VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Paralelizace genetického algoritmu pomocí Cilk++

Parallelization of a Genetic Algorithm with Cilk++

2013

Michal Golis

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Golis**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Paralelizace genetického algoritmu pomocí Cilk++
Parallelization of a Genetic Algorithm with Cilk++

Zásady pro vypracování:

Cílem práce je navrhnout a implementovat paralelní verzi genetického algoritmu pomocí technologie Cilk++.
Diplomant se seznámí s jazykem Cilk++ a metodou genetických algoritmů. V rámci práce navrhne a implementuje genetický algoritmus v Cilk++.

Seznam doporučené odborné literatury:

[1] Evoluční výpočetní techniky - principy a aplikace, Oplatková, Zuzana; Ošmera, Pavel; Šeda, Miloš; Včelař, František; Zelinka, Ivan, BEN
[2] Intel Cilk Plus, Lambert M. Surhone (Editor), Mariam T. Tennoe (Editor), Susan F. Henssonow (Editor). Betascript Publishing (December 2010)

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Krömer, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 31. července 2013 ..... Michal Golej .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 31. července 2013 ..... Michal Golej .....

Chtěl bych poděkovat vedoucímu své bakalářské práce, Ing. Pavlovi Krömerovi, Ph.D.,
za příjemnou spolupráci, za spoustu cenných námětů i praktických rad, za kritické připomínky
i za přátelské povzbuzení ve chvílích, kdy jsem ztrácel optimismus. Také bych rád poděkoval
všem svým přátelům a známým, kteří mi poskytli pomoc a podporu.

Abstrakt

Genetické algoritmy a jejich paralelizace jsou v informačních technologiích využívanou strategií k vyhledávání vhodných kombinací řešení optimalizačních úloh. Cílem práce je vytvořit základní přehled jednoduchých a paralelních genetických algoritmů, navrhnout projekt a implementovat paralelní verzi algoritmu pomocí technologie Cilk++, včetně jeho testování. Projekt řeší Problém obchodního cestujícího paralelním genetickým algoritmem za pomocí Island modelu a modelu Master – Slave. Výsledkem je množina nalezených řešení včetně porovnání efektivity a hodnocení kvality.

Klíčová slova: genetický algoritmus, evoluční výpočetní techniky, optimalizace, paralelismus, Cilk++

Abstract

Genetical algorithms and their parallelization strategy are used in informational technologies for searching suitable combinations to solve optimization tasks. The purpose of this work is to create basic summary of simple and parallelal genetical alghorithms, design a project and implement parallel version of algorithm with Cilk++technology including it's testing. The project solves the Problem of traveling salesman with parallel genetical algorithm with The Island model and the model Master – Slave. The result is a set of found solutions including the comparison of efficiency and evaluation of quality.

Keywords: genetic algorithm, evolutionary computation, optimalization, paralelism, Cilk++

Seznam použitých zkratek a symbolů

DAG	–	Directed acyclic graph
EVT	–	Evoluční výpočetní techniky
GA	–	Genetický algoritmus
I	–	Island
IF	–	Input file
M	–	Master-Slave
MIT	–	Massachusetts Institute of Technology
MO	–	Mode
MR	–	Migration rate
NoE	–	Number of evolutions
NoO	–	Number of objects
OF	–	Output file
opt	–	optimální
PGA	–	Paralelní genetický algoritmus
S	–	Serial
SCX	–	Sequential constructive crossover
TSP	–	Traveling salesman problem

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Genetické algoritmy ve výpočetní technice jsou poměrně mladou disciplínou a jsou spojeny se jmény Lawrence J. Fogel, Ingo Rechanberg, Hans-Paul Schwefel a John Rolland, kteří se v 50. - 80. letech minulého století zabývali studiem evoluční strategie, evolučního programování a genetických algoritmů. V současné době jsou předmětem intenzivního studia všechny tři směry sloučené do jedné oblasti nazývané evoluční výpočetní techniky.

Bakalářská práce je formálně členěna do dvou částí. První část práce je teoretická, charakterizuje základní principy GA se zaměřením na jejich vlastnosti a možnosti paralelizace, jsou zde uvedeny výhody, ale také nevýhody GA. Druhá část popisuje konkrétní úlohu obchodního cestujícího se zaměřením na implementaci, testování a hodnocení GA v prostředí CILK++.

