Semestrální práce z KIV/PPR

Paralelní programování

Standardní zadání

25. Ledna 2016
David Košek, A14N0132P
kosek@students.zcu.cz

Obsah

1. Zadání

Úloha spadá do oblasti řešení problému, zda lze funkci transportérů glukózy dostatečně přesně nahradit modelem, který explicitně nepočítá s akcemi spouštěnými detekcí inzulínu v krvi. Bez zacházení do přílišných detailů, mějme následující funkce:

$$\varphi(t) = t + \Delta t + k * \frac{i(t) - i(t-h)}{h} \quad (1)$$

$$\alpha = cg \quad (2)$$

$$\beta = p - cg * i(t) \quad (3)$$

$$\gamma = c - i(\varphi(t)) \quad (4)$$

$$D = \beta^2 - 4 * \alpha * \gamma \quad (5)$$

$$b(t) = \frac{-\beta + \sqrt{D}}{2 * \alpha} \quad (6)$$

Rovnice (6) v sobě zahrnuje rovnice (1) až (5). Ve výsledku se tak jedná o funkci dvou proměnných - b(t) a i(t) - a sady parametrů. Obě proměnné byly získány měřením. Vaším úkolem je najít takovou sadu parametrů uvedených rovnic, pro které bude rozdíl mezi pravou a levou stranou rovnice (6) minimální. Rozdíl mezi oběma stranami vypočítejte pro všechny t, ve kterých je k dispozici hodnota b(t), a to podle následující metriky - čím menší číslo vyjde podle zvolené metriky, tím lepší jsou testované parametry:

- součet průměrné relativní chyby a standardní odchylky relativních chyb
- relativní chyba se vypočítá jako absolutní rozdíl mezi naměřenou a vypočítanou hodnotou, který se vydělí naměřenou hodnotou - tj. b(t)

Hodnoty b(t) a i(t) nemusely být naměřeny ve stejných časech t, a protože b(t) bylo měřené s daleko menší frekvencí než i(t), je třeba hodnoty i(t) nejprve buď aproximovat, nebo interpolovat - sami si zvolte vhodnou metodu.

1.1 Verze úlohy

Zpracujte úlohu alespoň ve dvou verzích ze tří možných:

1. Paralelní program pro systém se sdílenou pamětí
2. x86 CPU + OpenCL GPGPU
3. Paralelní program pro systém s distribuovanou pamětí

1.2 Meze parametrů

Čas je ve zdrojovém souboru zadán jako kombinace data a času během dne. Datum a čas je nutné konvertovat do čísla s plovoucí desetinnou čárkou, kdy jeden den == 1.0. Jinak nebudou parametry vycházet v následujících mezích!

Parametr	Meze
p	0 až 2
cg	-0.5 až 0
c	-10 až 10
dt, h	0 - až 40 minut, tj. 0,0277777
k	-1 až 1, číslo by mělo být blízko nuly

Efektivní meze výpočtu budou zaneseny v konfiguračním souboru.

Doporučené řešení

Metod nalezení parametrů může existovat více, nicméně doporučeným řešením je **Nealder-Mead algoritmus**.

1.3 Data

Naměřené hodnoty jsou uloženy ve formátu SQLite verze 3. Konkrétně jsou uloženy v tabulce measuredvalue. Funkci b(t) odpovídá sloupec blood, který vyjadřuje koncentraci glukózy v krvi v [mmol/l]. Funkci i(t) odpovídá sloupec ist, který vyjadřuje koncetraci v intersticiální tekutině v [mmol/l]. Čas měření je zanesen ve sloupci measuredat, a je ve formátu ISO 8601. Data jsou seskupena do tzv. segmentů, viz sloupec segmentid. Naměřená data zpracovávejte vždy po celých segmentech. Jméno segmentu lze dohledat v tabulce timesegment a jméno pacienta analogicky v tabulce subject.

1.3 Další statistiky

Program také spusťte s jedním vláknem/procesem a změřte čas výpočtu sériovým kódem a čas výpočtu paralelizovaným kódem (pro všechny verze paralelizovaného kódu zvlášť). Z těchto hodnot vypočítejte následující ukazatele:

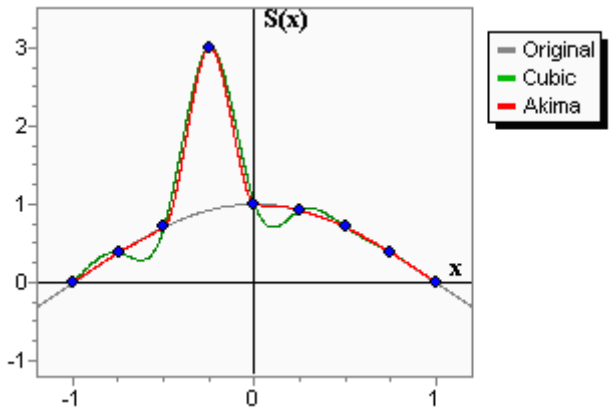
- Amdahlův zákon, f - čas sériově prováděné části kódu
- Gustafsonův zákon, α - část kódu, kterou nelze paralelizovat
- Karp-Flattova metrika, e - část sériově prováděné části kódu

2. Teoretický úvod

2.1 Interpolace

Interpolace v numerické matematice znamená nalezení přibližné hodnoty funkce v nějakém intervalu, je-li její hodnota známa jen v některých jiných bodech tohoto intervalu. Od aproximace se interpolace liší tím, že hledaná křivka přesně prochází všemi známými (změřenými) body.

