového objemu dat zpracovávaných. Za *vysokou intenzitu komunikace* je považována komunikace v aplikaci, při jejímž běhu je objem dat, přenášených v průběhu výpočtu, srovnatelně velký s celkovým objemem dat zpracovávaných.

Na základě těchto parametrů byly stanoveny tyto kategorie testovacích aplikací:

- aplikace pracující s celočíselnou aritmetikou a s nízkou intenzitou komunikace,
- aplikace pracující s celočíselnou aritmetikou a s vysokou intenzitou komunikace,
- aplikace pracující s floating–point aritmetikou a s nízkou intenzitou komunikace.

Přestože při volbě kategorií testovacích aplikací byly uvažovány dva parametry, z nichž každý může nabývat dvou hodnot (potenciálně tedy čtyři kombinace), jsou ve výsledku kategorie pouze tři. Je to z toho důvodu, že nemá smysl zařazovat kategorii *Aplikace pracující s floating–point aritmetikou a s vysokou intenzitou komunikace*, protože případ vysoké intenzity komunikace je už testován u celočíselné aritmetiky a tak by tato nová kategorie nemohla přinést žádné zásadní nové poznatky. Z hlediska rychlosti komunikace totiž nezáleží na tom jaký datový typ přenášená data reprezentují.

Ještě před implementací a testováním konkrétních paralelních aplikací, běžících na výpočetním clusteru, bylo potřeba implementovat také jejich sekvenční varianty (běžící pouze na jednom uzlu), aby bylo možné následně porovnat rychlosti výpočtů.

## 4.2 Zvolené testovací aplikace a jejich implementace

Všechny testovací aplikace byly naprogramovány v jazyce *C++*. Nejedná se ovšem o klasické *C++*, tak je definováno v nejnovějším standardu [14]. Jedná se o jeho podmnožinu, která neobsahuje všechny standardní knihovny tohoto jazyka. Ovšem na druhou stranu poskytuje vývojové prostředí *Arduino IDE* některé vlastní knihovny, které jsou navrženy s ohledem na vlastnosti vývojových platforem *Arduino* a možnosti jejich mikrokontrolérů. Detailní dokumentaci prostředí *Arduino IDE* je možné nalézt na oficiálních webových stránkách výrobce [1].

V případě implementace testovacích aplikací nebyly využity objektové vlastnosti programovacího jazyka *C++* a to z toho důvodu, že správa objektů při běhu programu znamená nezanedbatelný nárůst spotřeby výpočetních zdrojů, obzvláště pak operační paměti. Kapacita operační paměti je v případě použité vývojové platformy (ostatně jako u většiny mikrokontrolérů) značně omezená a proto je nezbytné ji využívat efektivně. Navíc u všech testovacích aplikací běží na jednom uzlu vždy jen jeden proces. Za takovýchto okolností by použití objektového paradigmatu bylo nejen neefektivní, ale také vyloženě nepotřebné. Jedinou výjimku představuje využití standardní *C++* třídy *String*, která je použita pro zjednodušení práce s textovým výstupem po skončení hlavního výpočtu, tedy ve chvíli, kdy už omezená kapacita operační paměti nepředstavuje problém.

Kompilátor vývojového prostředí *Arduino IDE* vždy zakomponuje hlavní tělo výpočtu do nekonečné smyčky, ze které se případně mohou volat další podprogramy. Tato smyčka je přestavována funkcí „*void loop()*“, která je v podstatě obdobou funkce „*int main(int argc, char* argv[])*“, s tím rozdílem, že nemá žádné parametry ani návratovou hodnotu a je vykonávána stále dokola. To představuje určitou komplikaci, pokud je potřeba, aby program proběhl pouze jednou a poté zobrazil výsledek. Za tímto účelem je běh programu podmíněn aktivací externího přepínače, který je propojen s digitálním vstup–výstupním pinem, takzvaným *Start pinem*, jehož fyzické zapojení je popsáno v podkapitole 3.4.2.

V dalším textu této práce se bude pracovat s pojmem *byte*, přičemž tento pojem může nabývat dvou významů. Prvním může být klasická informatická interpretace, tedy že *byte* je jednotka množství dat představující osmici bitů. V tomto případě bude slovo „byte“ v textu uvedeno bez jakéhokoliv zvýraznění. Druhý možný význam představuje datový typ, který je implementován v rámci standardních knihoven vývojového prostředí *Arduino IDE*. Tento datový typ představuje osmibitové celé číslo bez znaménka (ekvivalent datového typu *unsigned char* v jazyce *C++*). V tomto druhém případě bude v následujícím textu a ukázkách kódu termín **byte** vyznačen tučně.

### 4.2.1 Sekvenční součet prvků v poli

Tato testovací aplikace představuje sekvenční variantu výpočtu pracujícího s celočíselnou aritmetikou, přenos informací mezi jednotlivými výpočetními uzly je nulový. Samotný výpočet je z hlediska algoritmizace velice jednoduchý. Na začátku běhu aplikace je vygenerováno pole **bytů**, poté je pole zpracováno for cyklem, který sečte všechny jeho prvky, výsledná suma je poté zobrazena prostřednictvím výstupní konzole. Princip výpočtu je prostřednictvím pseudokódu naznačen v algoritmu 1.

