ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

KIV/PPR

Standardní zadání pro Intel TBB & MPI

Autor: Antonín NEUMANN, A14N0139P

E-mail: neumann@students.zcu.

Akademický rok: 2015/2016

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:

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

$$\alpha = cg \tag{2}$$

$$\beta = p - cg \cdot i(t) \tag{3}$$

$$\gamma = c - i(\varphi(t)) \tag{4}$$

$$D = \beta^2 - 4 \cdot \alpha \cdot \gamma \tag{5}$$

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

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 Doporučené řešení

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

Ačkoliv funkce *b(t)* ovlivňuje funkci *i(t)* více než naopak, obě funkce *b(t)* a *i(t)* na sobě vzájemně závisí. Proto může být někdy být parametr *cg* či *c* nula, případně mohou být oba nenulové.

1.2 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 koncentraci 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 Program

- Vytvořte jednoduchý program, může to být i konzolová aplikace
- Program si vezme z příkazové řádky jméno souboru, ve kterém jsou naměřené hodnoty proměnné a jméno souboru, ve kterém jsou stanoveny meze parametrů
- Program zapíše nalezené parametry do tabulky difuse2params; s bude vždy 1
- Po dokončení výpočtu se uživateli vypíší souhrnné informace o době běhu programu, parametrech použitého algoritmu, a v přehledné tabulce vypíše nalezené parametry a vypočítané hodnoty metriky
- Vypište další statistiky