Při analýze knihoven nabízející interpolování jsem váhal mezi *Cubic Spline interpolation* a *alglib*. Obě zmíněné knihovny jsou dostupné v open source verzi, tudíž hlavním výběrovým kritériem byla funkcionalita. V níž jsem se přiklonil na stranu *alglib*, protože nabízí více implementačních typů interpolace včetně zmíněné *Cubic spline* metody. Pro semestrální práci jsem vybral *Akime spline* metodu, která je přesnější než *Cubic spline* a netrpí „překmity“. Samozřejmě vybraná knihovna nabízí i další metody. Na obrázku č. 1 jsou zobrazeny obě metody. [1]



Obrázek 1: Porovnání dvou interpolačních metod s původní funkcí.

2.2 Generování náhodných čísel

Generátor pseudonáhodných čísel je program, jehož výstupem je deterministicky a efektivně určená posloupnost čísel taková, že je statisticky k nerozeznání od náhodné posloupnosti čísel. Generátor lze ovlivnit mnoha způsoby, ale nejčastější způsob je zvolit jiné pravděpodobnostní rozdělení. Pro naši práci využijeme implementaci *Mersenne twisteru mt19937* a rovnoměrné rozdělení. Tento generátor náhodných čísel má dlouhou periodu opakování čísel a prošel statistickými testy náhodnosti. Princip této metody je uveden v [2].

2.3 Nelder-Mead

„Nelder-Mead simplexová metoda se řadí do třídy algoritmů založených na přímém prohledávání stavového prostoru. Velkou výhodou algoritmu je, že nevyžaduje vyhodnocení derivací účelové funkce, jak je tomu v případě gradientních metod. Díky této vlastnosti umožňuje zpracovávat nespojité funkce a funkce, u nichž je vyčíslení derivací obtížné či neúměrně drahé.“ [3]

V každém kroku algoritmu jsou následující operace:

1. Setřídění bodů simplexu podle velikosti (metriky)
2. Reflexe nejhoršího bodu proti těžišti protější stěny
3. Expanze dalším protažením reflexe v úspěšném směru
4. Vnitřní kontrakce bodu
5. Vnější kontrakce všech bodů směrem k aktuálnímu nejlepšímu bodu

Algoritmus probíhá v závislosti na počtu požadovaných iterací. Je ale možné stanovit si zastavovací podmínky a omezit tím využití všech iteračních kroků a zrychlit výpočet. Mezi vhodné podmínky se řadí sledování simplexu a porovnání ho s požadovaným ε, pokud je simplex menší než stanované ε, předpokládá se, že se výsledné řešení příliš nezlepší a algoritmus lze ukončit. Další mezi zastavovací podmínky lze uvažovat sledování simplexu, který se po dlouho dobu (velký počet iterací) nezměnil. Například pokud máme 10 000 iterací a po dobu 5000 iterací máme stále stejný simplex, lze předpokládat, že se nám už daný simplex nezlepší a lze algoritmus ukončit.

Algoritmus včetně vývojového diagramu a určení výpočtů jednotlivých kroků je popsán v [4] (v odkazu lze také nalézt grafické znázornění expanze, reflexe a obou druhů kontrakce).