**Algoritmus 1:** Algoritmus pro sečtení prvků pole datového typu **byte**, sekvenční výpočet na jednom uzlu

```
define ARRAYLENGHT 1000

byte array[ARRAYLENGHT];
long sum = 0;
float time1, time2;

while StartPin == LOW do
    ;
end

fillArray(array);
time1 = gettime();

for int i=0; i<ARRAYLENGHT; i++ do
    sum += sumArray(array);
end

time2 = gettime();
print("sum = ",sum);
print("time = ",time2 - time1);

while StartPin == HIGH do
    ;
end
```

Před samotným během cyklu program zaznamená aktuální čas mikrokontroléru v mikrosekundách *time1* a po skončení cyklu zaznamená čas *time2*. Rozdíl těchto dvou hodnot pak představuje dobu trvání hlavního výpočtu. Základní verze programu pracuje s polem o velikosti 1000 bytů, které se do operační paměti platformy, která má kapacitu 2,5 KB bez problémů vejde. Aby bylo možné otestovat rychlost zpracování i několikanásobně větších polí, je výše zmíněný výpočet jednoduše vykonán několikrát po sobě s tím, že opakovaně probíhá pouze sčítání prvků pole, nikoliv už jeho generování.

### 4.2.2 Paralelní součet prvků v poli: data uložena v každém uzlu

Tato testovací aplikace představuje paralelní variantu výpočtu pracujícího s celočíselnou aritmetikou a s nízkou intenzitou komunikace. Princip algoritmu je takový, že na začátku výpočtu si každý ze čtyř uzlů vygeneruje vlastní pole **bytů**. Toto pole má čtvrtinovou velikost vzhledem k celkové velikosti zpracovávaných dat. Například v případě testování délky výpočtu sčítání prvků v 1000–prvkovém poli si každý uzel vygeneruje pole o délce 250 prvků. Po aktivaci *Start pinu* je spuštěn paralelní výpočet, ve kterém každý ze čtyř uzlů provede součet své části pole. Poté uzel typu *master* postupně vybízí všechna tři zařízení typu *slave*, aby mu zaslala svůj dílčí součet. Uzel *master* poté spojí získaná data ve výsledný součet, který zobrazí na výstup. Doba pro vygenerování pole se do času výpočtu nezapočítává. Princip výpočtu uzlu typu *master* je naznačen prostřednictvím pseudokódu v algoritmu 2, princip výpočtu uzlu typu *slave* pak v algoritmu 3.

**Algoritmus 2:** Algoritmus pro sečtení prvků pole datového typu **byte**, paralelní výpočet, data uložena v každém uzlu před začátkem výpočtu, verze pro uzel typu *master*

```
define NUMBEROFSLAVES 3
define ARRAYLENGHT 250

byte array[ARRAYLENGHT];
long sum = 0;
float time1, time2;

while StartPin == LOW do
    ;
end

fillArray(array);
time1 = gettime();

for int i=0; i<ARRAYLENGHT; i++ do
    sum += sumArray(array);
end

for int j=1; j<=NUMBEROFSLAVES; j++ do
    sum += requestSumFromSlave(j);
end

time2 = gettime();
print("sum = ",sum);
print("time = ",time2 - time1);

while StartPin == HIGH do
    ;
end
```

**Algoritmus 3:** Algoritmus pro sečtení prvků pole datového typu **byte**, paralelní výpočet, data uložena v každém uzlu před začátkem výpočtu, verze pro uzel typu *slave*

```
define ARRAYLENGHT 250

byte array[ARRAYLENGHT];
long sum = 0;

while StartPin == LOW do
    ;
end

fillArray(array);

for int i=0; i<ARRAYLENGHT; i++ do
    sum += sumArray(array);
end

while StartPin == HIGH do
    ;
end
```

Součástí implementace programu pro uzel typu *slave* je také funkce pro ošetření přerušení, které je vyvoláno požadavkem na zaslání dat do uzlu *master*. Tato funkce pošle uzlu *master* dílčí součet reprezentovaný proměnou *sum*.

### 4.2.3 Paralelní součet prvků v poli: data distribuována z hlavního uzlu

Tato testovací aplikace představuje paralelní variantu výpočtu pracujícího s celočíselnou aritmetikou a s vysokou intenzitou komunikace. Princip algoritmu je takový, že uzel typu *master* na začátku výpočtu vygeneruje pole **bytů**. Toto pole má čtvrtinovou velikost vzhledem k celkové velikosti zpracovávaných dat. Po aktivaci *Start pinu* uzel *master* potupně naváže komunikaci s každým uzlem typu *slave* a pošle mu ke zpracování celé pole, které vygeneroval. Pro zjednodušení výpočtu je každému uzlu *slave* posláno to stejné pole, aby nemuselo být generováno každé zvlášť. Poté co všechny uzly *slave* mají svá data ke zpracování, začnou všechny čtyři uzly provádět sčítání své části pole. Ve chvíli, kdy uzel *master* dokončí výpočet svého dílčího součtu, začne postupně vybízet všechna zařízení *slave* k zaslání jejich dílčích součtů. Poté uzel *master* spojí dílčí součty ve výslednou sumu, kterou zobrazí na výstup. Do délky výpočtu se nezapočítává čas pro vygenerování pole. Měří se čas potřebný pro přenos polí ke všem uzlům *slave*, paralelní výpočet a přenos dílčích součtů do uzlu *master*. Princip výpočtu uzlu typu *master* je naznačen prostřednictvím pseudokódu

v algoritmu 4.