1.4 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

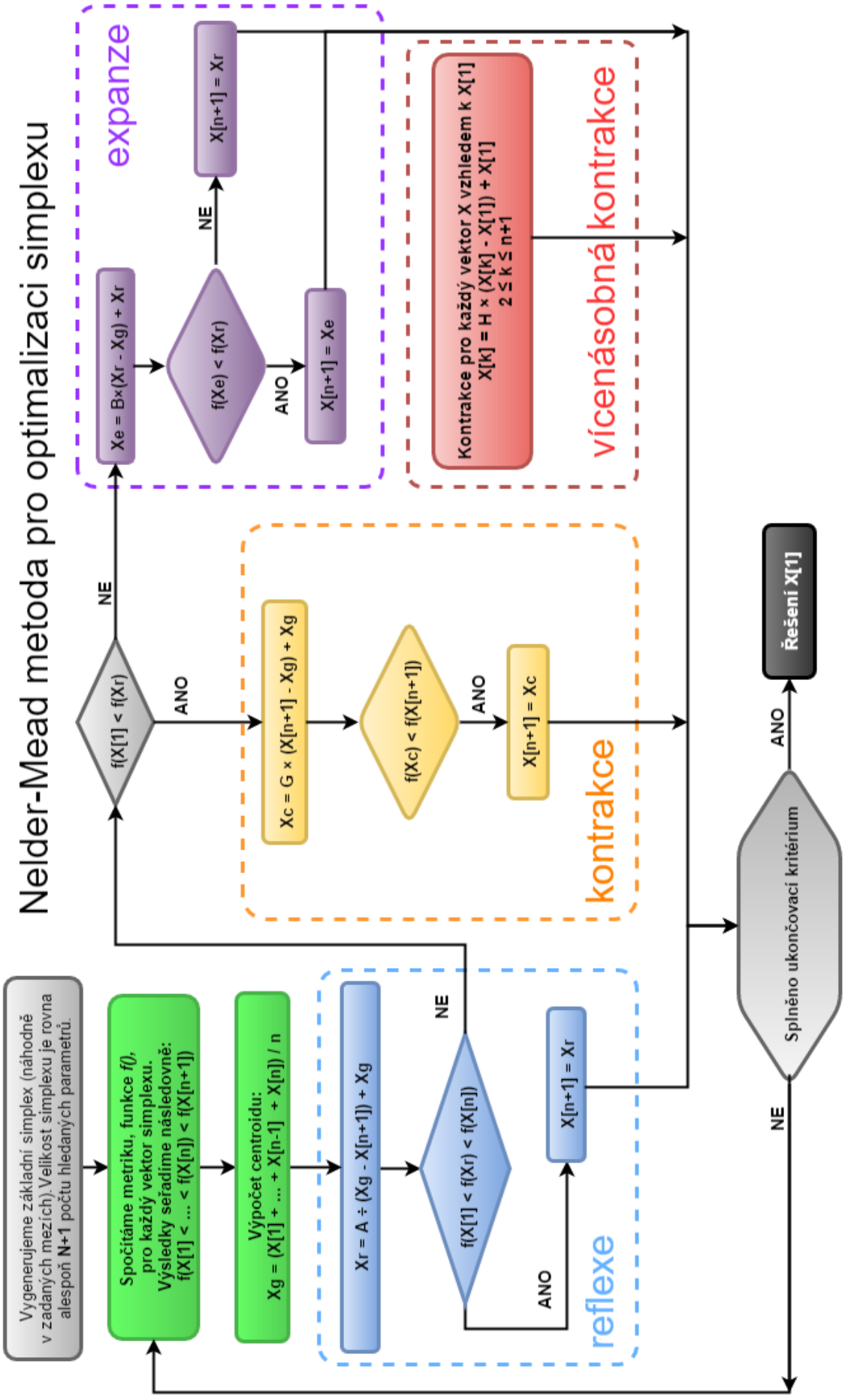
2 Algoritmy

2.1 Nelder-Mead metoda pro optimalizaci simplexu

Tento algoritmus byl doporučeným řešením pro nalezení parametrů rovnice *(6)*, jedná o numerickou metodu nalezení minima, použito v mém případě, nebo maxima funkce vícerozměrného prostoru. Algoritmus je dobře popsaný v anglickém jazyce na stránkách codeguru panem Botao Jia [1].

Algoritmus pracuje s tzv. Simplexem, jde o mnohostěn tvořený N+1 vrcholy v N dimenzionálním prostoru *(ve 3D prostoru by byl tvořen 4 vrcholy)*. Simplex v mém případě představuje matici tvořenou sadou parametrů (*p*, *cg*, *c*, *dt*, *h* a *k*). Jelikož máme celkem 6 parametrů, počet řádků matice simplexu musí být nejméně 7.

Celý průběh algoritmu je možné vidět na obrázku 1 – jedná se vlastní počeštěný přepis podle diagramu pana Botao a celý algoritmus je z něho snadno pochopitelný. K optimalizaci Nelder-Mead algoritmem, potřebujeme vygenerovat simplex – více v samostatné části – a umět spočítat metriku, v diagramu funkce *f()*, která je definovaná v zadání jako, *součet průměrné relativní chyby a standardní odchylky relativních chyb.* V diagramu uvedené proměnné *X*, představují vektor všech 6 parametrů.



Obrázek 1: Nelder-Mead optimalizace simplexu

Jednotlivé prvky simplexu na začátku (po spočtení jejich metrik) vždy seřadíme vzestupně podle hodnoty metriky. Tímto postupně dochází ke zlepšování vždy té nejhorší metriky, vyjma vícenásobné kontrakce u které dochází ke zlepšení všech mimo dosud nejlepší nalezené metriky.

2.2 Interpolace hodnot funkce i(t)

Z důvodu odlišné frekvence měření hodnot i(t) a b(t) uvedené v zadání, jsem se rozhodl hodnoty funkce i(t) interpolovat.

Samotná interpolace je prováděna knihovnou ALGLIB (dostupné na www.alglib.net) pomocí metody akima (funkce *alglib::spline1dbuildakima*) a to základě porovnání uvedeného v dokumentu Comparison of linear, cubic spline and akima interpolation methods [2].

3 Implementace

Implementace mého programu je z velké části ovlivněna programováním v OOP jazycích, zejména Javě a tak jsem i v C++ sáhl po objektovém programování za pomocí tříd.

Moje struktura programu by se dala rozdělit do několika logických skupin, které popisuje obrázek 2. Dále má každý soubor vlastní hlavičkový soubor, speciální hlavičkové soubory obsahující nastavení programu a použité konstanty jsou v diagramu zakresleny zvlášť.