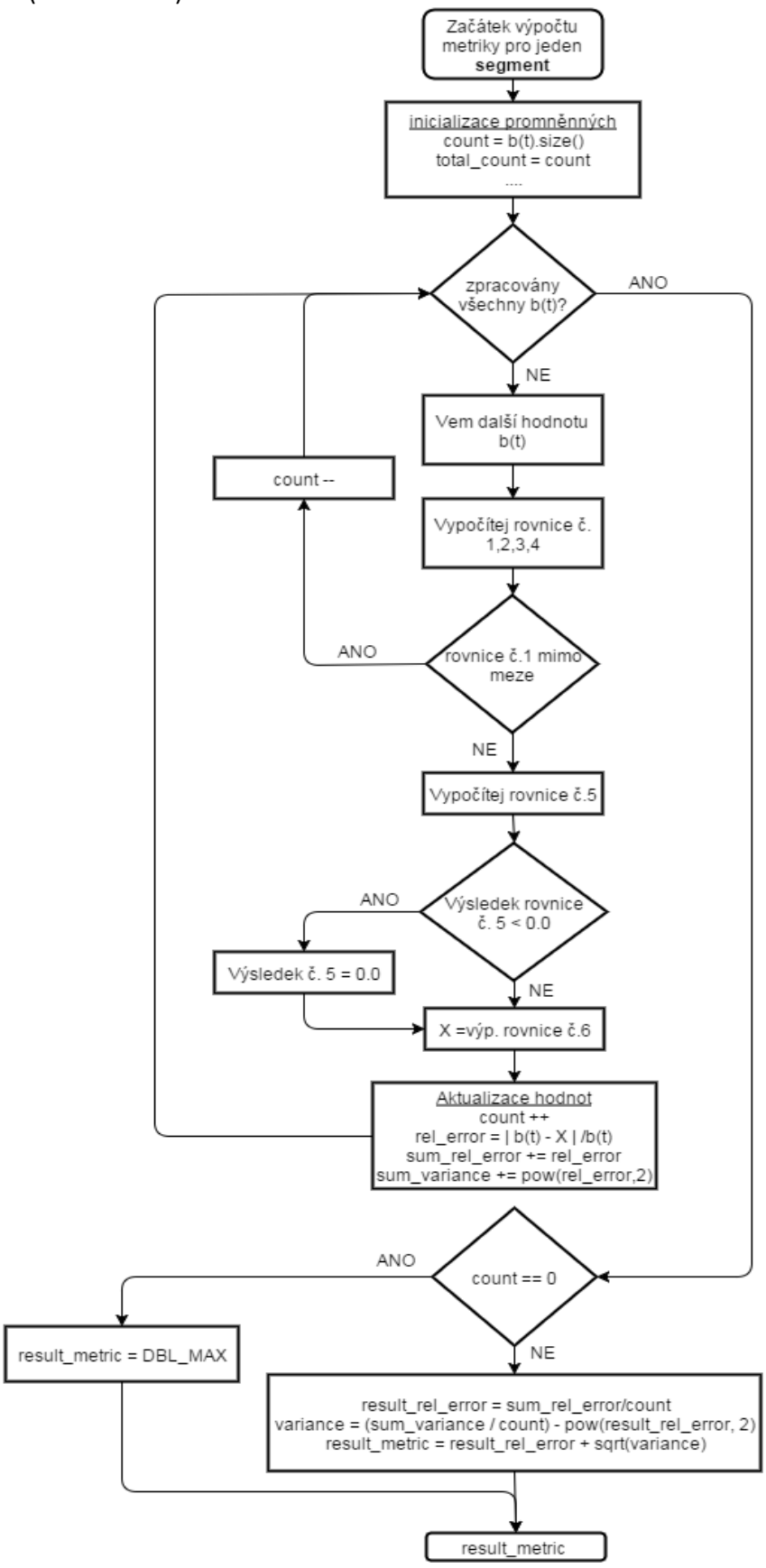
2.3.1 Určení počátečního simplexu

Pro úspěšnou implementaci metody je však ještě nutné vyřešit generaci počátečního simplexu a ukončení optimalizace. Vytvoření prvotního simplexu je celkem jednoduché. Pokud máme přibližný odhad (meze parametrů), kde bychom měli hledat řešení, volíme body simplexu v jeho blízkosti. V programu se použije smyčka, která vygeneruje požadovaný počet sad parametrů, pro které se poté vypočítají metriky. Algoritmus dále iterativně pokračuje modifikacemi simplexu popsanými v předchozí části. Ukončení optimalizace je taktéž popsán v předchozí části.

2.3.2 Výpočet metriky

Při výpočtu metriky pro množinu parametrů se pro každou hodnotu b*(t)* vypočítá pravá strana rovnice v čase *t*. Pro všechny vypočítané pravé strany, se poté určí průměrná relativní chyba a standardní odchylka relativních chyb. Pokud nastane, že by některé body musely být extrapolovány, nejsou ve výpočtu zahrnuty. Výsledná metrika je rovna součtu průměrné

relativní chyba se směrodatnou odchylkou. Celý výpočet je znázorněn na následujícím diagramu (Obrázek č. 2).

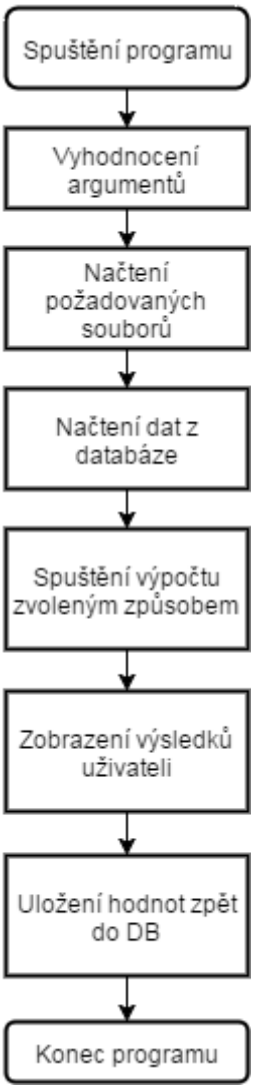


Obrázek 2: Vývojový diagram výpočtu metriky pro zadanou sadu parametrů.

3. Implementace

Pro realizaci semestrální práce jsem zvolil programovací jazyk C++. K práci s vlákny využívám knihovnu *Thread Building Blocks* [5], která usnadňuje práci s vlákny a měla by dosahovat lepšího výkonu oproti vláknům definovaným uživatelem. Pro potřebnou interpolaci využívám dříve zmíněnou knihovnu *Alglib* (viz. 2.1). Pro práci s databází bylo nutné přidat knihovnu *sqlite3*.

Semestrální práce se rozděluje na dva samostatné projekty. První z nich je pro sériovou verzi a paralelní s využitím sdílené paměti s pomocí *TBB.* Druhý projekt využívá paměť sdílenou s pomocí knihovny *MPI*. Načítání dat, generování parametrů, výpočet, komunikace s databází je implementována pro oba projekty stejně. Během běhu programu jsou vypisovány informativní hlášky pro uživatele, aby viděl, že program skutečně pracuje. Po skončení výpočtu jsou výsledné statistiky zobrazeny uživateli. Základní konstrukce aplikace je znázorněna na následujícím obrázku č. 3.



