ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra počítačů

Paralelní řešič velkých soustav lineárních nerovnic ve sdílené paměti

Parallel solver of large system of linear inequalities using shared memory

bakalářská práce

Studijní program: Otevřená informatika
Studijní obor: Software

Vedoucí práce: doc. Ing. Ivan Šimeček, Ph.D.

Martin Pažout

Praha 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Pažout** Jméno: **Martin** Osobní číslo: **468953**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Paralelní řešič velkých soustav lineárních nerovnic ve sdílené paměti

Název bakalářské práce anglicky:

Parallel solver of large system of linear inequalities using shared memory

Pokyny pro vypracování:

1, Zadefinujte patřičné matematické pojmy
2, V literatuře nastudujte a implementujte Fourier-Motzkin metodu pro řešení soustav lineárních nerovnic.
3, Po poradě s vedoucím práce metodu optimalizujte, co se týče rychlosti a pamětové složitosti.
4, Proveďte paralelní implementaci s pomocí knihovny OpenMP
5, Výsledky porovnejte s teoretickými předpoklady

Seznam doporučené literatury:

1, DAGUM, L. a R. MENON. OpenMP: an industry standard API for shared-memory programming. IEEE Computational Science and Engineering. 5(1), 46-55. DOI: 10.1109/99.660313. ISSN 10709924. Dostupné také z: http://ieeexplore.ieee.org/document/660313/
2, FRITSCH, Richard. Parallel solver of large systems of linear inequalities. Praha, 2014. Diplomová práce. Czech Technical University, Faculty of Information Technology.
3, MAKARA, T. Vplyv druhov aritmetiky na Korovin/Tsiskaridz/Voronkovov algoritmus pre riešenie lineárnych nerovníc. Praha, 2014. Bakalářská práce. Czech Technical University, Faculty of Information Technology.
4, DANTZIG, George B a B Curtis EAVES. Fourier-Motzkin elimination and its dual. Journal of Combinatorial Theory, Series A, Volume 14, Issue 3, 1973. Dostupné také z: http://www.sciencedirect.com/science/article/pii/0097316573900046

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Ivan Šimeček, Ph.D., katedra počítačových systémů FIT

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **22.10.2018** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

doc. Ing. Ivan Šimeček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Prohlášení

„Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.“

Místo, Datum

Podpis autora práce

Poděkování

Chtěl bych poděkovat vedoucímu práce doc. Ing. Ivanu Šimečkovi, Ph.D. a panu Mgr. Branislavovi Bošanskému, Ph.D. za ochotu a pomoc při vedení mé bakalářské práce. Současně oceňuji možnost přístupu ke gridové infrastruktuře kterou mi poskytla virtuální organizace české Národní Gridové Iniciativy MetaCentrum.

Abstrakt

V rámci této práce byla provedena analýza, návrh a implementace paralelního řešiče velkých soustav lineárních nerovnic pomocí Fourier-Motzkinovy eliminace ve sdílené paměti, pomocí OpenMP. Obsahem této práce je popis problému, návrh jeho řešení a implementace paralelního řešiče. V závěru je zhodnocena efektivita implementovaného řešiče.

Klíčová slova lineární nerovnice, Fourier-Motzkin eliminace, paralelní, OpenMP, C++

Abstract

This thesis focuses on analysis, design and implementation of parallel solver for large systems of linear inequalities using the Fourier-Motzkin elimination algorithm in shared memory using OpenMP. The content consists of a description of the problem, a treatise on the design and implementation of parallel solver and a detailed evaluation of its performance.

Key words linear inequalities, Fourier-Motzkin elimination, parallel, OpenMP, C++

Obsah

1. Úvod do problematiky

V této kapitole představíme teoretické podklady využívané v FME. Bude se jednak jednat o definici základních pojmů, které budou využívány v průběhu práce. Následně představíme samotnou metodu a demonstrujeme ji na uvedeném příkladu. Dále porovnáme cíle této práce s předešlými pracemi zaobírajícími se FME. Nakonec bude uvedeno využití FME v reálném světě.

1.1 Základní pojmy

Zde definuji základní pojmy, které se využívají v mé práci. Pokud se vedle názvy pojmu vyskytuje citace, jedná se o pojem převzatý z určité práce.

1.1.1 Soustava lineárních nerovnic

Definice převzata z [1]. Soustavu m lineárních nerovnic o n neznámých nad tělesem racionálních čísel definujeme takto

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &\leq b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &\leq b_2 \\ \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad &\vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n &\leq b_m \end{aligned}$$

kde $a_{1,1}, \cdots a_{m,n}$; $b_1, \cdots, b_m \in \mathbb{Q}$ a $x_1, \cdots, x_n$ jsou neznámé. Řešením soustavy je n-tice $(r_1, \cdots, r_n)$, $r_1, \cdots, r_n \in \mathbb{Q}$ právě tehdy, když po dosazení hodnot za příslušné neznámé $x_1 = r_1, \cdots, x_n = r_n$ dostaneme pro všechny nerovnice soustavy pravdivou nerovnost. Řekneme že, n-tice $(r_1, \cdots, r_n)$ řeší soustavu L. V případě, že soustava nerovnic nemá žádné řešení, řekneme, že soustava nerovnice je nesplnitelná, jinak splnitelná je.

Nechť značení $L(N)$ odpovídá soutavě lineárních nerovnic o N neznámích.

1.1.2 Menší rovno tvar nerovnice

Nechť *l* je lineární nerovnice, potom menší rovno tvar nerovnice je lineární nerovnice ve tvaru

$$a_1x_1 + a_2x_2 + \cdots + a_kx_k \leq b$$

Kde x_i je i-tá neznámá, a_i je skalár patřící k i-té neznámé a b je skalár na pravé straně.

1.1.3 Normalizace

Nechť existuje lineární nerovnice $a_1x_1 + a_2x_2 + \cdots + a_kx_k \leq b$ a existuje takové i, že $a_i \neq 0$, potom normalizaci $norm(l,i)$ této nerovnice definujeme jako

$$norm(l,i) = \frac{a_1x_1}{|a_i|} + \frac{a_2x_2}{|a_i|} + \cdots + x_i + \cdots + \frac{a_kx_k}{|a_i|} \leq \frac{b}{|a_i|}$$

1.1.4 Silnější hranice

Definice převzata z [2]. Nechť L je soustava lineárních nerovnic (1.1.1) $L = \{x_i \geq t, x_i \geq \text{ť}\}$, kde $t, \text{ť} \in L(N-1)$ a to tak že x_i není součástí t ani ť. Pak

$$\text{ť} > t \Leftrightarrow L \equiv x_i \geq \text{ť}$$

Obdobně, nechť Ľ je další soustava lineárních nerovnic: $Ľ = \{t \geq x_i, \text{ť} \geq x_i\}$. Pak

$$\text{ť} < t \Leftrightarrow L \equiv x_i \geq \text{ť}$$

1.1.5 Horní mez lineární nerovnice

Definice převzata z [1]. Nechť l je lineární nerovnice v normalizovaném tvaru dle i je i-tá neznámá potom je horní hranice rovna

$$upper(l,i) = \begin{cases} a_1x_1 + a_2x_2 + \cdots + a_{i-1}x_{i-1} + a_{i+1}x_{i+1} + \cdots + a_kx_k - b & a_i < 0, \\ \infty & jinak. \end{cases}$$

1.1.6 Dolní mez lineární nerovnice

Definice převzata z [1]. Nechť l je lineární nerovnice v normalizovaném tvaru dle i je i-tá neznámá, potom je horní mez rovna

$$lower(l,i) = \begin{cases} -(a_1x_1 + a_2x_2 + \cdots + a_{i-1}x_{i-1} + a_{i+1}x_{i+1} + \cdots + a_kx_k) + b & a_i > 0, \\ -\infty & jinak. \end{cases}$$

1.1.7 SN je řešitelná

Definice převzata z [3]. Nutná podmínka k řešitelnému systému nerovnic(1.1.1) je, že neexistuje žádná množina proměnných ($y_1 \geq 0,\ y_2 \geq 0, \cdots, y_m \geq 0$) takových, že

$$\sum_{i=1}^{m} y_ib_i > 0,\ \sum_{i=1}^{m} y_ia_{ij} = 0 \qquad j = (1, \cdots, n)$$

1.1.8 Konvexní množina

Definice převzata z [4]. Množina $S \subseteq \mathbb{R}^n$ je konvexní pokud každá úsečka spojující každé dva vrcholy množiny S celá leží v S. Tedy pro $s_1, s_2 \in S$ a pro jakékoliv $\propto$, kde $0 \leq \propto \leq 1$, dostaneme $\propto s_1 + (1-\propto)s_2 \in S$

1.1.9 Konvexní Polyhedron

Definice převzata z [4]. Polyhedron je konvexní množina, která je daná průnikem konečného počtu nadrovin a polorovin, nebo jako konvexní kombinace konečného počtu vrcholů a paprsků.

1.1.10 Souběh

Podle [5] definujeme souběh takto. Souběh (race condition) je situace, kdy při přístupu dvou nebo více procesů ke sdíleným datům dojde k chybě, přestože se každý z procesů samostatně chová korektně. K chybě dochází díky tomu, že data jsou modifikována některým procesem v době, kdy s nimi jiný´ proces provádí několik operací, o kterých se předpokládalo, že budou provedeny jako jeden nedělitelný celek.

1.1.11 Atomická proměnná

Pro definici atomické proměnné využijeme popis významu implementace atomicity v OpenMP [6]. Atomická proměnná je taková proměnná, která automatizuje přístup k specifické paměti. Zaručuje, že nedojde k souběhu pomocí přímého ovládání souběžných vláken, která by mohla zapisovat nebo číst z konkrétní paměti.