**Algoritmus 4:** Algoritmus pro sečtení prvků pole datového typu **byte**, paralelní výpočet, data pro zpracování distribuována z hlavního uzlu, verze pro uzel typu *master*

```
define NUMBEROFSLAVES 3
define ARRAYLENGHT 250

byte array[ARRAYLENGHT];
long sum = 0;
float time1, time2;

while StartPin == LOW do
    ;
end

fillArray(array);
time1 = gettime();

for int i=1; i<=NUMBEROFSLAVES; i++ do
    sendArrayToSlave(array,i);
end

for int j=0; j<ARRAYLENGHT; j++ do
    sum += sumArray(array);
end

for int k=1; k<=NUMBEROFSLAVES; k++ do
    sum += requestSumFromSlave(k);
end

time2 = gettime();
print("sum = ",sum);
print("time = ",time2 - time1);

while StartPin == HIGH do
    ;
end
```

Implementace programu pro uzel typu *slave* tvoří dvě funkce pro ošetření přerušení. První reaguje na požadavek pro příjem dat uzlem *slave* od uzlu *master* a její součástí je jak příjem dat pole, tak provedení součtu jeho prvků, který uloží do globální proměnné. Druhá funkce reaguje na požadavek pro odeslání dat do uzlu *master*, tato funkce posílá dílčí součet reprezentovaný globální proměnnou *sum*.

### Komunikační protokol založený na TWI rozhraní

Při implementaci testovací aplikace, která využívá přenos velkého množství dat(v řádu tisícovek bytů) se standardní konfigurace komunikačního rozhraní *TWI* ukázala jako limitující, protože kapacita komunikačního bufferu je v případě vývojové platformy *Arduino Leonardo*

omezena na 32 bytů [2]. Větší množství nelze v rámci jedné transakce přenést.

Proto byl pro potřeby této práce implementován jednoduchý komunikační protokol, který umožňuje mezi uzly clusteru přenést data o velikosti v řádu kilobytů. princip komunikačního protokolu spočívá v tom, že data určená k přenosu jsou rozdělena do paketů, přičemž tyto pakety jsou posílány ve sledu jednotlivých navzájem navazujících *TWI* transakcí.

Pakety mohou být dvojího druhu:

- inicializační pakety,
- datové pakety.

*Inicializační paket* je v rámci jedné komunikace vždy jeden a je posílán jako první. Jeho celková délka je vždy 5 bytů. První byte má vždy hodnotu nula, zbývající 4 byty reprezentují 32–bitové celé číslo, které určuje celkovou délku objemu dat (v bytech), který bude v rámci této komunikace přeposlán.

*Datových paketů* může být v rámci jedné komunikace více (konkrétně 1-255) a jsou posílány po inicializačním paketu. Hlavička datového paketu má vždy dva byty, první byte představuje číslo paketu (nabývá hodnot 1-255), druhý pak počet bytů v datové části paketu (nabývá hodnot 1-30). Následující datová část má proměnlivou délku 1-30 bytů a může obsahovat libovolné informace.

Objem dat, která lze prostřednictvím tohoto protokolu v rámci jedné komunikace přenést, je odvozen ze vztahu 4.1, kde $n$ je maximální počet datových paketů a $c$ je maximální množství dat přenášených v jednom datovém paketu.

$$C = n * c = 255 * 30[B] = 7650B \tag{4.1}$$

Efektivitu tohoto protokolu lze popsat vztahem, který určuje poměr mezi objemem aplikačních dat a všech dat přenesených (to znamená včetně režie $R$). Při využití protokolu na maximum je jeho percentuální efektivita vypočítána ve vztahu 4.2.

$$E = \frac{C}{C+R} * 100\% = \frac{n*c}{n*c+5+2*n} * 100\% = \frac{7650}{7650+5+510} * 100\% \doteq 93,7\% \tag{4.2}$$

### 4.2.4 Sekvenční výpočet čísla $\pi$

Tato testovací aplikace představuje sekvenční variantu výpočtu pracujícího s floating–point aritmetikou, přenos informací mezi jednotlivými výpočetními uzly je nulový. Tato aplikace provádí výpočet čísla $\pi$ prostřednictvím *formule BBP* (Bailey–Borwein–Plouffe) [17], která je uvedena ve vztahu 4.3.

$$\pi = \sum_{k=0}^{\infty} \left[ \left(\frac{1}{16}\right)^k \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right] \tag{4.3}$$

Tuto formuli je možné upravit na ekvivalentní vztah 4.4.

$$\pi = \sum_{k=0}^{\infty} \left[ \left(\frac{1}{16}\right)^k \left( \frac{120k^2+151k+47}{512k^4+1024k^3+712k^2+194k+15} \right) \right] \tag{4.4}$$

Výhoda tohoto algoritmu spočívá v tom, že výpočet jednotlivých členů sumy je možné provádět nezávisle na sobě. Proto je tento algoritmus vhodný pro případnou paralelizaci.