2 Genetické algoritmy

Genetické algoritmy (GA) jako numerické - heuristické algoritmy patří do skupiny evolučních výpočetních technik (EVT), které se inspirují přírodními vědami. Prakticky se GA využívají v mnoha oblastech lidské činnosti, a to jak technických, tak i netechnických odvětvích ke zkoumání různorodých řešení a jejich kombinací a vytváří základ dalších programů a specifických operátorů [6].

GA jsou metody pro prohledávání stavového prostoru aplikovatelné na širokou škálu typů úloh bez závislosti na typu řešeného problému. GA patří do třídy stochastických algoritmů s využitím vlastností deterministických algoritmů. GA se liší od náhodných prohledávacích metod tím, že kombinují elementy řízeného i stochastického prohledávání a pracují s celou populací potenciálních řešení [4].

Ve své podstatě simulují evoluční proces, který pozorujeme v přírodě již po tisíciletí. Vychází přímo z principů Darwinovy teorie přirozeného výběru a Mendelovy teorie genetiky a přebírají výrazy biologických disciplín např. gen, genotyp, chromozom, fenotyp a další [10].

K řešení problému pomocí GA je zapotřebí splnit určité požadavky, a to především:

- zvolit algoritmus pro vytvoření počáteční populace,
- vhodně zvolit formát, tzn. reprezentaci chromozomu (kandidátní řešení problému),
- sestavit algoritmus pro účelovou funkci,
- vytvořit algoritmus pro genetické operace: selekci, křížení a mutaci,
- vhodně nastavit výběr parametrů - velikost populace, maximální počet generací apod. [4].

Výhody:

- může vyřešit jakýkoliv optimalizační problém, který lze zapsat jako chromozom,
- konečným výsledkem je několik řešení,
- GA jsou jednoduché na pochopení bez rozsáhlých znalostí matematiky,
- GA je možné snadno přenést do stávajících simulací a modelů [4].

Nevýhody:

- u některých problémů jde velmi těžce definovat funkci vhodnosti a účelovou funkci,
- není zaručeno nalezení globálního optima,
- vysoká výpočetní náročnost [4].

2.1 Terminologie genetických algoritmů

Každý jedinec má svůj seznam parametrů (obvykle jsou parametry vyjádřeny v binární podobě), které se u genetických algoritmů nazývají geny. Všechny geny pak společně vytváří řetězec, kterému říkáme chromozóm. Genotypem nazýváme sadu parametrů chromozomu. Chromozom můžeme chápat jako kandidátní řešení genetického algoritmu. GA musí mí pro svou funkci definovanou účelovou funkci (cost function) a funkci vhodnosti (fitness function) [6].

Účelová funkce je matematický model problému, resp. životní prostředí, kde jedinci žijí a probíhá simulace evoluce. Funkce vhodnosti potom transformuje výsledek účelové funkce do daného intervalu [4].

2.2 Postup genetického algoritmu

Populace genetického algoritmu je tvořena jedinci, kteří jsou reprezentováni chromozomy (kandidátní řešení problému). Každý jedinec také obsahuje vhodnost (ohodnocení), která udává kvalitu jedince. Vlivem křížení nejlepších jedinců populace vznikají noví a snad i silnější jedinci. Simulací evoluce noví jedinci přežívají a dále se kříží, slábnou nebo umírají. Postup genetického algoritmu (Obr. 1) lze shrnout do následujících desíti bodů [6].

2.2.1 Vymezení parametrů

Jedná se o definování parametrů, které budou řídit běh algoritmu, nebo ho ukončí. Součástí parametrů je i účelová funkce a vhodnost. Jako parametr se také udává maximální počet generací-evolučních kroků. Hodnota počtu evolucí se udává proto, aby se zabránilo nekonečné evoluci [6].

2.2.2 Generování počáteční populace

Jedná se o vytvoření počáteční množiny jedinců a jejich parametrů. Parametry jedinců jsou zvoleny nahodile a tak každý jedinec představuje možné, ne příliš kvalitní řešení. V tomto kroku je kritickým parametrem velikost populace. Příliš velká populace zabere více času na ohodnocení jedinců, naopak příliš malá populace nebude mít dostatek genetického materiálu na další evoluční vývoj [6].

2.2.3 Ohodnocení populace

Jedná se o vyhodnocení jedinců počáteční populace účelovou funkcí. Vrácená hodnota může být ještě normalizována funkcí vhodnosti [6].