1.1.12 Jedna fáze FME

Jako jednu fázi algoritmu chápeme aplikaci jednotlivých kroků algoritmu k eliminování jedné neznámé.

1.1.13 Validní cesta průchodu

Jako cestu si představíme posloupnost fází FME (viz def. 2.1.12), která vede na eliminaci neznámých až na jednu. Cesta je validní právě tehdy, když poslední neznámá nebyla dosud vyhodnocena, tedy nebylo pro ni už nalezeno řešení.

1.1.14 Počítačový cluster

Počítačový cluster [7] je seskupení volně vázaných počítačů, které spolu úzce spolupracují, takže se na venek mohou tvářit jako jeden počítač.

1.2 Cíl práce:

Cílem práce je napsání paralelního řešiče SN pomocí metody FME využívající OpenMP API. Aby toto řešení bylo jak časově, tak paměťové efektivní. Druhým úkolem je pak vymyslet heuristiku průchodu stromu, který FME vytváří, zefektivnit tak o to víc metodu odstraněním duplicitních výpočtů. Dalším elementem práce je prioritní průchod stromem, tak abychom vytvořili co nejméně nerovnic. Na konci pak porovnat výsledky s teoretickými předpoklady.

1.3 Fourier-Motzkin eliminace:

FME je algoritmus pro řešení soustavy lineárních nerovnic. Je to rozšíření Gaussovi eliminační metody, kdy se za pomocí sčítání vhodných násobků rovnic postupně eliminují jednotlivé proměnné. Prvně byla popsána francouzským matematikem J. B. J. Fourier v roce 1823 a později znovuobjevena americkým matematikem T. S. Motzkinem, a to nezávisle. Následující text vychází z [3].

Máme-li soustavu lineárních nerovnic L, která má m nerovnic s n neznámými. Což si lze představit jako

$$Ax \leq b$$

Kde $A \in \mathbb{R}^{m\times n}$ je matice tvořená jednotlivými skaláry před neznámými, vektor $x \in \mathbb{R}^n, x = (x_1,\ x_2,\ \dots,\ x_n)$ znázorňuje jednotlivé neznámé a vektor $b \in \mathbb{R}^m$, který odpovídá pravým stranám nerovnic.

Nerovnice z této soustavy můžeme rozdělit dle první neznámé do tří částí:

L_+ nerovnice s kladným x_1

L_- nerovnice se záporným x_1

L_0 nerovnice s nulovým x_1

Tedy

$$L_+ \qquad \beta_i x_1 + a_i\bar{x} \leq b_i \qquad i = 1,\cdots,m_1$$

$$L_- \qquad -\beta_j x_1 + a_j\bar{x} \leq b_j \qquad j = m_1+1,\cdots,m_2$$

$$L_0 \qquad a_k\bar{x} \leq b_k \qquad k = m_2+1,\cdots,m$$

Kde $\boldsymbol{a_r\bar{x}}$ reprezentuje zbytek r-té nerovnice, $\boldsymbol{a_r\bar{x}} = a_2x_2 + \cdots + a_nx_n$, $\boldsymbol{b_r}$ je r-tý skalár pravé strany nerovnice a $\boldsymbol{\beta_r}$ je r-tý násobek x_1.

L_+ a L_- následně normalizujeme a dostaneme

$$L_+ \qquad x_1 + \frac{1}{\beta_i}a_i\bar{x} \leq \frac{1}{\beta_i}b_i \qquad i = 1,\cdots,m_1$$

$$L_- \qquad -x_1 + \frac{1}{\beta_j}a_j\bar{x} \leq \frac{1}{\beta_j}b_j \qquad j = m_1+1,\cdots,m_2$$

$$L_0 \qquad a_k\bar{x} \leq b_k \qquad k = m_2+1,\cdots,m$$

Z L_+ si uložíme horní mez *Upper(L_+)* a obdobně z L_- si uložíme dolní mez *Lower(L_-),* využijeme je pro zpětné reálné řešení soustavy.

Teď eliminujeme x_1, takže sečteme všechny nerovnice z L_+ se všemi nerovnicemi z L_- a dostaneme tak novou soustavu rovnic $\hat{L}$, do které ještě přidáme L_0. V $\hat{L}$ se již nevyskytuje x_1 a má tedy o jednu neznámou méně.

Na novou soustavu znovu aplikujeme předchozí kroky (seřazení, normalizace, …), eliminujeme tak další neznámou. Eliminujeme dokud:

- neeliminujeme všechny neznámé
- nezjistíme, že soustava nemá řešení.
- Zjistíme, že jich má nekonečné mnoho

Můžeme požadovat dva výsledky:

- Reálné řešení soustavy, to dostaneme pomocí zpětného dosazení uložených mezí
- Intervaly pro všechny proměnné odpovídající projekci na patřičnou osu

V průběhu algoritmu bychom také měli kontrolovat **nerovnice bez neznámé** tedy nerovnice tvaru $0 \leq \beta$, kde β je nějaký skalár. Pokud bude $\beta < 0$ můžeme prohlásit, že soustava nemá řešení.

1.4 Řešený příklad:

Jako obraz vydá za tisíc slov, tak lze algoritmus lépe pochopit pomocí příkladu jeho použití. V této části tedy popíšu postup FME na příkladu, a to navíc s podrobněji popsanými kroky.

Máme vždy soustavu ve tvaru $Ax \leq b$, a to můžeme zajistit pomocí převodu na menší rovno tvar nerovnice (reference na definici)

Budeme eliminovat podle x

$-2x + 2y \leq 3$

$2x + 4y \leq 6$

$8x + 0y \leq 7$

$0x - 3y \leq 1$

Podle toho jak jsme si FME definovali, provedeme **rozřazení**:

$\boldsymbol{L_+}$: $2x + 4y \leq 6$

$8x + 0y \leq 7$

$\boldsymbol{L_-}$: $-2x + 2y \leq 3$

$\boldsymbol{L_0}$: $0x - 3y \leq 1$

Radí se podle skaláru před x (+, -, 0)

Provedeme **normalizaci** podle x**:**

L_+: $x + 2y \leq 3$

$x + 0y \leq \frac{7}{8}$

L_-: $-x + y \leq \frac{3}{2}$

Všimněte si, že normalizuji pouze L_+a L_-, a to proto, že pro L_0 není potřeba, protože nefiguruje při eliminaci proměnné x, nemá tedy smysl. Význam normalizace je zřejmý, při sčítání nerovnic se eliminuje x, a to je velmi žádoucí v dalším kroku algoritmu, eliminaci.

Uložíme si horní a dolní meze (1.1.5 a 1.1.6). Používají se při hledání řešení soustavy a to tak, že vybereme určitou hodnotu z intervalu proměnné z větší hloubky průchodu a takto rekurzivně budeme dosazovat dokud „neprobubláme“ nahoru a dostaneme tak jedno řešení. Pokud hledáme pouze intervaly pro jednotlivé proměnné horní a dolní hranici si ukládat nemusíme.

Provedeme **eliminaci**:

$\hat{\boldsymbol{L}}$: $3y \leq \frac{9}{2}$

$2y \leq \frac{31}{8}$

$-3y \leq 1$

Můžeme si všimnout, že se nám počet rovnic zmenšil, to se ale stalo pouze proto, že máme pouze jednu rovnici v L_- tento stav nastává zřídka, většinou se naopak počet rovnic s přidávanou hloubkou prudce zvyšuje.

Teď by nastal krok **očíslování** a pokračovali bychom **rozdělením** pro nové neznámé. Ale jelikož jsme eliminovali všechny neznámé až na jednu, máme tedy **soustavou nerovnic o jedné neznámé**, dostaneme již projekce y na osu y. Nejprve nerovnice **normalizujeme:**

$\hat{\boldsymbol{L}}$: $y \leq \frac{3}{2}$

$y \leq \frac{31}{16}$

$-y \leq \frac{1}{3}$

A vybereme nejsilnější hranici (viz: 1.1.4) a dostaneme že $y \in [-\frac{1}{3}, \frac{3}{2}]$. Zde FME končí, pokud bychom chtěli reálné řešení soustavy, vybereme nějakou hodnotu z tohoto intervalu a dosadíme ji do horních a dolních mezí a znovu vyřešíme soustavu nerovnic o jedné neznámé.

1.5 Předešlé práce týkající se FME:

V této části rozebereme předešlé práce, ve kterých figuruje metoda FME. Dále je zde popsáno, o čem zmíněné práce pojednávali a jak se od nich tato práce liší.

1.5.1 Efektivní řešič velkých soustav lineárních nerovnic

Tato práce [8] je první, která se FME zabývala a je proto podmnožinou prací následujících. V práci se nejdříve rozebere obecný matematický postup řešení nerovnic a zobrazí se patřičné grafické znázornění, které SN popisuje. Následně je popsána a implementována sekvenční verze FME.

Od mé práce se liší jednak tím, že je napsána v programovacím jazyku Java a je sekvenční. Také práce není moc podrobná, co se týče implementace a neobsahuje žádné pseudokódy popisující jednotlivé algoritmy z kterých se FME skládá.

1.5.2 Vplyv druhov aritmetiky na Korovin/Tsiskaridz/Voronkovov algoritmus pre riešenie lineárnych nerovníc

Tato práce [1] se zabývá především implementací Conflict resolution algortimu, který také řeší SN. FME je zde jen použita pro porovnání. Je zde také kladen důraz na aritmetiku, ve které si budeme SN řešit. Respektive s jakou chybou budeme počítat, když nepoužijeme přesnou racionální aritmetiku, která je řešená na programové úrovni a vede tedy k pomalejšímu řešení. Ale použijeme aritmetiku s pohyblivou desetinou čárkou, kde nastávají zaokrouhlení a ztrácíme tedy přesnost.