Obrázek 3: Hrubá struktura implementace programu.

3.1 Načítání dat

V programu je za potřebí, načítat tři soubory (na pořadí nezáleží). Jedná se o soubor s požadovanými limity parametrů, dále konfigurační soubor s možným nastavením důležitých parametrů pro výpočet a algoritmus a posledním požadovaným souborem je databáze.

3.1.1 Databáze

Pro načtení a správné zacházení s daty jsem použil zmíněnou *sqlite3* knihovnu. Jak pracovat s touto knihovnou jsem se dočetl na internetu [6]. Hodnoty načítám na začátku běhu programu a ukládám si je do vektorů pro pozdější využití. Z databáze potřebujeme načíst následující:

- Koncentraci glukózy v krvi
- Koncentraci v intersticiální tekutině
- Časy měření

Konkrétně data jsou uloženy v tabulce *measuredvalue*. Funkci b(t) odpovídá sloupec *blood*, který vyjadřuje koncentraci glukózy v krvi v [mmol/l]. Funkci i(t) odpovídá sloupec *ist*, který vyjadřuje koncentraci v intersticiální tekutině v [mmol/l]. Čas měření je zanesen ve sloupci *measuredat* (viz. Zadání úlohy).

Naměřená data zpracovávám vždy po celých segmentech. Jelikož některé segmenty mohou chybět, tak si index vektoru, kam ukládám danou hodnotu, inkrementuji sám. S tím přichází další práce a to ta, že si musím uložit původní hodnotu segmentu, abych jí pod ní později mohl uložit zpět.

3.1.2 Limity parametrů

Při načítání hodnot ze souboru jsem zvolil jednoduchý a striktní postup, který je ale závislý na struktuře daného souboru. Hodnoty opět načítám na začátku programu ze zadaného souboru. Meze v souboru musí být ve formátu *název=hodnota* a musí být zadány střídavě minimum a maximum v zadaném pořadí. V ideálním případě, tak jako je vzorový soubor *bounds.ini*.

3.1.3 Konfigurace

V této části jsem vylepšil svůj dosavadní algoritmus pro načítání hodnot ze souboru. V inicializačním souboru *parameters.ini* se nachází několik parametrů povinných pro běh programu (Tabulka 1). Tyto parametry lze měnit – v případě, že chybí některý z parametrů, vypíše se varování a výpočet neproběhne. Hodnoty parametrů se načítají podle definovaných názvů. Tyto se tedy nesmí měnit, ale nezáleží na jejich pořadí.

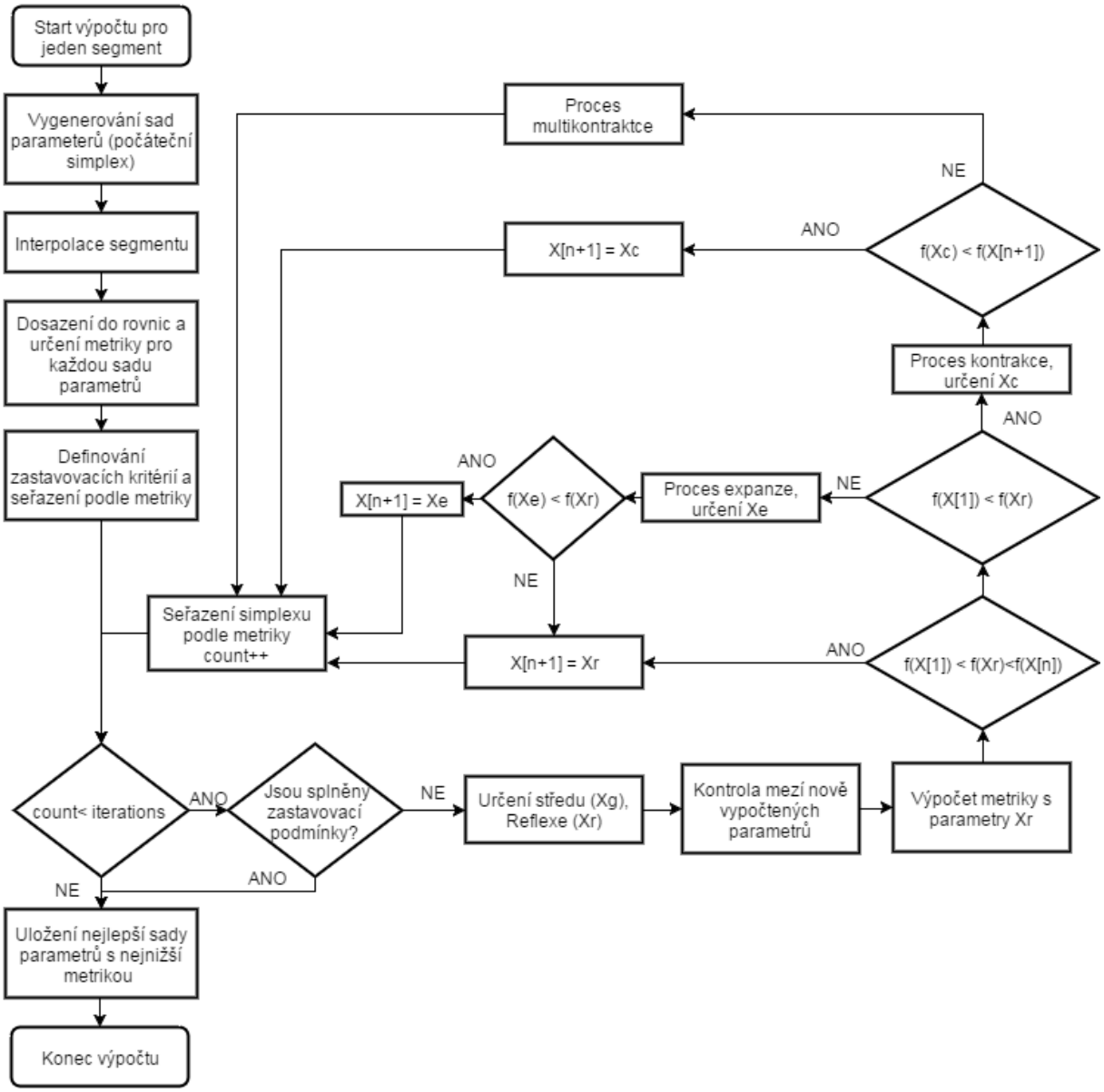
Název	Význam
iterations	Počet iterací algoritmu
min_set_of_params	Minimální počet vygenerovaných sad parametrů pro každý segment
precision	Požadovaná přesnost pro zastavovací podmínky algoritmu
A	Konstanta pro reflexi
B	Konstanta pro expanzi
G	Konstanta pro kontrakci
H	Konstanta pro multi-kontrakci