# Kapitola 5

# Experimenty a jejich výsledky

## 5.1 Popis postupu při experimentálním měření

Při testovacích experimentech na aplikacích, které sčítají prvky v poli, byla měřena doba výpočtu v závislosti na celkové velikosti zpracovávaného pole, přičemž pro každou možnou variantu bylo provedeno deset měření. Z každých deseti takto získaných hodnot byl vypočítán aritmetický průměr a medián. Právě medián byl zvolen jakožto nejvíce reprezentativní hodnota pro popis rychlosti výpočtu. Veškeré časové úseky byly měřeny v mikrosekundách prostřednictvím funkce *micros()*, která je součástí standardních knihoven vývojového prostředí *Arduino IDE* [1].

V případě testovací aplikace pro sekvenční výpočet čísla $\pi$ prostřednictvím *BBP formule* bylo zkoumáno kolik iteračních kroků tohoto algoritmu je schopná platforma vypočítat. Takto vypočítaná hodnota čísla $\pi$ byla porovnána s oficiální přesnější hodnotou [17].

## 5.2 Naměřené výsledky

V následujících tabulkách a grafech jsou shrnuty provedené experimenty a naměřené výsledky.

| **Doba výpočtu [v mikrosekundách] sekvenčního sčítání prvků pole v závislosti na jeho velikosti** | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | číslo měření | | | | | | | | | | | |
| počet prvků pole | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 8. | 9. | 10. | průměr | medián |
| 1000 | 896 | 884 | 896 | 888 | 896 | 896 | 896 | 896 | 896 | 896 | 894 | 896 |
| 2000 | 1788 | 1788 | 1788 | 1788 | 1784 | 1792 | 1788 | 1788 | 1780 | 1788 | 1787 | 1788 |
| 3000 | 2680 | 2672 | 2680 | 2684 | 2680 | 2680 | 2672 | 2676 | 2676 | 2676 | 2678 | 2678 |
| 4000 | 3560 | 3560 | 3572 | 3556 | 3564 | 3560 | 3568 | 3560 | 3568 | 3556 | 3562 | 3560 |
| 5000 | 4460 | 4460 | 4452 | 4448 | 4452 | 4460 | 4460 | 4452 | 4452 | 4448 | 4454 | 4452 |
| 6000 | 5352 | 5352 | 5348 | 5344 | 5348 | 5356 | 5352 | 5352 | 5344 | 5344 | 5349 | 5350 |
| 7000 | 6248 | 6244 | 6240 | 6236 | 6248 | 6236 | 6248 | 6236 | 6240 | 6236 | 6241 | 6240 |

Tabulka 5.1: Doba výpočtu [v mikrosekundách] sekvenčního sčítání prvků pole v závislosti na jeho velikosti