Má práce bude zabývat pouze FME a bude tedy popsána do větší hloubky. Má práce bude zároveň paralelní a bude řešit pouze hodnoty typu double, a to na úkor přesnosti.

1.5.3 Parallel solver of large systems of linear inequalities

Tato práce [9] se zabývá paralelní implementací FME s použitím knihovny Message Passing Interface. Jedná se tedy o distribuovanou paralelizaci. Jsou zde také popsány obecné postupy, které se při distribuované paralelizaci používají.

Má práce bude paralyzována pomocí OpenMP, tedy ve sdílené paměti. Také bude práce řešit určitý chytrý průchod stromem, který metoda vytváří, a to pomocí určité heuristiky.

1.5.4 Modified Fourier-Motzkin Elimination Algorithm for Reducing Systems of Linear Inequalities with Unconstrained Parameters

Práce [2] pojednává o implementaci modifikované FME, která využívá získané horní a dolní hranice, na základě kterých poté určí, v jakém kvadrantu nerovnice leží, a pokud do tohoto kvadrantu nezasahuje obor hodnot (neleží tam část polyhedronu) určí nerovnici jako redundantní. Jedná se o takový geometrický přístup k hledání redundancí nerovnic.

Od mé práce se liší již jenom zásahem do úpravy FME a také není paralelní.

1.5.5 Using Fourier-Motzkin-Elimination to Derive Capacity Models of Container Vessels

Práce [10] vytváří framework pro eliminaci proměnných, které nemají velký vliv na celkový výsledek. Důležitou částí frameworku je také předběžné zpracování, ve kterém řadou metod eliminujeme redundantní nerovnice. Framework je postaven pro LP problém efektivního převozu nákladu.

Od mé práce se liší jednak tím že se týká především vytvořením frameworku a FME je používáno v sekvenční verzi.

1.5.6 Fourier-Motzkin with Non-Linear Symbolic Constant Coefficients

Práce [11] pojednává o rozšíření FME využívané ke generování hranic for cyklů, která se dají popsat pomocí polyhedronu. Polyhedron ale musí být afinní, což se často nestává. Práce [11] tedy rozšířila FME tak aby dokázala brát v potaz symbolické proměnné při generaci hranic for cyklů a vystavila k tomu patřičný Framework.

Opět se práce nezabývá primárně FME, ale jejím rozšířením pro speciální případ a jedná se o sekvenční verzi.

1.6 Reálná aplikace FME

FME je především známá jako teoretický nástroj pro řešení LP, ale její průchod soustavou nerovnic umožňuje i její jinou aplikaci.

1.6.1 Integer linear programing

Další reálná aplikace zmíněná v [12] je analýza závislosti přístupu do pole dat, která potřebuje najít integery, které splňují soustavu rovnic a nerovnic, tedy zjistit zda existuje integerové řešení. Rovnice odpovídají maticím a vektorům reprezentujícím přístup a nerovnice vycházejí z hranic for cyklů. Využíváme zde skutečnost, že rovnice $x = y$ se dá přepsat na $x \leq y$ a $x \geq y$.

Integer linear programing je známý NP těžký problém a řešení pomocí heuristik není občas možné, a tak se nám nabízí použití FME na tento problém.

1.6.2 Určení pořadí exekuce for cyklu

V příkladu z [12] se aplikuje FME na dvojici nezávislých for cyklů

```
1. for ( i = 0; i <= 5; i++)
2.     for ( j = i ; j <= 7; j++)
3.         Z [j , i ] = 0;
```

Kód 1 Dvojice for cyklů před uspořádáním pomocí FME

Abychom zlepšili prostorovou lokalitu, můžeme uspořádat hranice for cyklů tak aby se k poli přistupovalo v lexikografickém pořadí.

```
1. for ( j = 0; j <= 7; j++)
2.     for ( i = 0; i <= min (5 , j ); i++)
3.         Z [j , i ] = 0;
```

Kód 2 Dvojice for cyklů po uspořádání pomocí FME

Tyto nové hranice for cyklu můžeme najít pomocí projekce konvexního polyhedromu (viz def. 1.1.9), který reprezentuje iterační prostor jako vnější dimenzi prostoru. Tato projekce se dá vypočítat pomocí FME.

Jak je vysvětleno v [12], obecně se při generaci hranic for cyklu opakovaně přepisují všechny nerovnice, co obsahují daný vnější index na horní a dolní hranice (viz def. 1.1.5 a 1.1.6) dokud nejsou nalezeny hranice pro všechny proměnné. Takhle se eliminuje dimenze reprezentující vnější cyklus a dostaneme Polyhedron s dimenzí o jednu menší.

1.6.3 Eliminace redundancí

V pracích [2] a [10] se z FME využívá především jeho průchod systémem nerovnic a využívá se k tomu, abychom detekovali, zda je daná nerovnice redundantní, tedy zda se dá eliminovat. Samozřejmě čím více redundancí eliminujeme, tím rychlejší bude výpočet.

V práci [2] se využívá geometrického přístupu, a to takového, že se pomocí FME získají horní a dolní hranice, podle kterých se určí kvadranty, a poté se testuje, zda daná nerovnice leží vůbec v kvadrantu, ve kterém leží Polyhedron.

V práci [10] se vytvořil Framework, který se používá na statický test dat, která obsahují proměnné, které nemají velký vliv na celkový výsledek a samozřejmě i redundance. Framework má tedy za úkol všechny tyto redundance a proměnné s malým vlivem na výsledek eliminovat. V jedné své části využívá průchodu FME a v jejím průběhu eliminuje nepotřebné proměnné.

2. Analýza a design

2.1 Analýza složitosti

2.1.1 Průchod FME

FME řeší vždy soustavu nerovnic, kde každou nerovnici postupně upravuje až do eliminace, kde se vytvářejí nové nerovnice bez jedné neznámé. Počet nových nerovnic odpovídá rovnici

$$L_+ * L_- + L_0 \tag{1}$$

Nové nerovnice se také dostanou do eliminace, a to dokud nezůstane pouze jedna neznámá, ze které se pak určí interval.

2.1.2 Výpočetní složitost

Z průchodu FME je vidět, že hlavní složitost algoritmu leží právě v kroku eliminace ve výše uvedené rovnici (viz rov. 1). Úpravy, které se provádí na nerovnice lze schovat do konstanty. Nejhorší případ pro m nerovnic bude:

$$|L_0| = 0, |L_+| = |L_-| = m/2 \tag{2}$$

pro první eliminaci se pak vygeneruje $m^2/4$ nerovnic.

Celkový počet pro m nerovnic o n neznámých bude asymptoticky odpovídat

$$T(n,m) = O(\sum_{r=0}^{n-1}(n-r)\frac{m^{2^r}}{4^{2^r-1}}) \tag{3}$$

2.1.3 Paměťová složitost

Jelikož se při každé eliminaci, která vlastně odpovídá součtu dvou nerovnic, vytvoří nerovnice nová, tak to znamená, že v průběhu FME se vytvoří při nejhorším případě

$$O(\sum_{r=0}^{n-1}(n-r)\frac{m^{2^r}}{4^{2^r-1}}) \tag{4}$$

reprezentací nerovnic v paměti. Pomocné proměnné lze z asymptotického hlediska zanedbat, hlavní paměťové nároky zabere uložení nerovnic.

2.1.4 Hlavní zpomalení algoritmu

Práce s pamětí je velmi časově náročná, a jak vyplývá z předchozího bodu, počet nerovnic nám exponenciálně roste. To znamená, že program většinu času stráví alokováním paměti. Vhodná alokace paměti bude kritická pro celkový výkon programu. Rychlost programu bude udávat rovnice (viz rov. 1), kterou se budeme snažit minimalizovat.

Velkou roli tedy bude hrát schopnost eliminace počtu nežádoucích nerovnic, tzn. nerovnic, které nepřispívají k novým výsledkům. S více nerovnicemi exponenciálně roste počet nerovnic, proto se nám s méně nerovnicemi celkový počet nerovnic exponenciálně sníží. Z tohoto důvodu se vyplatí eliminovat nežádoucí nerovnice co nejdříve.

2.2 Datové struktury

Jak bylo avizováno v analýze složitosti tak správa paměti bude kritická pro celkovou funkci programu. Důležitým faktorem je také to, že musíme ukládat předchozí nerovnice, pomocí nichž se reprezentuje daná konvexní polyedrická množina, která vymezuje řešení dané soustavy.

2.2.1 Nerovnice

Pro samotnou reprezentaci nerovnic je vhodné použití nějaké formy pole, které si bude udržovat jednotlivé násobky neznámých na patřičném indexu. Díky tomuto rozvržení bude přístup k násobkům neznámých dosažen v konstantním čase a to $O(1)$.

Alternativou je použít strukturu na bázi listu, kde by si jednotlivé prvky listu udržovali jak násobek neznámé, tak i její index. Přístup na index by zde byl $O(n)$, kde n je počet neznámých. Potenciálně bychom ale zabrali méně paměti, jak při postupné eliminaci neznámých a zejména u řídkých nerovnic. Obdobná reprezentace je používána například v programu matlab.