Tabulka 1: Přehled konfiguračních parametrů a jejich význam.

3.2 Sériový výpočet a TBB

Tyto dvě metody se od sebe liší pouze tím, že sériový výpočet používá pouze jedno vlákno. Pomocí *task scheduleru* je možné inicializovat předem určitý počet vláken. Tedy pro sériový výpočet je výpočtu přiřazeno jedno vlákno, pokud *scheduleru* neurčíme počet vláken, zjistí si maximální možný počet a ten využije pro výpočet. Jelikož výkonný kód je stejný pro jedno i více vláken, popíši pouze paralelní zpracování.

Před začátkem výpočtu si nadefinuji vektor výsledných sad parametrů pro jednotlivé segmenty a *spin lock*, který využívám pro vzájemné vyloučení při ukládání sady parametrů do výsledného vektoru. Samotný výpočet je vložen do těla metody *paraller_for* z knihovny *TBB.* Tento cyklus nám proběhne přesně tolik krát, kolik máme segmentů (pro každý segment jednou). Nejprve se pro daný segment vygenerují sady parametrů v požadovaných mezích. Poté potřebujeme provést interpolaci daného segmentu pro další krok. Tím je spočítání metrik všech vygenerovaných sad parametrů. Jak výpočet metriky probíhá je znázorněno na obrázku č. 2. Poté nám nezbývá než nad zadanou sadou parametrů aplikovat algoritmus Nelder-Meada. Předtím provedeme pouze inicializaci zastavovacích podmínek. Jako zastavovací podmínky využívám obě zmíněné v analýze Nelder-Meada algoritmu (viz. 2.3). Pro lepší představu o výpočtu a aplikování algoritmu je znázorněn na následujícím vývojovým diagramu (obr. 4).



Obrázek 4: Postup výpočtu včetně optimalizačního algoritmu Nelder-Mead.

Poznámka k vývojovému diagramu – po provedení, kteréhokoliv procesu (reflexe, kontrakce, atd.) dochází ke kontrole nově vypočtených parametrů podle požadovaných mezí a k výpočtu nové metriky, která je následně porovnána v rozhodovacím bloku. Tento postup jsem zobrazil pouze v prvním procesu reflexe ze dvou důvodů. První je místo v diagramu a druhý je ten, že reflexe nám proběhne vždy.

3.3 MPI

V tomto typu úlohy jsou procesy rozděleny na dva typy (při implementaci využíváme návrhový vzor *farmer-worker,* ukázka implementace na [7]). První z nich je řídící proces (*farmer*), ten je pouze jeden. Druhý typ procesu je výpočetní proces (*worker*), těch je *N-1*, kde *N* je počet přidělených procesů. Každý z procesů si nejprve načte data, nad kterými bude nadále pracovat (všichni mají stejná data), žádné načtené informace si procesy neposílají. Po načtení dat proces inicializuje, že bude využívat MPI pomocí funkce *MPI_Init*. Následně získá *rank* svého procesu pomocí funkce *MPI_Comm_rank* a počet procesů pomocí *MPI_Comm_size.* Poté se proces dělí na *farmer* nebo *worker*. Mezi procesy probíhá komunikace pomocí funkcí *MPI_Send* a *MPI_Recv*. Důležité je nastavení tagů, pod kterým tagem zprávu odesílám, pod ním také zprávu na druhé straně přijímám. Zde jsem nevytvářel žádné magické konstanty a vystačil jsem si s hodnotami 1 a 2 (u *farmera* pod tagem 1 odesílám data *workerovi* a pod tagem 2 hodnoty přijímám. U *workera* je to přesně naopak). Po ukončení procesu je nutné ukončit MPI pomocí funkce *MPI_Finalize*. Více o nastavení jednotlivých argumentů a prací s MPI knihovnou zle nalézt v [8].