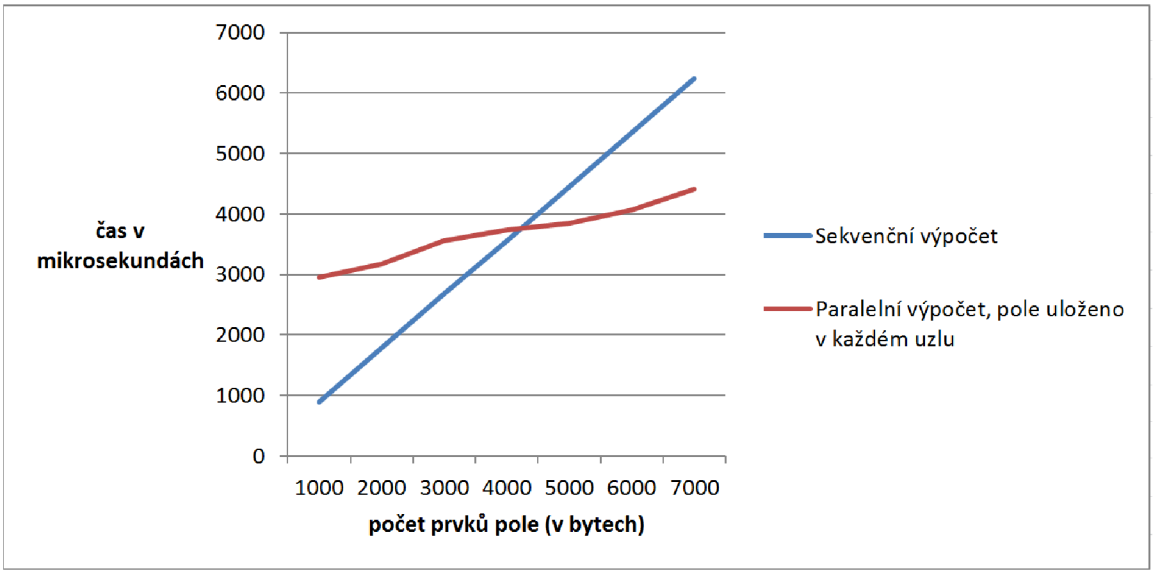
| Doba výpočtu [v mikrosekundách] paralelního sčítání prvků pole (pole uloženo v každém uzlu ) | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | číslo měření | | | | | | | | | | | |
| počet prvků pole | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 8. | 9. | 10. | průměr | medián |
| 1000 | 2960 | 2952 | 2952 | 2968 | 2944 | 2944 | 2972 | 2960 | 2944 | 2948 | 2954 | 2952 |
| 2000 | 3176 | 3176 | 3168 | 3168 | 3164 | 3188 | 3180 | 3180 | 3184 | 3180 | 3176 | 3178 |
| 3000 | 3564 | 3548 | 3552 | 3560 | 3556 | 3540 | 3568 | 3552 | 3552 | 3600 | 3559 | 3554 |
| 4000 | 3736 | 3740 | 3732 | 3728 | 3728 | 3732 | 3728 | 3728 | 3732 | 3728 | 3731 | 3730 |
| 5000 | 3856 | 3840 | 3836 | 3840 | 3840 | 3844 | 3840 | 3856 | 3856 | 3844 | 3845 | 3842 |
| 6000 | 4072 | 4072 | 4068 | 4064 | 4056 | 4088 | 4056 | 4056 | 4056 | 4060 | 4065 | 4062 |
| 7000 | 4408 | 4408 | 4412 | 4408 | 4408 | 4408 | 4396 | 4388 | 4400 | 4396 | 4403 | 4408 |

Tabulka 5.2: Doba výpočtu [v mikrosekundách] paralelního sčítání prvků pole v závislosti na jeho velikosti, pole je uloženo v každém uzlu před začátkem výpočtu

| Doba výpočtu [v mikrosekundách] paralelního sčítání prvků pole (pole distribuováno z hlavního uzlu ) | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | číslo měření | | | | | | | | | | | |
| počet prvků pole | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 8. | 9. | 10. | průměr | medián |
| 1000 | 91400 | 91616 | 91400 | 91400 | 91416 | 91440 | 91428 | 91600 | 91552 | 91576 | 91491 | 91440 |
| 2000 | 188200 | 188420 | 188336 | 188280 | 188432 | 188404 | 188464 | 188320 | 188440 | 188400 | 188372 | 188402 |
| 3000 | 296120 | 296160 | 296604 | 296292 | 296484 | 296256 | 296316 | 296308 | 296288 | 296304 | 296313 | 296298 |
| 4000 | 418448 | 418700 | 418388 | 418456 | 418460 | 418416 | 419040 | 418296 | 419012 | 418464 | 418568 | 418458 |
| 5000 | 548092 | 547992 | 548992 | 548420 | 548048 | 547780 | 547892 | 547872 | 547900 | 548320 | 548031 | 547992 |
| 6000 | 687768 | 688088 | 687908 | 688308 | 688080 | 687872 | 687912 | 688520 | 688120 | 687720 | 688030 | 687996 |
| 7000 | 843912 | 844340 | 844084 | 843948 | 844256 | 844004 | 844144 | 844276 | 844284 | 844320 | 844157 | 844200 |

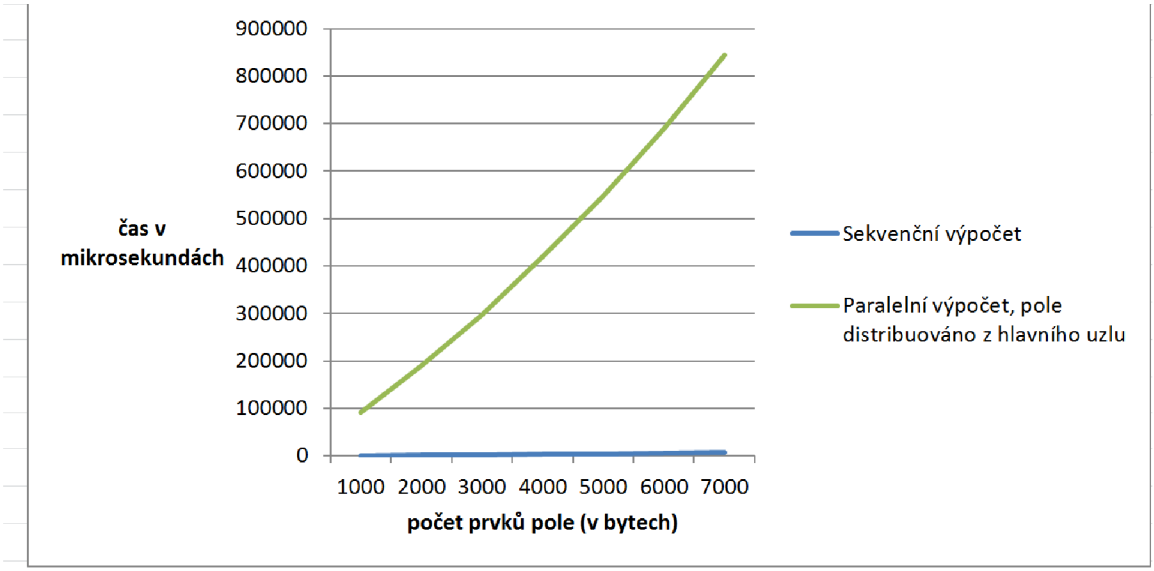
Tabulka 5.3: Doba výpočtu [v mikrosekundách] paralelního sčítání prvků pole v závislosti na jeho velikosti, pole je uloženo v každém uzlu před začátkem výpočtu