2.2.2 Soustava nerovnic

Soustava nerovnic se intuitivně dá uložit do dynamického pole, které se zvětšuje při dalších krocích algoritmu. Dle Kesslera [13] se do tohoto pole nebudou ukládat samotné nerovnice, ale spíše ukazatele na ně. Jelikož v průběhu algoritmu musíme nerovnice seřadit, tak záměna ukazatele je rychlejší než překopírování celé nerovnice.

Alternativně se místo dynamického pole dá použít struktura na bázi listu. A to tak že pro každou fázi eliminace si vytvoříme jeden uzel, kde budou uloženy dané nerovnice v dynamickém poli. Výhodou listu je to, že v paměti budeme mít více menších bloků paměti, než jeden velký. A zároveň nebude nutné neustále přealokovávat předchozí nerovnice.

2.2.3 intervaly řešení

Jeden interval řešení se dá reprezentovat mnoha způsoby, ale pro naše účely nám ho bude dostatečné reprezentovat ho pomocí dvojice čísel. Jedno pro dolní hranici a druhého pro horní. Pro interval je také dobré vědět, z jakého uzlu bylo řešení vyvozeno, a proto si také ukládáme ukazatel na předcházející uzel.

2.3 Sekvenční algoritmus

Sekvenční algoritmus založený na algoritmu od Kesslera [14] se skládá z pěti kroků: seřazení, normalizace, eliminace, uložení nerovnic a nalezení intervalu. Tyto kroky se aplikují na soustavu nerovnic, aplikaci těchto kroků nazveme fází (viz def. 2.1.12). Postupně eliminujeme neznámé a to až do chvíle, než zůstanou nerovnice pouze s jednou proměnnou, ze kterých se následně určí výsledný interval hodnot. Vše se točí okolo eliminace, pro kterou si připravujeme nerovnice tak aby byla co nejefektivnější. Výsledkem algoritmu je interval hodnot pro danou poslední neeliminovanou proměnnou.

```
Vstup: systém nerovnic L
procedure Solve(L)
  while počet neeliminovaných nerovnic je > 1 do
    i ← neznámá která se bude eliminovat
    l ← seřadíme nerovnice dle typu neznámé na pozici i
    normalizujeme nerovnice l dle neznámé na pozici i
    new_l ← předalokujeme nové pole
    foreach pár a ∈ L+ a b ∈ L− do
      vytvoříme novou nerovnost a - b a uložíme ji do new_l
    end
    uložíme si původní nerovnice l
    l ← new_l
    l teď obsahuje nerovnice bez neznámé i
  end
  for každou nerovnost e ∈ l do
    if e je kladná dle i then
      min ← minimální hodnota z předchozího minima a pravé strany e
    else if e je záporná dle i then
      max ← maximální hodnota předchozího maxima a pravé strany e
    end
  end
end
```

Pseudokód 1 Sekvenční algoritmus

2.3.1 Seřazení nerovnic

Nerovnice mohou být tří typů kladné, záporné a nulové. Daný typ se určí dle indexu, který chceme v dané fázi eliminovat. Abychom si zrychlili i ulehčili práci v kroku eliminace, kde vytváříme z kladných a záporných nerovnic nerovnice nové, seřadíme si nerovnice v soustavě. Nerovnice budou seřazeny tak, aby „šel“ patřičný typ vždy za sebou. Jelikož máme tři kritéria, musíme soustavu nerovnic projít dvakrát.

2.3.2 Normalizace nerovnic

Normalizace nerovnice spočívá ve vydělení násobkem neznámé na daném indexu. Získáme tak nerovnice, které budou mít jedničku na daném indexu, díky tomu se eliminace neznámé změní na pouhý součet dvou nerovnic. Normalizaci použijeme pouze na kladné a záporné nerovnice, pro nulové nerovnice normalizace nemá smysl, ba naopak docházelo by k dělení 0.

2.3.3 Eliminace jedné neznámé

Díky tomu, že jsme si již předpřipravili data, samotná eliminace je triviální. Eliminace pouze sečte každou kladnou nerovnici s každou zápornou. Vytvoří se tak nové nerovnice, které mají o jednu neznámou méně.

2.3.4 Uložení nerovnic

V tomto kroku se uloží soustava nerovnic použitá v dané fázi, a připraví se soustava nová, pro následné použití. Do nové soustavy nerovnic se také nakopírují nerovnice nulové, protože již pro další neznámé nulové být nemusejí.

2.3.5 Nalezení intervalu

Nerovnice v této fázi obsahuje již poslední neznámou a díky normalizaci je její násobek 1,-1 nebo 0. Díky této vlastnosti nerovnic nalezení intervalu spočívá pouze v projití soustavy nerovnic. Z kladných nerovnic vybíráme horní hranici intervalu, a ta bude odpovídat minimální hodnotě pravé strany nerovnice. Ze záporných nerovnic vybíráme dolní hranici intervalu, a ta bude odpovídat maximální hodnotě pravé strany nerovnice. Nulové nerovnice vynecháme. Můžeme tak učinit, jelikož nerovnice jsou reprezentované v menší rovno tvaru (viz def. 1.1.2).

2.4 Paralelizace

K implementaci paralelizace jsem zvolil OpenMP. Dle [15] je OpenMP množina direktiv procesoru a rutin knihovny, která rozšiřuje Fortran. Takto pomocí OpenMP dokážeme napsat paralelní programy, aniž bychom museli kód koncipovat přesně pro naše dané prostředí.

Samotná paralelizace programu vychází z [13]. Z průchodu FME je vidět, že se algoritmus dělí na fáze (viz def. 2.1.12). Jedna fáze se dá chápat jako eliminace jedné neznámé, kdy se posuneme o stupeň blíže k výsledku. Každá následující fáze závisí na kroku předešlém. Nezávislost dat je tedy právě na úrovni jednotlivé fáze a proto zde budeme paralelizovat.

Paralelizovat budeme tedy jednotlivé kroky fáze. S tím, že krok uložení nerovnic paralelizovat nebudeme, jelikož se jedná pouze o alokování paměti.

2.4.1 Postupy paralelizace s pomocí OpenMP

Při paralelizaci se budou opakovat níže uvedené postupy. Jsou realizovány pomocí funkcionalit OpenMP a C++. Nejedná se zde o všechny principy, které paralelizace má, ale reprezentují množinu, která je využita v této práci.

2.4.1.1 Paralelizace for cyklu

For cyklus se pohybuje na intervalu daném horní a dolní hranicí. Paralelizace tedy spočívá v rozdělení tohoto intervalu patřičným vláknům. Rozdělení zajišťuje OpenMP pomocí direktiv $\#pragma\ omp\ parallel$ [16], paralelizuje blok dat, a $\#pragma\ for$ [17] rozdělí interval for cyklu.

2.4.1.2 Předalokované pole a atomická proměnná

Pokud předem známe výslednou velikost pole, do kterého budeme zapisovat. Využijeme toho, a rovnou pole předalokujeme. Pomocí pomocné proměnné, sloužící jako index, budeme do tohoto pole zapisovat. Pomocná proměnná musí být atomická, zabráníme tak potenciálnímu souběhu (viz def. 2.1.10).

V algoritmu nepoužíváme atomickou proměnnou, ale místo ní používáme OpenMP direktivu $\#pragma\ omp\ atomic\ capture$ [6], která ochranu proti souběhu zajistí.

2.4.1.3 Privátní proměnná

Pro každé vlákno vytvoříme vlastní proměnnou. Nedochází pak k neustálému přepisovaní, pokud bychom použili proměnnou s globálním rozsahem, které by vedlo ke špatným výsledkům.

OpenMP má tento proces zautomatizovaný, a to pomocí klíčového slova $private(proměnná)$ [18], které přidáme argumentům při vytváření a rozdělení práce jádrům.

2.4.1.4 Redukce

Redukce $reduction\ (operator: list)$, je jedním z parametrů direktivy $\#pragma\ omp\ parallel$ [16], který pro každé vlákno vytvoří privátní proměnnou. Po dokončení úkolu, se na tyto proměnné po dvojicích aplikuje operátor, vytvoří se nová hodnota, která se pak dál propaguje. Propagujeme do té chvíle, než nám zůstane pouze jedna hodnota, a ta je výsledkem.

2.4.1.5 Std::move

Využíváme zde přesunovací sémantiku [19], která byla přidána v C++11. Pomocí $std::move$ zamezíme zbytečnému kopírování a dostaneme tak efektivnější kód.