3.3.1 Výpočetní proces (worker)

Výpočetní proces běží ve smyčce a čeká na přidělení práce. Po přijmutí nové zprávy se zpráva vyhodnotí, jsou pouze dvě možnosti. Buď jsem přijal požadavek na spočítání daného segmentu, nebo zprávu o tom, že jsou všechny segmenty již spočtené a proces se může ukončit. V podstatě přijímám index do pole, kvůli již zmíněnému problému se segmenty (viz. 3.1.1). Při přijmutí indexu k výpočtu spočítám parametry s metrikou pro daný segment a odešlu výsledky zpět. Zpět posílám pole obsahující jednotlivé parametry a k ním spočtenou metriku.

3.3.2 Řídící proces (farmer)

Řídící proces obecně rozděluje práci a zpracovává výsledky, které nakonec uloží do databáze. Nejprve proces pošle všem výpočetním procesům počáteční práci a poté čeká ve smyčce na jejich odpovědi. Výpočetní procesy začnou odpovídat a proces si výsledek uloží a pošle nový segment, pro který nemá výsledek. Po odeslání zpráv pro všechny segmenty si spočítáme rozdíl celkového počtu segmentů a počtu přijatých segmentů, abychom věděli, kolik ještě musíme přijmout výsledků. Nakonec všem výpočetním procesům pošleme flag o ukončení, uložíme data a proces se ukončí.

4. Statistiky

Popis HW, na kterém probíhalo testování:

Typ zařízení	*Notebook Lenovo ThinkPad T520*
Procesor	*Intel® Core™ i7-2670QM CPU @2.20GHz*
Operační paměť	*16,00 GB (Použitelné: 15,9GB)*
Disk	*Samsung SSD 850 EVO 250G*
OS	*MS Windows 7 Professional 64-bit Server Pack 1*

Procesor má 4 procesy (8 procesorových vláken). Veškeré měření probíhá v release verzi. Časy jsou měřeny vždy desetkrát a poté zprůměrovány. Přehled průměrných časů jsou zobrazeny v tabulce 2. Všechny naměřené hodnoty jsou v příloze A a B. Pro měření byly nastaveny zastavovací podmínky – přesnost měření na 0.00005 a počet iterací na 10 000 (tj. 5000 iterací beze změny metriky). Z naměřených hodnot vypočítáme další statistiky. Nejprve si dané statistiky ukážeme, jak se počítají a poté je aplikujeme na naše data, protože jednou to musíme udělat pro sdílenou paměť a jednou pro distribuovanou.

Způsob výpočtu	Čas [s]	Poznámka
Sériový	15,195	Spuštěno pro 1 proces
TBB	5,345	TBB bylo přiřazeno max. procesů (8)
MPI	56,72	Spuštěno na clusteru hydra.fav.zcu.cz s 2 procesy (1 Farmer, 1 Worker)
MPI	30,157	Spuštěno na clusteru hydra.fav.zcu.cz s 5 procesy (1 Farmer, 4 Workeři)

*Pozn.: Výpočet pro distribuovanou paměť mohl být ovlivněn vytížením clusteru hydra.fav.zcu.cz.

Tabulka 2: Přehled průměrných časů jednotlivých způsobů výpočtu.

4.1 Amdahlův zákon

„Amdahlův zákon je pravidlo používané v informatice k vyjádření maximálního předpokládaného zlepšení systému poté, co je vylepšena pouze některá z jeho částí.“ [9]

Uvádí se v několika ekvivalentních formách. Uvedeme si často prezentovaný tvar Amdahlova zákona:

$$S = \frac{1}{f + \frac{1-f}{N}}$$

Ze zákona máme určit f, tedy poměr času stráveného výpočtem sériové části ku celkovému.

$$f = \frac{t_s}{t_s + t_p}$$

Ukázka výpočtu pro sériový způsob výpočtu na sdílené paměti.

$t_p = 14{,}971\ s$ - čas paralelní části prováděné sériově
$t_s = 0{,}224\ s$ - čas sériové části

$$f = \frac{t_s}{t_s + t_p} = \frac{0{,}224}{15{,}195} = 0{,}0147416$$

4.2 Gustafsonův zákon

Gustafsonův zákon určuje zrychlení $S(p)$ jako rozdíl mezi počtem procesorů P a počtem procesorů $P-1$. Výchozí vzorce:

$$S(p) = p - \alpha * (p - 1)$$

$$\alpha = \frac{p - S}{p - 1} = \frac{p - \frac{t_{serial}}{t_{paralel}}}{p - 1}$$

Ukázka výpočtu pro sériový způsob výpočtu na sdílené paměti.

$$S = \frac{t_{serial}}{t_{paralel}} = \frac{15{,}195}{5{,}214} = 2{,}914269$$

$$\alpha = \frac{8 - 2{,}914269}{8 - 1} = 0{,}726533$$

4.3 Karp-Flattova metrika

Z Karp-Flattovy metriky můžeme část sériově prováděné části kódu e určit ze základního tvaru:

$$e = \frac{\frac{1}{\Psi} - \frac{1}{p}}{1 - \frac{1}{p}}$$

Ψ nám udává zrychlení na p – procesech

$$\Psi = \frac{T(1)}{T(p)} = \frac{T_s + T_p}{T_s + \frac{T_p}{p}}$$

Ukázka výpočtu pro sériový způsob výpočtu na sdílené paměti.