3.1 Drivery

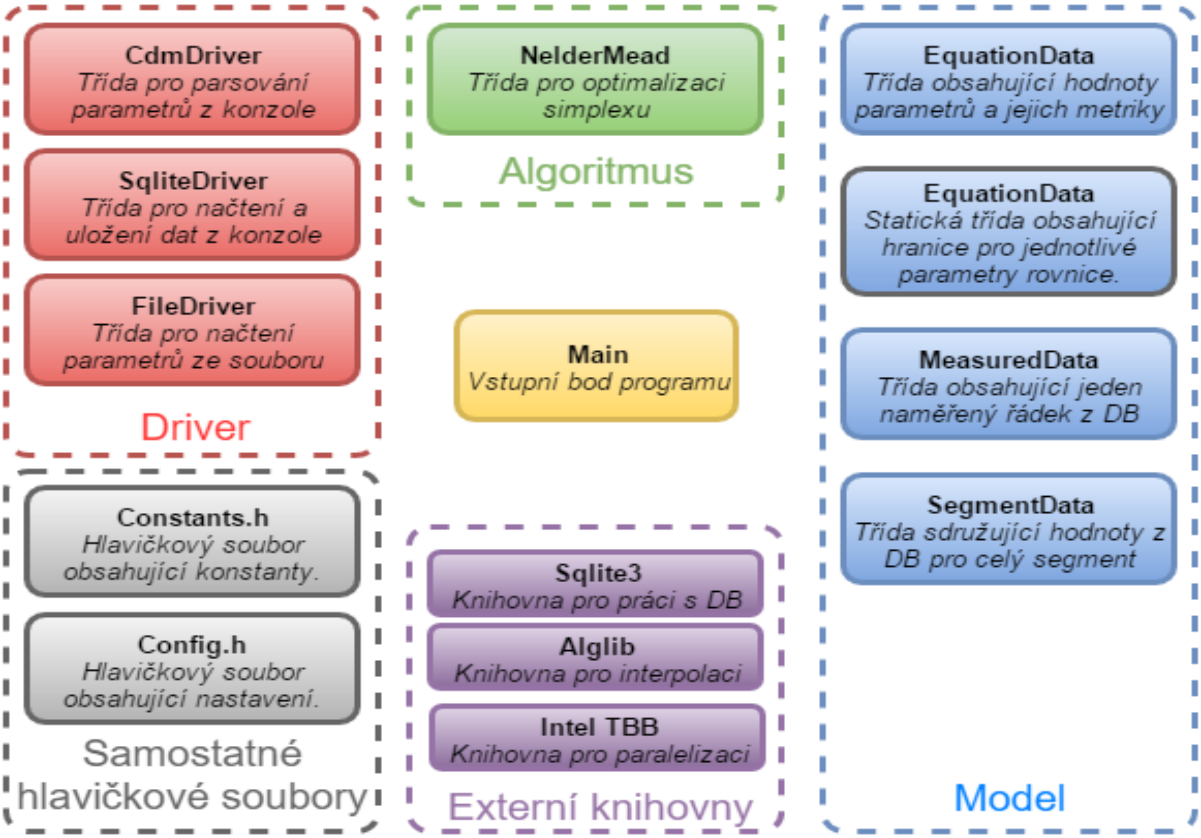
Třídy v této skupině slouží pro načítání a ukládání dat z/na externí médium. Jmenovitě CmdDriver pro načtení parametrů příkazové řádky a vypisující nápovědu k použití programu, FileDriver načítající meze parametrů ze zadaného souboru a SqliteDriver načítající data pro výpočet a ukládající výsledky pro SQLite3 databáze.

3.2 Model

Tato skupina obsahuje datové třídy, které slouží pro uložení dat do paměti a jejich manipulaci během výpočtu. Všechny modelové třídy mají přetížený operátor << pro jejich snadnější a přímočarý výpis na konzoly.

3.2.1 MeasuredData

Tato třída přímo odpovídá řádku tabulky a obsahuje ID řádku tabulky, datum (převedený z ISO 8601 do datového typu *double* s parametrem 1,0 = 1 den), hodnotu naměřené koncentrace glukózy v krvi a hodnotu naměřené koncentrace intersticiální tekutiny.



Obrázek 2: Struktura programu

3.2.2 EquationData a EquationBounds

Třída *EquationBounds* obsahuje pouze meze pro jednotlivé parametry a jedná se o statickou třídu.

Třída *EquationData* obsahuje všech 6 hledaných parametrů a jejich metriku, zároveň využívá třídu *EquationBounds* pro kontrolu mezí jednotlivých parametrů v metodách souhrnně označovaných jako *settery*.

3.2.3 SegmentData

Tato třída sdružuje jednotlivé naměřené hodnoty pro celý segment (vektor typu *MeasuredData*) a obsahuje i nejlepší parametry (typ *EquationData*). Dále obsahuje užitečné metody pro výpočet interpolované hodnoty *ist* a další.