2.4.2 Seřazení

Paralelizace seřazení je založená na základě předalokování pole, do kterého budeme jeden typ prvku zapisovat na začátek, a druhý na konec. Tento proces provedeme dvakrát a dostaneme tak seřazené pole dle tří typů prvků. Každé jádro má svou vlastní proměnnou a ta se využívá k indexaci do předalokovaného pole. Každé jádro má také globálně přístupné atomické proměnné (viz def. 2.1.11) start a konec, podle kterých se určí hodnota pro indexaci.

```
Vstup: systém nerovnic L, vybranou neznámou i
procedure Sort(L,i)
  tmp ← předalokujeme si pomocné pole
  for každou nerovnost n ϵ L do
    if n má nulovou neznámou i then
      idx ← dosud nepoužitý index ze začátku pole s
     s ← s + 1
    else
      idx ← dosud nepoužitý index z konce pole k
      k ← k +1
    end
    tmp[idx] ← n
  end
end
```

Pseudokód 2 Seřazení nerovnic dle vybrané neznámé

2.4.3 Normalizace

Normalizace se volá pouze na prvky, které nejsou mezi sebou závislé. A proto se mezi jádra rozdělí jednotlivé části pole, na jejichž prvky se pak zavolá funkce normalizace. Zde se dá použít vektorizace, jelikož ve skutečnosti voláme dělení na každý prvek v poli, který reprezentuje nerovnici.

```
Vstup: systém nerovnic L, vybranou neznámou i
procedure Normalize(L,i)
  for každou nerovnost n ϵ L do
    div ← absolutní hodnota násobku neznámé i
    for j ← 0 to délka nerovnice do
      n[j] ← n[j] / div
    end
  end
end
```

Pseudokód 3 Normalizace nerovnic dle vybrané neznámé

2.4.4 Eliminace

Eliminace spočívá v součtu všech možných dvojic kladných a záporných nerovnic. Tyto dvojice na sobě nejsou závislé, a proto je můžeme obdobně jako u normalizace přerozdělit mezi vlákna. Také zde využijeme skutečnost, že známe počet nových nerovnic a předalokujeme si pole. Toto pole budeme navíc indexovat pomocí atomické proměnné (viz def. 2.1.11). Vektorizace je zde také možná (sčítáme dvě pole hodnot).

```
Vstup: systém nerovnic L, vybranou neznámou i
Výstup: pole nových nerovnic
procedure Eliminate(L,i)
  ret ← předalokujeme si pomocné pole
  for každou nerovnost a ϵ L+ do
    for každou nerovnost b ϵ L_ do
      new_n ← a - b
      idx ← dosud nepoužitý index z pole k
      k ← k +1
      ret[idx] ← new_n
    end
  end
  return ret
end
```

Pseudokód 4 Eliminace vybrané neznámé

2.4.5 Nalezení intervalu

Při hledání intervalu procházíme pole nerovnic a hledáme minimální (maximální) hodnotu na pravé straně pro kladné (záporné) nerovnice. Pro řešení paralelizace tohoto problému je ideální použít redukci, kde najdeme z množiny nerovnic přidělené danému vláknu nejlepšího kandidáta, a toho pak zpětně porovnáváme s ostatními vlákny.

```
Vstup: systém nerovnic L, vybranou neznámou i
Výstup: horní a dolní hranice intervalu řešení
procedure FindBounds(L,i)
  for každou nerovnost n ϵ L do
    if n na indexu i je positivní then
      if min > levá strana n then
        min ← levá strana n
      end
    else if n na indexu i je negativní then
      if max < levá strana n then
        max ← levá strana n
      end
    end
  end
  return min, max
end
```

Pseudokód 5 Nalezení intervalu řešení pro danou neznámou

3. Realizace

3.1 Paměť

Při výběru jsem se snažil vybalancovat hranici mezi paměťovou náročností a rychlostí programu. Program je však stále velice náročný na paměť a pro stroje s malou pamětí RAM je víceméně nepoužitelný (nepoužíváme-li jej pouze na malé problémy). Třídy, co reprezentují patřičná data, zároveň přináší správu nad těmito daty a dodávají tak logiku do programu. Níže jsou uvedené základní třídy tvořící reprezentaci dat algoritmu.

3.1.1 Inequality

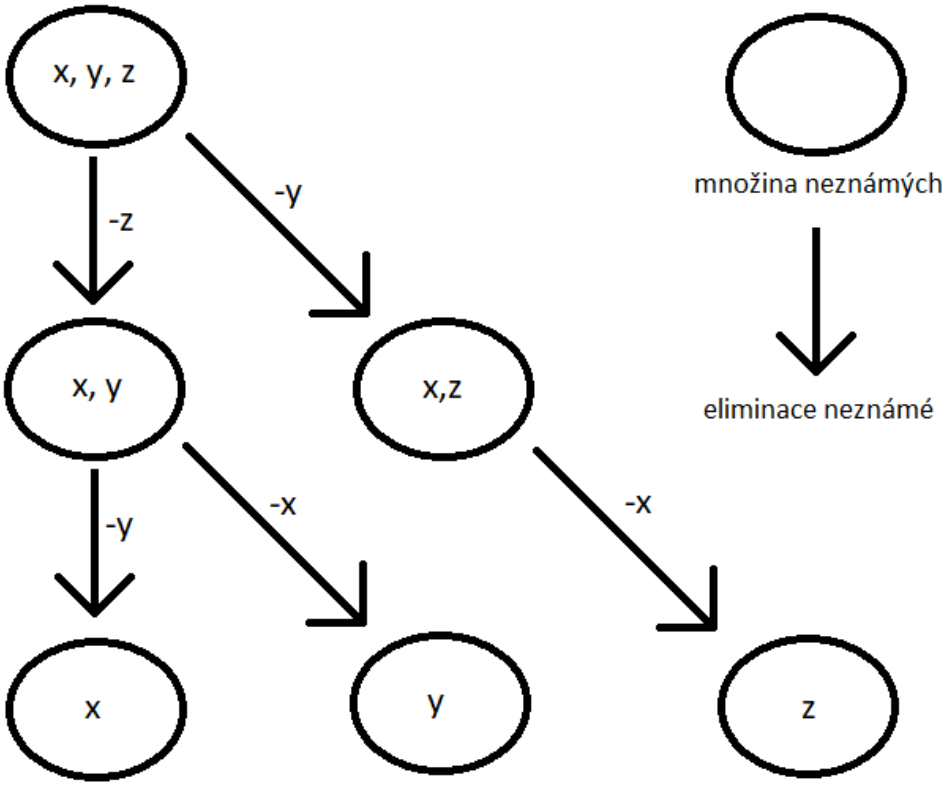
Jedná se o třídu reprezentující nerovnost. Pro reprezentaci hodnot násobků neznámých jsem se rozhodl pro $std::valarray$ [20], jejíž výhody jsou zmíněny v [21]. Hlavní důvod pro použití je, že podporují vektorové operace nad hodnotami, které, jelikož se jedná o třídu ze standartní knihovny C++, jsou optimalizované. Pravá strana nerovnice se ukládá na první pozici v $std::valarray$, což si můžeme dovolit, protože všechny změny hodnot se týkají i této hodnoty.

3.1.2 InequalityBox

Třída reprezentující soustavu nerovnic v dané fázi. V $std::vector$ [22] jsou uloženy ukazatele na třídu Inequality a to ukazatele $std::shared\_ptr$ [23]. Tento typ chytrého ukazatele jsem zvolil, protože se sám stará o dealokaci paměti, na kterou ukazuje a v mé implementaci dochází k ukládání ukazatelů, co ukazují na stejná data. Na tyto problémy je $std::shared\_ptr$ přímo dělaný a mimo jiné se z něj v C++ komunitě stává nový standart alokace paměti.

3.1.3 InequalityTree

Představíme si, že jeden uzel reprezentuje množinu neznámých, které chceme až na jednu eliminovat. Eliminujeme-li z uzlu jednu neznámou, dostaneme uzel nový. Z tohoto uzlu již nemůžeme dostat neznámou, kterou jsme eliminovali, ale můžeme dostat proměnné ostatní. Projdeme-li takto až na konec, kdy dostaneme první interval řešení, pak nemusíme začínat od znovu, ale naopak se jen vrátit na předchozí uzel a z něj získat interval pro hodnotu, kterou ještě neznáme.



Obr 1 Průchod založený na předešlých výpočtech. Vytvořil autor.

Třída InequalityTree tedy udržuje data v jednotlivých uzlech tohoto průchodu. Jednotlivé uzly si drží ukazatele jak na předka, tak na potomka. Nerovnice jsou uloženy v uzlu v ukazatelích na třídu InequalityBox. Uzly také obsahují pomocné pole eliminated, které se používá k určení, zda jsme již tuto neznámou neeliminovali.

3.2 Průchod algoritmu

Průchod algoritmu je silně spjat s třídou InequalityTree. Přidávání uzlů do stromu odpovídá průchodu algoritmu, tedy jednotlivým fázím eliminace. Díky tomu ušetříme paměť, jak bylo popsáno v InequalityTree (viz 4.1.3). Počet operací se také sníží (pro nalezení nového intervalu řešení nemusíme totiž začínat od začátku). Začít můžeme na předchozím uzlu a aplikovat algoritmus znovu. Předchozí uzel nesmí mít eliminované všechny cesty, kterými se z něj dá pokračovat. Pokud jsou všechny cesty eliminované, vybere se jeho předchůdce, a to dokud se nenajde vhodný uzel, nebo nenastane konec. Konec nastane, když nalezneme intervaly řešení pro všechny neznámé.

```
curr ← inicializujeme uzel prvnímy daty
while nejsou nenalezeny všechny intervaly do
  provede se fázi algoritmu
  posuneme se na následující uzel
  If uzel n nemá již žádné validní cesty or zbývá jedna neznámá then
    If nalezeny všechny intervaly then
      return
    else if zbývá jedna neznámá then
      najdeme a uložíme interval
    end
    curr ← posuneme na předchozí uzel, který má validní cesty
  end
end
```

Pseudokód 6 Průchod algoritmu

3.2.1 Heuristika

Pro danou neznámou ze všech nerovnic vypočítáme rovnici (viz rov 1) a získáme tak počet nerovnic, které by se při výběru eliminace této neznámé vytvořili.

```
vstup: soustava nerovnic L, neznámá i
vystup: počet vytvořených nerovnic když použijeme danou neznámou
procedure GreedyHeur(L ,i)
  for nerovnice n ∈ L do
    if n je kladná pro i then
      plus ← plus + 1
    else if n je záporná pro i then
      minus ← minus + 1
    else
      zero ← zero + 1
    end
  end
  return plus * minus + zero
end
```

Pseudokód 7 Heuristika nerovnice dle dané neznámé

3.2.2 Výběr neznámé k eliminaci

V průběhu algoritmu je nutné zvolit neznámou, kterou budeme eliminovat. Neznámé tedy ohodnotíme dle heuristiky a vybereme z nich tu, která je ohodnocena nejlépe. Musíme ale také brát v potaz, zda neznámou můžeme vůbec zvolit. Toto zjistíme pomocí dvou typů vektorů. První typ eliminated je obsažen v každém uzlu stromu a zaznamenává, jaké neznámé jsme již použili. Druhý typ finished obsahuje neznámé, pro které již máme interval řešení.