Z obrázku 5.1 je vidět, že v případě sekvenčního sčítání za použití jednoho výpočetního uzlu, je nárůst časové složitosti v závislosti na počtu prvků pole (podle očekávání) lineární. Dále je z tohoto grafu patrno, že v případě paralelního výpočtu, za použití clusteru, má časová složitost složitější průběh a pro nízký počet prvků pole je méně efektivní než sekvenční řešení. Ovšem od určitého počtu prvků pole je paralelní výpočet výrazně rychlejší.




Obrázek 5.1: Porovnání délky sekvenčního a paralelního výpočtu sčítání prvků v poli (pole uloženo v každém uzlu) v závislosti na množství zpracovávaných dat

Na obrázku 5.2 je vidět, že paralelní výpočet je pro případ distribuce pole z hlavního uzlu výrazně pomalejší než sekvenční řešení, a to bez ohledu na počet prvků pole.




Obrázek 5.2: Porovnání délky sekvenčního a paralelního výpočtu sčítání prvků v poli(pole distribuováno z hlavního uzlu) v závislosti na množství zpracovávaných dat

Tabulka 5.4 ukazuje přesnost výpočtu čísla $\pi$ v závislosti na počtu provedených kroků aproximačního algoritmu a srovnává takto získané číslo s přesnější oficiální hodnotou [18]. *Inf* je v tomto případě pojem, který je použit u standardní výstupní konzole pro výpis čísla, které nedokáže platforma reprezentovat.

| **Přesnost výpočtu čísla pi v závislosti na počtu vykonaných iterací aproximačního algoritmu** | | | |
|---|---|---|---|
| počet iterací | vypočítaná hodnota | oficiální hodnota | relativní odchylka [%] |
| 1 | 3,13333344459533 | 3,14159265358979 | -0,262898787499481 |
| 2 | 3,14142251014709 | 3,14159265358979 | -0,005415833988081 |
| 3 | 3,14158749580383 | 3,14159265358979 | -0,000164177426186 |
| 4 | 3,14159250259399 | 3,14159265358979 | -0,000004806345586 |
| 5 | 3,14159274101257 | 3,14159265358979 | 0,000002782753512 |
| 6 | 3,14159274101257 | 3,14159265358979 | 0,000002782753512 |
| 7 | 3,14159274101257 | 3,14159265358979 | 0,000002782753512 |
| 8 | 3,14159274101257 | 3,14159265358979 | 0,000002782753512 |
| 9 | inf | 3,14159265358979 | |

Tabulka 5.4: Závislost přesnosti výpočtu čísla $\pi$ na počtu provedených iterací aproximačního algoritmu *BBP formule*

## 5.3 Energetická náročnost výpočetního clusteru

Součástí experimentů s clusterem bylo i testování spotřeby energie. Získání přesných údajů by vyžadovalo pokročilé elektronické zařízení. Pro potřeby této práce byl proveden pouze hrubý odhad. Měření probíhalo tím způsobem, že byl rozpojen napájecí vodič připojeného *USB* kabelu a následně opět spojen se zabudováním ručičkového ampérmetru. Poté proběhlo měření proudu, protékajícího vodičem.

Experimenty ukázaly, že proudový odběr nevykazuje znatelné výkyvy v závislosti na právě prováděném výpočtu. Na základě naměřeného proudu a znalosti napájecího napětí *USB* rozhraní [9] 5 V, byl stanoven přibližný příkon jedné platformy i celého clusteru, jak je shrnuto v tabulce 5.5. Příkon zařízení $P$ napájeného stejnosměrným napětím byl spočítán podle vztahu $P = I * U$ [11], kde $I$ je hodnota naměřeného proudu a $U$ je hodnota stejnosměrného napětí.

| **Srovnání příkonu zařízení** | | | |
|---|---|---|---|
| | U [V] | I [A] | P [W] |
| 1 platforma | 5 | 0,038 | 0,19 |
| výpočetní cluster ze 4 platforem | 5 | 0,149 | 0,745 |

Tabulka 5.5: Odhad příkonu výpočetního clusteru

Je tedy vidět, že při čtyřnásobném počtu napájených platforem se celkový příkon nezvýšil čtyřikrát, ale pouze s koeficientem 3,92. Tuto odchylku lze přičíst chybě měření.