3.3 Algoritmus

Samotný algoritmus pro výpočet pomocí Nelder-Mead optimalizační metody je realizován třídou Nelder-Mead a velmi věrně reflektuje popis algoritmu uvedeného na obrázku 1. Včetně komentářů, která část kódu odpovídá reflexi, kontrakci, atp.

Jako ukončovací kritéria jsem zvolil 10000 iterací (lze nastavit) a pokles metriky pod zadané ε (opět lze nastavit). Řešením jsou poté parametry s nejmenší metrikou nacházející se ve vektoru na pozici 0.

Součástí této třídy je metoda pro výpočet jednotlivých metrik a metody určené pro generování a řazení simplexu.

3.4 Hlavičkové soubory

Každý zdrojový soubor má svůj vlastní hlavičkový soubor obsahující jemu potřebné závislosti a definice funkcí nebo tříd, atributů a metod.

Dále jsou v programu použity dva speciální hlavičkové soubory *Constants.h* obsahující některé konstanty potřebné pro běh programu – parametry A, G, B a H pro Nelder-Mead optimalizační algoritmus převzaté z ukázkové implementace toho algoritmu o pana Botao a dále implicitní meze pro jednotlivé parametry rovnice převzaté ze zadání na Portále.

Druhým hlavičkovým souborem je *Config.h* obsahující některé konstanty ovlivňující běh programu, například počet iterací pro ukončovací kritérium Nelder-Mead algoritmu.

3.5 Intel TBB a MPI

Rozdíl mezi implementací programu pomocí knihovny Intel Threading Building Blocks a Message Passing Library je pouze v souboru *main.cpp*, ostatní části programu jsou pro obě implementace totožné.

Program pracující s Intel TBB navíc obsahuje dvě mírně odlišné části kódu, které způsobují použití paralelního výpočtu nebo klasické sériové verze. Tyto části jsou odlišené pomocí preprocesorového makra `#ifdef`. Toto makro sice neumožňuje volbu mezi paralelní a sériovou verzí parametrem příkazového řádku, ale v případě zakomentování hlavičky *tbb.h* v souboru *config.h* je možné spustit program bez jakékoliv závislosti na knihovně Intel TBB.

Program využívající MPI pracuje standardně na bázi master – slave (vedoucí – podřízený), kdy vedoucí proces určuje co má který podřízený dělat. Bohužel se mi nepodařilo rozběhat posílání složitějších struktur, takže vedoucí posílá svým podřízeným jen číslo segmentu, které mají zpracovat a o jejich načtení i uložení se poté stará sám podřízený proces.

Celý program závislý na MPI by šel obdobně jako v případě knihovny Intel TBB oddělit od zbytku kódu definicí makra a sdružit jen společně se sériovou a paralelní verzí do jednoho souboru. Ovšem tím by se mohl stát soubor *main.cpp* zbytečně složitým a těžko pochopitelným.

4 Naměřené výsledky

Všechny měřené výsledky proběhli na počítači Lenovo ThinkPad E330, Core i5-3210M 2.5GHz, 8GB RAM v operačním systému GNU/Linux Ubuntu. A šlo o verze přeložitelné pomocí *makefile*.

4.1 Amdahlův zákon

Vyjadřuje maximální zlepšení programu, poté co je část kódu paralelizována. Rovnice 7 vyjadřuje Amdahlův zákon v základním tvaru a rovnice 8 potom ve tvaru potřebném pro výpočet.

$$S = \frac{1}{f + \frac{1-f}{N}} \tag{7}$$

$$f = \frac{\frac{N}{S} - 1}{N - 1} \tag{8}$$

Kde S je zrychlení systému definované jako podíl času „bez paralelizace“ a „s paralelizací“ a N je počet procesorů použitých při paralelizaci.

4.2 Gustafsonův zákon

$$S = N - \alpha \cdot (N - 1) \tag{9}$$

$$\alpha = \frac{N - S}{N - 1} \tag{10}$$

Rovnice 9 uvádí základní tvar a rovnice 10 potom variantu upravenou pro výpočet, proměnné jsou stejné jako u Amdahlova zákona.

4.3 Karp-Flattova metrika

$$e = \frac{\frac{1}{S} - \frac{1}{N}}{1 - \frac{1}{N}} \tag{11}$$

Zde je rovnice 11 již ve svém základním tvaru použitelní pro výpočet, proměnné jsou opět shodné.