Pomocí vektorů eliminated ověříme, zda jsme již danou neznámou nepoužili. Pokud vektor finished není prázdný, vybereme z něj nejlepší neznámou. Můžeme tak učinit, jelikož se vracíme do předchozího uzlu, který nemá jinou validní cestu, než takovou, kterou dostane eliminací neznámých v finished vektoru.

```
vstup: soustava nerovnic L, doposud eliminovane neznámé elim, dokončené neznámé fin
vystup: počet vytvořených nerovnic když použijeme danou neznámou
procedure ChooseNext(L,elim,fin)
  for neznámá n ∈ fin do
    if n ∈ elim then
      pokračuj další neznámou
    end
    tmp ← GreedyHeur(L,n)
    if tmp < best then
      best ← tmp
      best_n ← n
    end
  end
  if fin není prázdný then
    return best
  end
  for i ← 0 to počet neznámých do
    if i ∈ elim then
      pokračuj další neznámou
    end
    tmp ← GreedyHeur(L,n)
    if tmp < best then
      best_n ← i
    end
  end
  return best_n
end
```

Pseudokód 8 Výběr neznámé k eliminaci

3.2.3 Přesun na následující uzel

Při dokončení jedné fáze se musíme přesunout na nový uzel, který má validní cesty. Pokud jsme na nevalidním uzlu, posuneme se o uzel zpátky, a to do té doby, než najdeme validní uzel. Jakmile jsme na validním uzlu, vybereme prvního validního potomka a na něj se posuneme.

```
Vstup: pole fin obsahující neznámé, pro které již máme výsledek, uzel curr
Výstup: zda uzel může být prozkoumán nebo ne
procedure CanBeExplored (curr, fin)
  for i ← 0 to počet neznámích do
    if neznámá i již nebyla eliminována then
      if i ∉ fin then
        return true
      end
    end
  end
  return false
```

Pseudokód 9 Logika validity daného uzlu

```
Vstup: aktuální uzel curr, hotové neznámé fin, uzly children  následující aktuální uzel
Výstup: zda uzel může být prozkoumán nebo ne
procedure Next (curr,fin,children)
  if fin obsahuje všechny neznámé then
    return
  end
  while CanBeExplored(curr,fin) neplatí do
    curr ← předchůdce curr
  end
  for uzly u ∈ children do
    if CanBeExplored(u,fin) then
      curr ← u
      return
    end
  end
end
```

Pseudokód 10 Posunu na následující validní uzel

3.3 Sekvenční řešič

Implementace sekvenčního řešení bude přímo odpovídat paralelní verzi. Využijeme skutečnosti, kdy používáme vzdálené stroje poskytované z metacentra, kde díky jejich frameworku mohu nastavit počet poskytnutých jader. Toto řešení nebude plně odpovídat vlastní implementaci sekvenčního řešení, OpenMP bude alokovat nová jádra místo ponechání hlavního. Toto zpoždění je ale zanedbatelné a dostaneme tak lepší porovnání zrychlení paralelizace jelikož ho aplikujeme přímo na algoritmus.

3.4 Paralelní řešič

Realizace paralelizace je přímočará, v algoritmu se vyskytuje několik for cyklů a ty se budou paralelizovat pomocí direktiv OpenMP. Paralelizace části kódu se bude opakovat, jak bylo avizováno v Postupy paralelizace s pomocí OpenMP (viz 3.4.1).

3.4.1 Seřazení nerovnic

Paralelizujeme oba for cykly (viz 3.4.1.1) zodpovědné za řazení ukazatelů nerovnic, využíváme předalokované pole v kombinaci s atomickou proměnnou (viz 3.4.1.2). Po prvním cyklu se jako pomocné pole použije pole původní, předchozí pomocné pole se stává hlavním. Do nově vzniklého pomocného pole přesuneme pomocí $std::move$ (viz 3.4.1.5) dosud neseřazené ukazatele na nerovnice a seřadíme dle druhého parametru. Jelikož máme seřazené pole uložené v původní dočasné proměnné, tak ho zapíšeme do aktuálního uzlu.

3.4.2 Normalizace nerovnic

Pro normalizaci stačí pouze paralelizovat jeden for cyklus (viz 3.4.1.1) a na daný prvek zavolat funkci normalize, která pole reprezentující nerovnici vydělí hodnotou, která odpovídá násobku vybrané neznámé.

Máme dvě implementace normalizace nerovnic, kde jedna využívá toho, že jsou nerovnice seřazené a můžeme tak aplikovat normalizace jen na nenulové nerovnice. Druhá implementace prochází nerovnice všechny.

3.4.3 Eliminace jedné neznámé

V eliminaci je dvojitý for cyklus, paralelizujeme zde pouze vnější for cyklus dle Paralelizace for cyklu (viz 3.4.1.1). Jelikož víme, kolik nových nerovnice bude vytvořeno, použijeme opět kombinaci předalokovaného pole a atomické proměnné k uložení nové nerovnice (viz 3.4.1.2). Vytvoření nové nerovnice je pouze sečtení dvou *std::valarray* [20], výsledek použijeme k vytvoření nové nerovnice, která se pak uloží do předalokovaného pole.

3.4.4 Nalezení intervalu řešení

Pro nalezení intervalu paralelizujeme for cyklus (viz 3.4.1.1), který prochází všechny nerovnice. Pro nerovnice v daném vláknu se pomocí redukce (viz 3.4.1.4) hledá nejmenší hodnota horní hranice a největší hodnota dolní hranice. Nejmenší horní hranice se určí z kladných nerovnic, a při redukci se hledá její minimum, tedy $reduction(min: proměná)$ [16]. Největší dolní hranice se určí ze záporných nerovnic a při redukci se hledá její maximum, tedy $reduction(max: proměná)$.

4. Měřeni

4.1 Data

Jelikož je pro FME nejlepší možná nerovnice taková nerovnice, která má co nejvíc nul. Tak použitá data mají radikální efekt na rychlost programu. Vztah mezi nerovnicemi také hraje roli, protože při eliminaci se vytváří součtem nerovnice nová, pokud mají nerovnice u násobku neznámých opačné hodnoty. Tak dostaneme novou nerovnici s velkým počtem nul.

Samotná data jsou uložena ve složce data, pod názvem dataXX (XX reprezentuje očíslování dat). Soubor začíná vždy dvěma čísly oddělenými mezerou, s tím že první udává počet nerovnic a druhé udává počet neznámých. Na následujících řádcích jsou dále uvedeny hodnoty násobků neznámých dané nerovnice, ty jsou odděleny mezerou a jdou postupně za sebou. Poslední hodnota je hodnota konstanty levé strany nerovnice.

Data jsou dvou typů, a to hustá a řídká data, tedy data co celkově obsahují 13% a 87% nul.

4.2 Konfigurace stroje

Na otestování rychlosti programu používám počítačový cluster (viz def. 2.1.14) nympha.meta.zcu.cz, který je k dispozici na webu Metacentra. Metacentrum, přesněji MetaVO, je jedna z virtuálních organizací české Národní Gridové Iniciativy MetaCentrum NGI a je přístupná všem pracovníkům a studentům členů sdružení CESNET.

Cluster nympha.meta.zcu.cz obsahuje 64 uzlů, kde každý má následující hardwarovou specifikaci:

- 32x Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz
- 192 GB RAM
- disk 1x 960 GB NVMe
- připojení Ethernet 1Gb/s, Infiniband FDR 56Gb/s

4.3 Testování

Pro otestování správnosti výsledku jsem vyšel z práce [9], kde je k porovnání rychlosti použit programovací jazyk R [24]. Zde se programovací jazyk R použije jako reference správného výsledku. V R je FME realizováno pomocí rozšiřujícího balíčku editrules [25], který obsahuje sekvenční FME. Testoval jsem pomocí porovnání výsledků obou řešení, a k získání intervalů pro všechny neznámé jsem aplikoval sekvenční algoritmus z FME z R několikrát.

Nastal zde ale problém, jelikož má implementace neklade důraz na přesnost, která nastává při řešení náročnější soustavy nerovnic k rozdílu hodnoty intervalů. Tímto problémem se zabývá práce [1]. Je to důsledkem reprezentace reálných čísel v C++ pomocí double, které nejsou perfektně přesné. Nicméně pro malé soustavy nerovnic dostáváme shodné výsledky.

```
P <- editmatrix(c( „předem připravená data“))´
#neznámá pro kterou chceme interval řešení
chosen = 4
# list variables to eliminate
elim <- c (  "x2" , "x1","x3" )
F <- P
for ( v in elim )
{
        F <- eliminate (F , v, fancynames=TRUE)
}
#normalizace
for(i in 1:dim(F)[1]) {
 div = F[i,chosen];
 if(div == 0){
  next
 }
 if(div < 0){
        div = -div
 }
 for(j in 1:dim(F)[2]) {
  F[i,j] = F[i,j]/div
 }
}
#příprava pravé strany nerovnice
C = getb(F)
#nalezení intervalu
min = 500000
max = -500000
for(i in 1:dim(F)[1]) {
        if(F[i,chosen] == 0){
                next
        }

        if(F[i,chosen] < 0){
                if(max < C[i]){
                        max = C[i]
      }
   }else{
                if(min > C[i]){
                        min = C[i]
                }
        }
}
```

Kód 3 Testovací skript použití sekvenčního FME z R

4.4 Ohodnocení

Jak je popsáno v [26], nejdůležitější vlastnost paralelního algoritmu je zrychlení a účinnost. Čas běhu paralelního algoritmu $T(n,p)$, kde n je velikost problému a p je počet procesorů, je definován jako celkový čas od začátku výpočtu až po ukončení posledního běžícího procesoru. Paralelní zrychlení je definováno jako

$$S(n,p) = \frac{SU(n)}{T(n,p)} \tag{5}$$

Kde $SU(n)$ je čas sekvenčního algoritmu. Paralelní účinnost je definována jako

$$E(n,p) = \frac{S(n,p)}{p} \tag{6}$$

Při samotném měření zrychlení získáme z času běhu sekvenčního algoritmu děleno časem paralelního běhu algoritmu. Obdobně paralelní účinnost získáme vydělením paralelního zrychlení počtem procesorů.