## 5.4 Shrnutí výsledků experimentů

Z naměřených dat, která jsou shrnuta v tabulkách 5.1, 5.2 a 5.3 vyplývá, že rychlost výpočtu je stabilní bez výraznějších výkyvů, aritmetické průměry se od příslušných mediánů liší jen minimálně.

V případě paralelizace, kdy je část pole uložena v každém uzlu již před začátkem výpočtu, je nárůst času výpočtu pomalejší a od určitého objemu dat dochází v porovnání se sekvečním řešením k významnému urychlení.

V případě paralelizace, kdy je pole před začátkem výpočtu distribuváno z hlavního uzlu, nejenže dochází k několikanásobnému zpomalení oproti sekvenčnímu řešení, ale časová složitost zde vykazuje polynomální charakter. Pakliže bychom naměřené hodnoty času výpočtu aproximovali polynomem 2. stupně, dospěli bychom (po zaokrouhlení) ke vztahu 5.1 kde $y$ je doba výpočtu v mikrosekundách a $x$ je počet prvků v poli. Uvedený vztah byl odvozen pomocí nástroje *Regresion Tool* [28].

$$y = 0,0056x^2 + 80x + 5303 \tag{5.1}$$

Z naměřených dat, která jsou shrnuta v tabulce 5.4, vyplývá že maximální počet iterací aproximačního algoritmu, které je platforma schopná vypočítat je 8. Pro vyšší hodnoty parametru $k$ v *BBP formuli* 4.4 jsou zpracovávána příliš vysoká čísla a způsob reprezentace dat na vývojové platformě neumožňuje s těmito čísly pracovat. Proto už je vypočítaný výsledek po deváté iteraci interpretován jako *inf*.

Dále je v tabulce 5.4 vidět, že výpočetní platforma není schopna příliš přesně reprezentovat ani desetinná čísla, protože už od páté iterace se výsledek nadále nezpřesňuje a zůstává stejný. Vzhledem k nízké přesnosti (maximálně 6 desetinných míst) výpočtu čísla $\pi$ na zvolené platformě nebylo přistoupeno k paralelizaci tohoto algoritmu. Problém nedostatečné přesnosti a rozsahu datových typů by bylo možné vyřešit použitím jiné reprezentace dat, která by pracovala s větším množstvím bitů, například s 64 (místo současných 32). Takovýto nárůst velikosti datových typů by pak ovšem znamenal zpomalení výpočtu a zejména zpomalení komunikace mezi uzly.

# Kapitola 6

# Závěr

Na základě provedených experimentů bylo zjištěno, že v případě paralelizace výpočtu prostřednictvím clusteru, složeného z jednoduchých mikrokontrolérů, lze dosáhnou určitého urychlení, ale za cenu značných omezení. Aplikace musí primárně pracovat s celými čísly a objem dat přenášený mezi jednotlivými uzly musí být naprosto minimální. Ukázalo se, že komunikační rozhraní TWI není vhodné pro přenos velkého objemu dat a představuje zásadní omezení celého výpočtu.

Tuto problematiku je možno dále rozvinout dalšími experimenty, například testování clusterů s více výpočetními uzly, nebo použití jiných komunikačních rozhraní. Také by bylo možné různá komunikační rozhraní navzájem kombinovat, případně využít složitějšího mikrokontroléru s více rozhraními jednoho typu a vytvořit tak mnohem komplexnější topologii výpočetního clusteru.

Jako další možnost rozvoje tohoto tématu je využití víceprocesového výpočtu, a to nejenom v rámci celého clusteru, ale také v rámci jednoho výpočetního uzlu. Například při využití jednoduchého operačního systému *Femto OS* [25].

Dále by bylo vhodné prověřit vlastnosti tohoto clusteru v reálné aplikaci. Jako vhodný kandidát se nabízí například sběr a zpracování dat v senzorových sítích.

# Literatura


[1] Arduino Software: *Arduino IDE Software [online]*. [Online; navštíveno 1.4.2016]. URL https://www.arduino.cc/en/Main/Software

[2] Arduino Software: *Arduino Leonardo [online]*. [Online; navštíveno 2.4.2016]. URL https://www.arduino.cc/en/Main/ArduinoBoardLeonardo

[3] Atmel Corporation : *ATmega32U4 [online]*. [Online; navštíveno 2.4.2016]. URL http://www.atmel.com/devices/ATMEGA32U4.aspx

[4] Atmel Corporation : *ATmega32U4 [online]*. [Online; navštíveno 2.4.2016]. URL http://www.atmel.com/devices/atmega32u4.aspx?tab=parameters

[5] Atmel Corporation : *AVR Libc Reference Manual [online]*. [Online; navštíveno 2.3.2016]. URL http://www.atmel.com/webdoc/AVRLibcReferenceManual/group__twi_demo_1twi_demo_intro.html

[6] Atmel Corporation: *Atmel AVR 8-bit and 32-bit Microcontrollers [online]*. [Online; navštíveno 25.4.2016]. URL http://www.atmel.com/products/microcontrollers/avr/

[7] Atmel Corporation: *Atmel-ICE Physical Interfaces [online]*. [Online; navštíveno 23.4.2016]. URL http://www.atmel.com/webdoc/atmelice/atmelice.using_ocd_physical_spi.html

[8] Catsoulis John: *Designing Embedded Hardware*. O'Reilly Media, 2005, ISBN 0596007558.