$t_p = 14{,}971\ s$ - čas paralelní části
$t_s = 0{,}224\ s$ - čas sériové části

$$\Psi = \frac{T(1)}{T(p)} = \frac{T_s + T_p}{T_s + \frac{T_p}{p}} = \frac{15{,}195}{0{,}224 + \frac{14{,}971}{8}} = 7{,}25168$$

$$e = \frac{\frac{1}{\Psi} - \frac{1}{p}}{1 - \frac{1}{p}} = \frac{\frac{1}{7{,}25168} - \frac{1}{8}}{1 - \frac{1}{8}} = 0{,}0147418$$

4.4 Výsledky

	Amdahlův zákon	**Gustafsonův zákon**	**Karp-Flattova metrika**
TBB	0,014741	0,726533	0,014741
MPI	0,010329	0,787433	0,010329

Tabulka 3: Přehled výsledků statistik po dosazení do uvedených vzorců.

5. Uživatelský manuál

Program je napsaný jako konzolová aplikace v jazyce C++. Projekt je dodán ve dvou verzích. První verze je pro sdílenou paměť (sériový a paralelní výpočet) jako projekt Visual Studia 2012. Druhá verze slouží pro výpočet s distribuovanou pamětí (MPI).

5.1 Kompilace

Verze TBB je dodána a byla vyvíjena v programu *Microsoft Visual Studio 2012*. Proto pro kompilaci této verze stačí spustit projekt (koncovka *vcxproj*). Velmi důležité je, před přeložením projektu nastavit správně cesty ke knihovnám TBB. Dodaná verze obsahuje statické cesty na stroji, kde byla verze implementována. Tato verze dále využívá knihovny *sqlite3* a *alglib*, kterou jsou umístěné v adresáři *Lib.* Pokud dojde k přemístění toho souboru, je nutné pozměnit cesty v hlavičkových souborech, kde jsou použity. Projekt je možné zkompilovat pro *Debug* nebo *Release* konfiguraci (poté je však nutné správné nastavení cest, jak pro debug, tak release).

Verze MPI byla vyvíjená pod operačním systémem *Linux – Debian*. Pro správné zkompilování je nutné mít nainstalovanou knihovnu *OpenMPI* (či jinou knihovnu implementující MPI) a knihovnu pro práci s databází *sqlite3*. Jedná se o následující příkazy k instalaci:

- apt-get install sqlite3
- apt-get install openmpi-bin openmpi-doc libopenmpi-dev

V ostatních distribucích Linuxu se mohou balíčky jmenovat jinak.

Pokud jsou splněny potřebné závislosti, lze program přeložit pomocí připraveného makefilu (příkazem *make*). Připravený makefile používá pro kompilaci příkaz mpic++, přilinkuje potřebnou *sqlite3* knihovnu a přilinkuje i statickou knihovnu *Alglib.* Pokud by s knihovnou *Alglib* nastal problém, má připravený svůj makefile pro kompilaci.

5.2 Spuštění TBB (a sériové) verze

Program se spouští z příkazové řádky následujícím způsobem:

A14N0132P.exe [parametry]

Parametry:

- -d [název databáze ve formátu sqlite3]
- -b [název souboru s mezemi parametrů]
- -c [název konfiguračního souboru]
- -e [zvolený výpočet tbb nebo serial]
- -h

Všechny z parametrů jsou povinné, krom posledního pro vypsání nápovědy, na pořadí parametrů nezáleží. Konečný příkaz může vypadat například takto:

A14N0132P.exe -d direcnet.sqlite -b bounds.ini -c parameters.ini -e tbb

5.3 Spuštění MPI verze

Program na OS Linux lze spustit příkazem:

mpirun -np [počet procesů] A14N0132P.exe [parametry]

Parametry jsou stejné jako v předchozím případě, pouze bez možnosti nastavení výpočtu. Zde je předem určený výpočet pro distribuovanou paměť s pomocí návrhového vzoru *farmer-worker.* Konečný příkaz může vypadat takto:

mpirun -np 4 A14N0132P.exe -d direcnet.sqlite -b bounds.ini -c parameters.ini

6. Závěr

Cílem semestrální práce bylo realizovat program, který hledá parametry rovnice popisujících model transportérů glukózy na základě naměřených dat uložených v databázi sqlite3. Práce je vytvořena v programovacím jazyce C++. K interpolaci segmentů využívám knihovnu Alglib. K optimalizaci dat využívám doporučený algoritmus Nelder-Mead.

Z počátku jsem vytvořil pouze sériovou verzi. Zde jsem projekt rozdělil na dva. Pro sdílenou a distribuovanou paměť. Verzi se sdílenou pamětí jsem následně pomocí knihovny Intel TBB zparalelizoval. Verzi pro distribuovanou paměť jsem začal vyvíjet pod Linuxem, protože mě stejně čekala spustit na clusteru hydra.fav.zcu.cz. Navíc jsem narazil na problémy s instalací knihovny MPI pod OS Windows 7.