4.5 Výsledky

Výsledky jsem získal komunikací s clusterem pomocí plánovače PBS Professional [27], který vytváří a koriguje fronty patřičným clusterům. Pro uživatele metacentra je vyhrazeno úložiště dat, na které jsem nahrál soubory tvořící můj program a skript. Daný skript jsem přidal do fronty na zmíněný použitý cluster nympha.meta.zcu.cz. Po určitém čase plánovač přiřadil zdroje a spustil skript.

Skript pomocí CMake [28] zkompiloval program na přiděleném stroji a program spustil. Při kompilaci byly použity příznaky – $fopenmp$, $-O3$ a $-Wall$. Také bylo použito C++14, které je aktivováno pomocí – $std = c++14$. Program se pak na stroji spustil a výstup se zkopíroval zpět do mého uložiště.

Samotný čas běhu algoritmu se měřil získáním časové stopy před spuštěním řešení programu a po něm. Je tedy vynechán čas načtení dat a samotné uvolnění paměti.

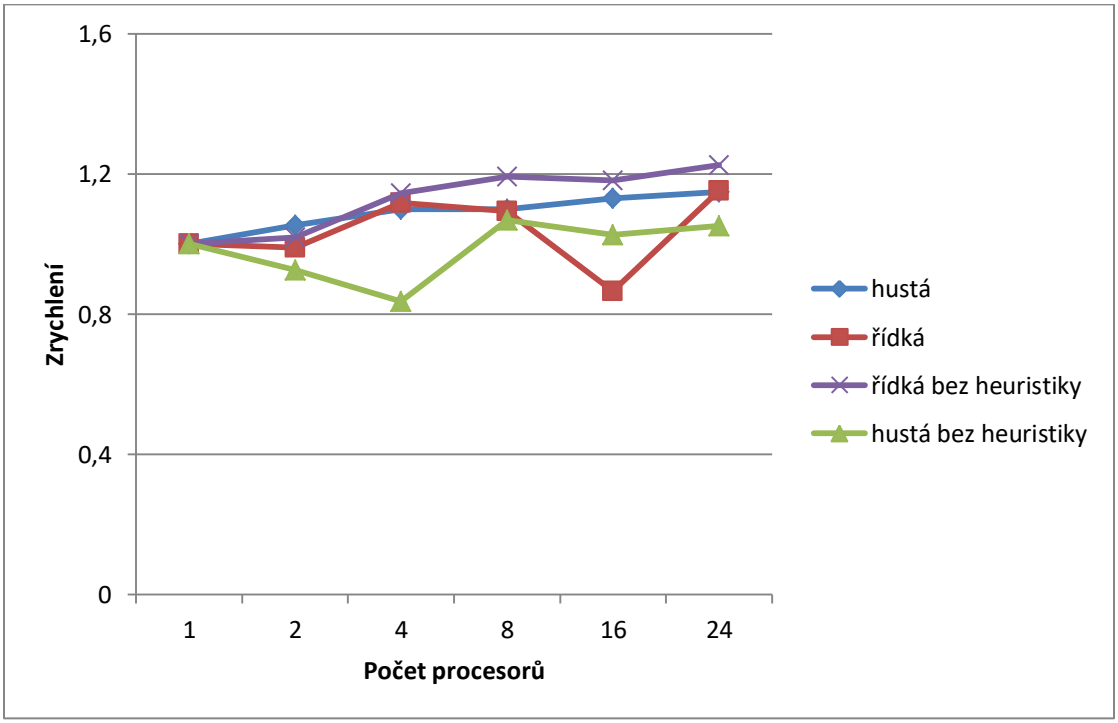
Husté nerovnice 16x5			
p	čas [ms]	zrychlení	účinnost
1	73671	1	1
2	69922	1,053617	0,526808
4	66996	1,099633	0,274908
8	66996	1,099633	0,137454
16	65123	1,131259	0,070704
26	64104	1,149242	0,044202

Řídké nerovnice 150x6			
p	čas [ms]	zrychlení	účinnost
1	33943	1	1
2	34294	0,989765	0,494882
4	30344	1,118607	0,279652
8	31034	1,093736	0,136717
16	39190	0,866114	0,054132
24	29450	1,152564	0,048023

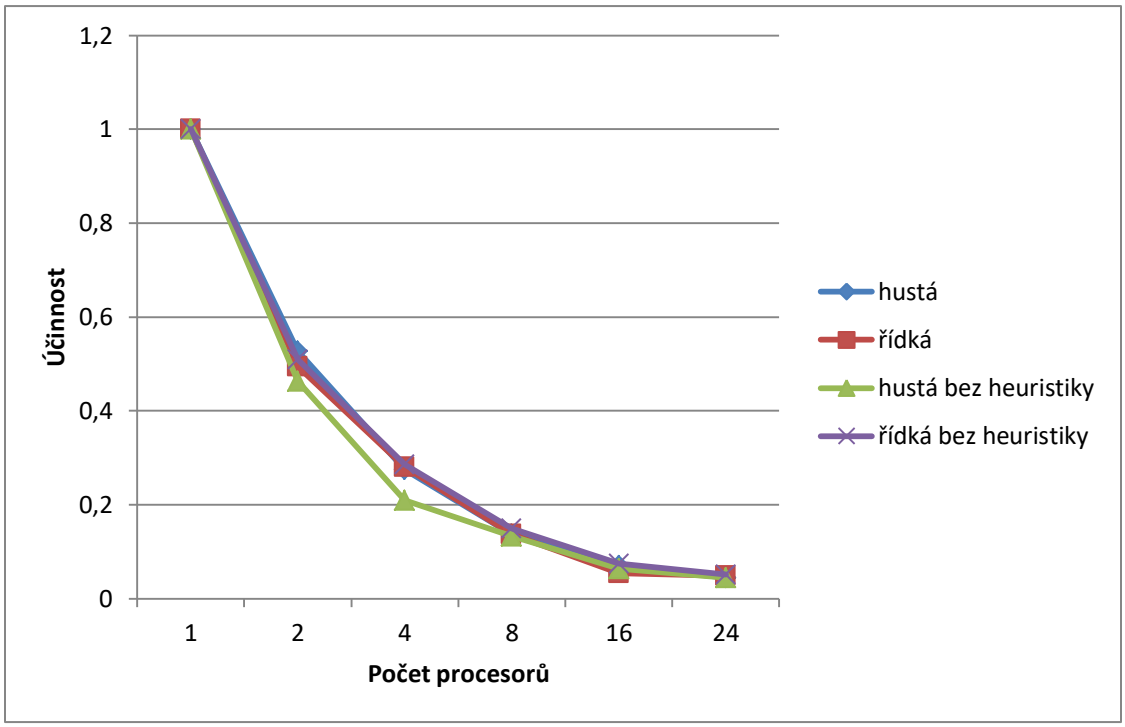
Husté nerovnice 16x5 bez heuristiky			
p	čas [ms]	zrychlení	účinnost
1	176159	1	1
2	190324	0,925574	0,462787
4	210482	0,836931	0,209233
8	164905	1,068245	0,133531
16	171739	1,025737	0,064109
24	167543	1,051426	0,043809

Řídké nerovnice 150x6 bez heuristiky			
p	čas [ms]	zrychlení	účinnost
1	214469	1	1
2	210420	1,019242	p
4	187358	1,144702	0,286175
8	179856	1,192448	0,149056
16	181631	1,180795	0,0738
24	175054	1,225159	0,051048

Tabulka 1 Výsledky



Graf 1 Paralelní zrychlení



Graf 2 Paralelní účinnost

Z naměřených dat je patrné, že heuristika má velký vliv na rychlost algoritmu, kdy dostáváme průměrně třikrát rychlejší běh programu než bez heuristiky. Je tu ale korelace s použitou pamětí RAM při výpočtu, která se s použitím heuristiky také sníží. Takže zrychlení by bylo možné zdůvodnit právě tímto menším použitím paměti.

Paralelizace víceméně nemá žádný vliv na FME, dosahuje pouze mírného zrychlení. Tomu by odpovídalo i samotné využití procesoru, kde při použití jednoho procesoru je využit z 98% a s přidáním dalších procesorů se tento výstup vydělí počtem procesorů (např. pro 2 procesory je využití 49%).

Samotné výsledky se při opakovaném výpočtu mění, a to v řádu několika sekund. Může tak nastat situace, kdy nejpomalejší čas dostaneme při využití 16 jader, ale nejlepší při použití 26, jako nastalo u měření dat řídkých nerovnic (viz Graf 1). Toto by indikovalo, že záleží na kompilátoru, jak v danou chvíli optimalizuje kód.