2.2.4 Výběr rodičů

Jedná se o výběr jedinců pro reprodukci (křížení). Jedinci jsou většinou vybíráni podle vhodnosti, ale mohou být vybráni i podle jiných kritérií v závislosti na řešené úloze [6].

2.2.5 Tvorba potomků

Křížením kombinací genů rodičů jedinců vznikají noví jedinci. Obvykle se používá metoda jednobodového křížení (crossover point). Křížení je zcela závislé na typu problému a na reprezentaci genů chromozomu [6].

2.2.6 Mutace potomků

Jedná se o náhodnou změnu n genů v chromozomu (simulace biologické mutace). Pravděpodobnost mutace by neměla být příliš velká, při velké pravděpodobnosti je prohledávání stavového prostoru náhodné [6].

2.2.7 Ohodnocení potomků

Po vzniku a mutaci jedinců nové generace evolučního kroku je potřeba nové jedince ohodnotit účelovou funkcí s případnou normalizací [6].

2.2.8 Výběr nejlepších jedinců

Nejlepší jedinci jsou vybíráni z množiny rodičů a potomků. Výběr jedinců je prováděn dle vhodnosti nebo podle jiných kritérií, v závislosti na řešené úloze [6].

2.2.9 Vytvoření nové populace

Nová populace je vytvářena výběrem z nejlepších jedinců [6].

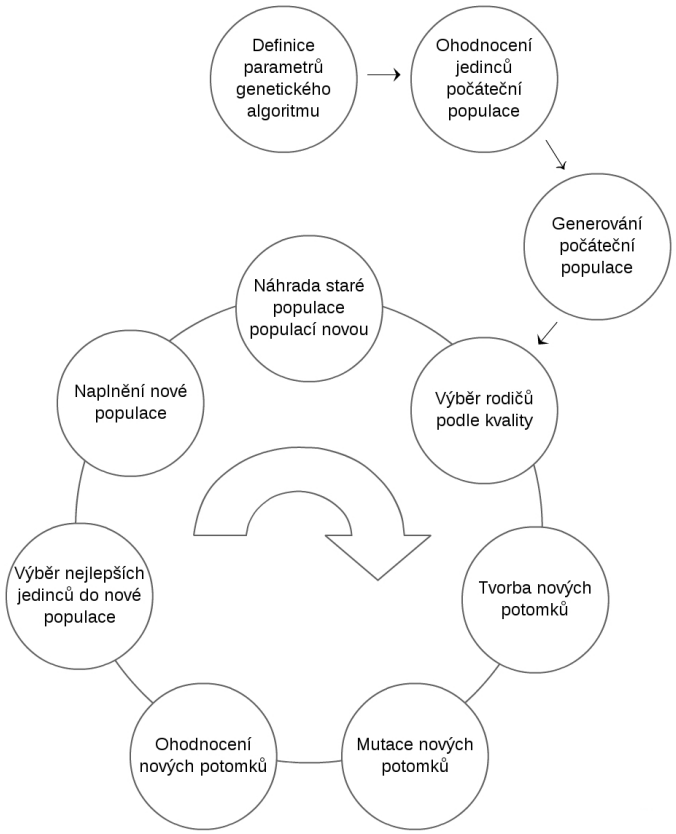
2.2.10 Redukce staré populace

Stará populace je likvidována redukcí nejhorších jedinců podle zvolené strategie [6].

Výběr rodičů, tvorba potomků, mutace potomků, ohodnocení potomků, výběr nejlepších jedinců, vytvoření nové populace a redukce staré populace se opakují tak dlouho, dokud není nalezeno dostatečně kvalitní řešení nebo do vyčerpání uživatelem zadaných počtu evolučních cyklů [6].

2.3 Účelová funkce

V genetických algoritmech účelová funkce (fitness function) ohodnotí každého jednotlivce v populaci. Tato hodnota určuje kvalitu jedince tj. vlastní řešení problému. Funkce vhodnosti je většinou určena jako část řešeného problému. Funkce by měla být co nejrychlejší vzhledem k tomu, že hodnotí celou populaci. V případě pomalé účelové funkce se využívají paralelní a distribuované systémy [3][4].



Obr. 1: Postup genetického algoritmu [6]

2.4 Kódování

Zakódování genů chromozomu je přímo závislé na řešeném problému. Existuje několik druhů kódování. Geny mohou být reprezentovány celými čísly i reálnými čísly. Nejčastěji se v praxi setkáváme s binárním a permutačním kódováním.