[9] Compaq Computer Corporation, Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc,Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V.: *Universal Serial Bus Specification*. 2000.

[10] Corelis Inc.: *SPI Interface [online]*. [Online; navštíveno 27.4.2016]. URL http://www.corelis.com/education/SPI_Tutorial.htm

[11] EngenneringToolbox: *Electrical Formulas [online]*. [Online; navštíveno 8.5.2016]. URL http://www.engineeringtoolbox.com/electrical-formulas-d_455.html

[12] Harris Sarah, Harris David: *Digital Design and Computer Architecture: ARM Edition*. Morgan Kaufmann, 2015, ISBN 0128000562.

[13] HPC Study: *Permoník [online]*. [Online; navštíveno 8.5.2016].
URL http://studium.it4i.cz/en/superpocitace/#!

[14] ISO: *Programming Language C++*. 2014,ISO/IEC 14882:2014.

[15] Koninklijke Philips Electronics N.V.: *I2C – What's That? [online]*. [Online; navštíveno 29.4.2016].
URL http://www.i2c-bus.org

[16] Koninklijke Philips Electronics N.V.: *I2C MANUAL*. 2003.

[17] MathWorld: *BBP Formula [online]*. [Online; navštíveno 25.4.2016].
URL http://mathworld.wolfram.com/BBPFormula.html

[18] MathWorld: *Pi [online]*. [Online; navštíveno 9.5.2016].
URL http://mathworld.wolfram.com/Pi.htmll

[19] MCU Programmer: *8051 mcu, von Neumann vs Harvard Architectures [online]*. [Online; navštíveno 24.4.2016].
URL http://mcu-programming.blogspot.cz/2007/04/8051-mcu-von-neumann-vs-harvard.html

[20] Microchip Technology: *Get ready to see a new world of 8-bit PIC® MCUs [online]*. [Online; navštíveno 25.4.2016].
URL http://www.microchip.com/design-centers/8-bit

[21] Microchip Technology: *PIC10F200 introduction [online]*. [Online; navštíveno 24.4.2016].
URL http://www.microchip.com/wwwproducts/en/PIC10F200

[22] Noergaard Tammy.: *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Newnes, 2005, ISBN 0750677929.

[23] Renesas Electronics Corporation: *78K Family [online]*. [Online; navštíveno 25.4.2016].
URL http://am.renesas.com/products/mpumcu/78k/index.jsp

[24] Roberts Eric: *RISC vs. CISC [online]*. [Online; navštíveno 20.4.2016].
URL http://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

[25] Ruud Vlaming: *Femto OS: RTOS for small MCU's like AVR [online]*. [Online; navštíveno 7.5.2016].
URL http://www.femtoos.org

[26] Schwarz Josef, Růžička Richard, Strnadel Josef: *Mikroprocesorové a vestavěné systémy*. 2006, [reg č.; CZ.04.1.03/3.2.15.1/000].

[27] Wikimedia: *I2C [online]*. [Online; navštíveno 29.4.2016].
URL https://commons.wikimedia.org/wiki/File:I2C.svg

[28] Xuru: *Online Polynomial Regression [online]*. [Online; navštíveno 1.4.2016].
URL http://www.xuru.org/rt/pr.asp

# Přílohy

# Seznam příloh

# Příloha A

# Obsah CD

## A.1 Technická zpráva

Tato složka obsahuje PDF soubor s touto technickou zprávou. Také jsou zde všechny potřebné soubory pro vytvoření PDF dokumentu použitím nástroje LaTeX.

## A.2 Zdrojové kódy programů

Tato složka obsahuje zdrojové kódy veškeré implementace, které jsou uloženy ve formě *ino* souborů, což jsou soubory pro vývojové prostředí *Arduino IDE* [1].

Tato složka také obsahuje zdrojové soubory knihovny *Arrays.h*, která obsahuje funkce pro práci s poli a která byla implementována pro potřeby této práce.

# Příloha B

# Manuál

## B.1 Fyzické propojení uzlů

Na vývojové platformě je *TWI* rozhraní připojeno na piny 2 a 3. *Start pin* je definován na pinu číslo 4. Napájení je připojeno na piny označené *5 V* a *GND*.

## B.2 Programování platformy

Pro naprogramování vývojové platformy *Arduino Leonardo* [2] je potřeba vývojové prostředí *Arduino IDE* [1], které je nainstalováno na PC, ke kterému je platforma připojena prostřednictvím *USB*. Vývojové prostředí samo detekuje připojené zařízení. Před naprogramováním je nutné nastavit programátor *AVRISP mk II* a typ vývojové desky *Arduino Leonardo*.