Přidáváním dalších procesorů se rychlost algoritmu nezvyšuje, je víceméně stále stejná, proto účinnost (viz rov. 6) paralelizace lineárně klesá.

5. Závěr

V úvodu bakalářské práce jsou definovány základní pojmy použité v průběhu této práce. Poté byla FME matematicky definována a pomocí řešeného příkladu bylo demonstrováno její základní použití. V této části byl také podrobně zadefinován plánovaný cíl této práce, který byl porovnán s předešlými pracemi týkající se FME (byli zde vyzdviženy jejich rozdíly). Současně zde bylo popsáno reálné použití algoritmu FME.

V druhé kapitole Analýza a design jsou analyzovány jednotlivé části algoritmu a je zde navrženo, jak tyto části můžeme řešit. Nejprve je popsána analytická složitost FME. Následuje ji popis postupů k řešení sekvenčního algoritmu. Analýza paralelizace sekvenčního algoritmu obdobně nabízí postupy, které můžeme použít při řešení. Také jsou tu představeny jednotlivé možnosti reprezentace dat.

Třetí kapitola Realizace je popis realizace FME. Z druhé kapitoly jsou zde již vybrané postupy a přímo jejich realizace pomocí patřičných prostředků poskytovaných OpenMP a C++14. Jednotlivé části této kapitoly zde tedy navazují na sekce kapitoly druhé.

Ve čtvrté kapitole Měření jsou popsány části týkající se měření a testování dat, je zde také řečeno, jak jsou daná data reprezentována a jak jsou ohodnocena. Kapitola také obsahuje samotné zhodnocení těchto dat a patřičné závěry vycházející z této práce.

Co se rozšíření této práce týče, bylo toto téma již hodně popsáno, ale dosud zde nebyla práce, která by spojila všechny zmíněné práce do jedné a udělala tak jeden velký program. Z práce také vyplívá důležitost práce s pamětí, potenciálně by se dal vytvořit určitý chytrý alokátor paměti, který by efektivně alokoval paměť pro nerovnice v kroku eliminace.

Reference

[1] MAKARA, T. *Vplyv druhov aritmetiky na Korovin/Tsiskaridz/Voronkovov algoritmus pre riešenie lineárnych nerovníc*. Praha, 2014. Bakalářská práce. Czech Technical University, Faculty of Information Technology.

[2] BENCOMO, Mario, Luis GUTIERREZ a Martine CEBERIO. *Modified Fourier-Motzkin Elimination Algorithm for Reducing Systems of Linear Inequalities with Unconstrained Parameters* [online]. 2011 [cit. 2019-01-09]. Dostupné z: https://digitalcommons.utep.edu/cs_techrep/593/

[3] DANTZIG, George a B EAVES. *Fourier-Motzkin elimination and its dual*. Journal of Combinatorial Theory, Series A, Volume 14, Issue 3, 1973. Dostupné také z: http://www.sciencedirect.com/science/article/pii/0097316573900046

[4] HERCEG, Martin, Michal KVASNICA, Colin JONES a Manfred MORARI. Multi-Parametric Toolbox 3.0. In: *2013 European Control Conference (ECC)* [online]. IEEE, 2013, s. 502-510 [cit. 2019-01-10]. DOI: 10.23919/ECC.2013.6669862. ISBN 978-3-033-03962-9. Dostupné z: https://ieeexplore.ieee.org/document/6669862/

[5] KOLÁŘ, Petr. *Operační systémy* [online]. Liberec, 2005 [cit. 2019-05-09]. Dostupné z: http://www.nti.tul.cz/~kolar/os/

[6] #pragma omp atomic. *IBM* [online]. b.r. [cit. 2019-05-09]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSGH2K_13.1.3/com.ibm.xlc1313.aix.doc/compiler_ref/prag_omp_atomic.html

[7] Počítačový cluster. *Wikipedia* [online]. b.r. [cit. 2019-05-14]. Dostupné z: https://cs.wikipedia.org/wiki/Po%C4%8D%C3%ADta%C4%8Dov%C3%BD_cluster

[8] KOPP, Martin. *Efektivní řešič velkých soustav lineárních nerovnic*. Plzeň, 2010. Bakalářská práce. České vysoké učení technické v Praze, fakulta elektrotechnická. Vedoucí práce Ing. Ivan Šimeček, Ph.D.

[9] FRITSCH, Richard. *Parallel solver of large systems of linear inequalities*. Praha, 2014. Diplomová práce. Czech Technical University, Faculty of Information Technology.

[10] AJSPUR, Mai a Rune JENSEN. *Using Fourier-Motzkin-Elimination to Derive Capacity Models of Container Vessels*. Denmark, 2017. Report. IT University of Copenhagen.

[11] SURIANA, Patricia. *Fourier-motzkin with non-linear symbolic constant coefficients*. Massachusetts, 2016. Master thesis. Massachusetts Institute of Technology.

[12] AHO, Alfred, Monika LAM, Ravi SETHI a Jeffrey ULLMAN. *Compilers*. 2nd ed. Boston: Pearson/Addison Wesley, 2007. ISBN 978-0321486813.

[13] Parallel Fourier-Motzkin elimination. In: BOUGÉ, L. *Euro-Par '96 parallel processing: second International Euro-Par Conference, Lyon, France, August 26-29, 1996 : proceedings* [online]. New York: Springer, 1996 [cit. 2019-01-11]. ISBN 978-3-540-61627-6.

[14] BOUGÉ, L. *Euro-Par '96 parallel processing: second International Euro-Par Conference, Lyon, France, August 26-29, 1996 : proceedings* [online]. New York: Springer, 1996 [cit. 2019-01-11]. ISBN 978-3-540-61627-6.

[15] DAGUM, L. a R. MENON. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*. b.r., **5**(1), 46-55. DOI: 10.1109/99.660313. ISSN 10709924. Dostupné také z: http://ieeexplore.ieee.org/document/660313/

[16] #pragma omp parallel. *IBM* [online]. b.r. [cit. 2019-05-09]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSGH2K_13.1.2/com.ibm.xlc1312.aix.doc/compiler_ref/prag_omp_parallel.html

[17] #pragma omp for. *IBM* [online]. b.r. [cit. 2019-05-09]. Dostupné z: https://www.ibm.com/support/knowledgecenter/en/SSGH2K_13.1.2/com.ibm.xlc131.aix.doc/compiler_ref/prag_omp_for.html

[18] Shared and private variables in a parallel environment. *IBM* [online]. b.r. [cit. 2019-05-09]. Dostupné z: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.cbcpx01/cuppvars.htm

[19] Std::move. *Cppreference* [online]. 2018 [cit. 2019-05-09]. Dostupné z: https://en.cppreference.com/w/cpp/utility/move

[20] Std::valarray. *Cppreference* [online]. b.r. [cit. 2019-05-09]. Dostupné z: https://en.cppreference.com/w/cpp/numeric/valarray

[21] STROUSTRUP, Bjarne. *The C programming language*. Fourth edition. Upper Saddle River, NJ: Addison-Wesley, 2013. ISBN 03-215-6384-0.

[22] Std::vector. *Cppreference* [online]. b.r. [cit. 2019-05-09]. Dostupné z: https://cs.cppreference.com/w/cpp/container/vector

[23] Std::shared_ptr. *Cppreference* [online]. 2018 [cit. 2019-05-09]. Dostupné z: https://en.cppreference.com/w/cpp/memory/shared_ptr

[24] *The R Project for Statistical Computing* [online]. b.r. [cit. 2019-05-11]. Dostupné z: https://www.r-project.org/

[25] Editrules. *Rdocumentation* [online]. b.r. [cit. 2019-05-11]. Dostupné z: https://www.rdocumentation.org/packages/editrules/versions/2.9.3

[26] TVRDÍK, Pavel. *Parallel algorithms and computing*. Praha: Vydavatelství ČVUT, 2003. ISBN 80-010-2824-0.

[27] *PBS Professional* [online]. b.r. [cit. 2019-05-14].

[28] CMake. *Cmake* [online]. b.r. [cit. 2019-05-15]. Dostupné z: https://cmake.org/

Přílohy

A1 Seznam použitých zkratek

API Application Programing Interface

FIT Fakulta informačních technologii ČVUT

FME Fourier-Motzkin elimination

SN soustava nerovnic

A2 Použití programu

Pro použití programu je nutné naimportovat hlavičkové soubory $Solver.hpp$ a $utils.hpp$. $Solver.hpp$ reprezentuje kód samotného paralelního řešiče FME. $Utils.hpp$ obsahuje pomocné funkce na práci s daty a změření času běhu programu. Pro kompilaci programu se zavolejte $cmake$ na soubor $CMakeLists.txt$, vytvoří se tak makefile a pomocí něj program zkompilujte. V souboru $main.cpp$ pak můžete definovat vlastní užití programu.

Možné použití programu by mohlo vypadat takto:

```
1.  //include řešič a pomocné funkce
2.  #include "Solver.hpp"
3.  #include "utils/utils.hpp"
4.
5.  int main() {
6.      //vytvoříme řešič
7.      Solver solver = Solver();
8.      //vygenerujeme data
9.      genData(20,3,20,"newdata");
10.     //přečteme data z vygenerovaného oběktu
11.     auto tmp = readData("newdata");
12.     //změříme čas běhu řešiče
13.     timeAlg(solver,tmp,"pickfirst",true);
14.     //uvolníme paměť
15.     solver.free();
16.     return 0;
17. }
```

Kód 4 Ukázka použití programu