Vypočítané hodnoty pro jednotlivé segmenty se ukládají do databáze a na konci běhu programu jsou vypsány uživateli do konzole se základním výpisem parametrů. Výsledky metriky ovlivňují náhodně vygenerované parametry a také počet sad parametrů a počet iterací (čím více sad a iterací, tím je výpočet přesnější, ale také zabere více času).

Na základě naměřených hodnot vyplývá, že nejrychlejší výpočet probíhá pomocí paralelního výpočtu na sdílené paměti. Naopak nejpomalejší běh byl, zaznamenám pomocí distribuované paměti pro sériovou verzi (1 *farmer*, 1 *worker*). Hodnoty měřené pro distribuovanou paměť na clusteru hydra, mohly být ovlivněny vytížením tohoto serveru. Po naměření hodnot byly vypočteny požadované statistiky.

Program splňuje veškeré body zadání. Zdrojový kód je komentovaný a dobře čitelný. Pomocí *valgrindu* jsem odstranil *memory leaky* (u TBB verze všechny, u MPI zůstávají knihovní *memory leaky*). Možných vylepšení je celá řada, zejména v implementaci výpočtu, nyní to je jedna velká funkce, tak jí patřičně rozdělit do více funkcí podle jednotlivého využití. V neposlední řadě myslet více na to, že uživatel je pouze člověk a více ošetřit vstupní hodnoty.

Práce byla pro mne velmi náročná hned z několika důvodů. Do kontaktu s C++ jsem se na škole nedostal. Proto je dost možné, že můj kód nebude ideálně optimalizovaný. Na druhou stranu jsem získal nové zkušenosti a poznal nové knihovny pro paralelní práci.

LITERATURA

[1] ALGLIB Project, [online] [cit. 28. 1. 2016] Dostupné z: http://www.alglib.net/

[2] bc. Pavel Novotný, *Knihovna pro generování pseudonáhodných čísel*, [online] [cit. 29. 1. 2016] Dostupné z: http://programator.pro/pages/pseudorandom/dokumentace.html#x1-10001

[3] Pavel Koška, *Hybrid Nelder-Mead a rojové optimalizace*, ČVUT 2013, Diplomová práce, Fakulta elektrotechnická, Katedra elektromagnetického pole.

[4] JIA, B. *Simplex Optimization Algorithm and Implemetation in C++ Programming* [online] [cit. 28. 1. 2016] Dostupné z: http://www.codeguru.com/cpp/article.php/c17505/Simplex-Optimization-Algorithm-and-Implemetation-in-C-Programming.htm

[5] TBB, *Threading Building Blocks* [online]. [cit. 29. 1. 2016] Dostupné z: https://www.threadingbuildingblocks.org/

[6] Tutorialspoint, *SQLITE C/C++ TUTORIAL* [online] [cit. 30. 1. 2016] Dostupné z: http://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm

[7] The university of Edinburgh, *Parallel programming languages and systems* [online] [cit. 30. 1. 2016] Dostupné z: http://www.inf.ed.ac.uk/teaching/courses/ppls/farm.c

[8] MPI TUTORIAL, *MPI Hello World* [online] [cit. 30. 1. 2016] Dostupné z: http://mpitutorial.com/tutorials/mpi-hello-world/

[9] WIKIPEDIE, *Amdahlův zákon* [online] [cit. 31. 1. 2016] Dostupné z: https://cs.wikipedia.org/wiki/Amdahl%C5%AFv_z%C3%A1kon

PŘÍLOHA A

NAMĚŘENÉ ČASY PRO SDÍLENOU PAMĚŤ

Výsledky pro přidělení jednoho procesu.

č. měření	sériová část [s]	paralelní část [s]
1	0,21	12,33
2	0,21	15,28
3	0,23	15,23
4	0,23	17,32
5	0,21	15,67
6	0,24	13,9
7	0,23	16,05
8	0,23	15,21
9	0,22	13,77
10	0,23	14,95
Průměr [s]	**0,224**	**14,971**

Výsledky po přidělení max. počtu procesů (8)

č. měření	sériová část [s]	paralelní část [s]
1	0,23	3,06
2	0,23	5,05
3	0,25	5,72
4	0,23	7,45
5	0,23	5,14
6	0,23	4,88
7	0,24	4,91
8	0,23	5,86
9	0,21	4,16
10	0,23	5,91
Průměr [s]	**0,231**	**5,214**

PŘÍLOHA B

NAMĚŘENÉ ČASY PRO DISTRIBUOVANOU PAMĚŤ

Výsledky pro přidělení jednoho farmera a jednoho workera.

č. měření	sériová část [s]	paralelní část [s]
1	0,55	68,51
2	0,57	58,45
3	0,58	66,04
4	0,57	87,5
5	0,66	47,89
6	0,75	36,91
7	0,55	44,91
8	0,57	60,62
9	0,56	49,55
10	0,56	46,82
Průměr [s]	**0,592**	**56,72**

Výsledky pro přidělení jednoho farmera a 4 workerů.

č. měření	sériová část [s]	paralelní část [s]
1	0,84	33,03
2	0,82	31,33
3	0,88	30,62
4	0,93	34,25
5	0,85	21,06
6	0,87	36,38
7	0,86	32,13
8	0,93	37,32
9	0,91	38,21
10	0,82	21,27
Průměr [s]	**0,818**	**30,157**