Typ programu (počet použitých procesů)	Čas běhu programu [µs]	Amdahlův zákon	Gustafsonův zákon	Karp-Flattova metrika
Sériová verze (1 proces)	55485755	-	-	-
Intel TBB verze (4 procesy)	29401092	0,3731805458	0,704266506	0,3731805458
MPI verze (5 proceů – 1 master, 4 slave)	46657836	0,7878634134	0,9369315013	0,7878634134

5 Uživatelský manuál

5.1 Překlad

5.1.1 Projekt pro Visual Studio

V projekt pro VS 2013 je nutné nastavit patřičné cesty ke knihovně Intel TBB. Mnou nastavené cesty jsou vidět na obrázcích v příloze B.

5.1.2 Verze pro Hydru

Verze testovaná na Ubuntu a školním serveru Hydra, používá pro překlad *makefile*, stačí tedy v adresáři zadat příkaz *make* a zdrojové soubory se přeloží do spustitelného souboru.

5.2 Spuštění a parametry

Spuštění programu přeloženého pomocí make je možné jednoduchým skriptem run.sh nebo příkazem: ./program BOUNDS_FILE SQLITE_FILE

Při spuštění bez parametrů program vypíše drobnou nápovědu, jiné parametry nejsou implementovány.

5.3 Nastavení

Všechna nastavení ovlivňující běh programu se provádějí v souboru Config.h, lze zde nastavit počet iterací Nelder-Mead algoritmu, použití knihovny Intel TBB a další. Vše je dobře okomentováno přímo v příslušném souboru.

6 Závěr

Práce v C++ pro mě byla nová a většinu věcí jsem se učil za pochodu. Volba implementace pomocí tříd pro mě byla záchranná, jelikož jako programátor zvyklý na objektové jazyky s Garbage collectorem jako je Java, jsem byl rád, že i v C++ mohu používat staticky alokované objekty. Zřejmě je tím ovlivněna rychlost celého programu, ale zase je podle mého zlepšena celá jeho struktura.

Knihovna Intel TBB mě velice překvapila, téměř až nadchla. Podle mě, alespoň na obdobně složité programy, je naprosto zbytečné používat klasická vlákna.

U knihovny MPI vznikl ovšem problém s přenosem vlastních objektů. Takže jsem musel nechat každé vlákno samostatně načítat data, což opět zcela jistě zpomaluje celý výpočet. Tato část mojí implementace by tedy jistě zasloužila trochu upravit.

Ve výsledku ale vše funguje a počítá, některé metriky a tím pádem i parametry nevycházejí úplně hezky (soudě podle některých jiných, ale nevím co je u této metody ještě dobrá a co už špatná metrika), ale to je zřejmě způsobeno náhodným generováním parametrů, které ne vždy vygeneruje optimální hodnoty, které by optimalizační metoda dokázala optimalizovat jako nejlepší možné.

7 Zdroje literatury

1. Jia Botao, Simplex Optimization Algorithm and Implemetation in C++ Programming, 7. 7. 2010, [citováno 31. 1. 2016]. Dostupné on-line: http://www.codeguru.com/cpp/article.php/c17505/Simplex-Optimization-Algorithm-and-Implemetation-in-C-Programming.htm
2. Hüseyin Özdemir, Comparison of linear, cubic spline and akima interpolation methods, 20. 8. 2007, [citováno 31.1. 2016]. Dostupné on-line: http://www.jive.nl/jivewiki/lib/exe/fetch.php?media=expres:fabric:interpolation.pdf
3. Wes Kendall, MPI Hello World, [citováné 31. 1. 2016]. Dostupné on-line: http://mpitutorial.com/tutorials/mpi-hello-world/

8 Ostatní zdroje

1. Intel Threading Building Blocks (Intel TBB), https://www.threadingbuildingblocks.org/
2. Open MPI, https://www.open-mpi.org/
3. ALGLIB, http://www.alglib.net/
4. SQLite, https://www.sqlite.org/

9 Přílohy

9.1 Příloha A – vybrané zdrojové kódy

- Soubor Config.h
- Soubor Constants.h
- Soubor main.cpp pro Intel TBB
- Soubor main.cpp pro MPI
- Soubor Nelder-Mead.cpp

Přiložené jako samostatná příloha v souboru Priloha_A.pdf

9.2 Příloha B – nastavení cest knihovny Intel TBB ve VS 2013

Přiložené jako samostatná příloha v archívu Priloha_B.zip