Binární kódování je nejrozšířenějším kódováním, které bylo používáno už v prvních genetických algoritmech. Kódování je velmi jednoduché. Je vhodné např. pro řešení Problému batohu (Knapsack problem). Pro některé úlohy však není zcela přirozené, proto je zapotřebí změnit algoritmus křížení a mutace [7].

Permutační kódování se používá u problému, kde řešení je jedna z permutací prvků, např. Problém obchodního cestujícího (Travelling salesman problem) [7].

2.5 Velikost populace

Velikost populace je jeden z parametrů genetického algoritmu. Velikost populace přímo ovlivňuje průběh genetického algoritmu, proto je potřeba zvolit tento parametr nanejvýš obezřetně. Příliš velká populace způsobí prodloužení výpočetního času a sníží efektivitu algoritmu, na druhou stranu malá populace může mít problém s nedostatkem informací pro identifikaci a vývoj lepších řešení [7].

2.6 Pravděpodobnost překrytí

Pravděpodobnost křížení udává, kolik procent jednotlivců ze staré generace přežije a přidá se do nové generace společně s potomky. Většinou je pravděpodobnost nenulová, existují ale i algoritmy s nulovou pravděpodobností překrytí [4].

2.7 Nahrazovací strategie

Po selekci rodičů se vytváří nová sada jednotlivců (potomků), která bude tvořit další generaci. V případě nulové pravděpodobnosti překrytí vznikne nová generace a stará zanikne. Při nenulové pravděpodobnosti křížení je potřeba určit strategii, která nahradí staré jedince novými. Je tedy potřeba rozhodnout, které jedince nahradit novou generací. Existuje několik přístupů, např. nahrazení rodičů potomky, nejhorší jedince nebo náhodné jedince nahradit právě vytvořenými jedinci. Tak dosáhneme opět původní velikosti populace. Nejčastěji se používá náhrada nejhorších jedinců [3][4].

2.8 Elitismus

Během evoluce mohou být, vlivem náhody, nejlepší jedinci z evolučního procesu vyřazeni. Nežádoucímu jevu lze zabránit přenesením nejlepších jedinců, většinou 1 až 3 jedinců, do další populace. Tento elitářský mechanismus účinně zamezí možnosti ztráty nejlepšího, dosud nalezeného jedince [3][4][7].

3 Operace genetických algoritmů

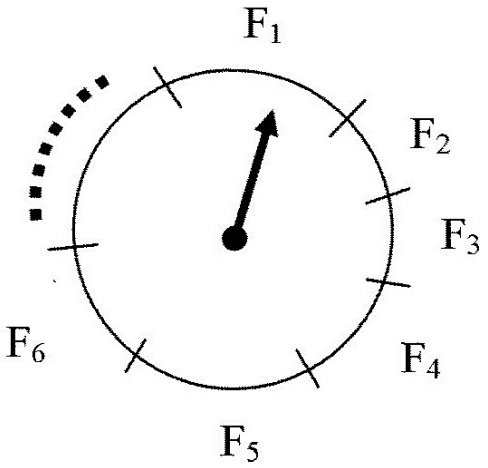
Za hlavní genetické operace jsou považovány selekce, křížení, mutace a nahrazení. Genetické operace vitálně ovlivňují průběh celého genetického algoritmu a ovlivňují zásadně vznik nové generace. Jednotlivé operace jsou přímo závislé na způsobu reprezentace chromozomu a řešeném problému, proto je nutné vybrat co nejpřirozenější způsob provádění operací vzhledem k řešenému problému.

3.1 Selekce

Jakmile je počáteční generace vytvořena a ohodnocena je čas vybrat jednotlivce pro vytvoření potomků tak, aby mohla být vytvořena další generace. Jednotlivci jsou vybíráni v jednoduchých genetických algoritmech podle vhodnosti. Existuje několik způsobů jak realizovat selekci, a to úměrnou selekcí (Roulette wheel), pořadovou selekcí (Rank selection), turnajem (Turnament) nebo jinou metodou, která je přirozená pro řešenou úlohu.

Úměrná selekce je metoda přidělení. Poměrově se každému jedinci přidělí v populaci číslo, které určuje šanci na jeho výběr, podle vlastního ohodnocení. Metoda se podobá ruletovému kolu, kde podle vhodnosti jedince je přiřazen počet políček ruletového kola, pak se provádí losování pro výběr každého dalšího rodiče (Obr. 2) [3][4][7].

Pořadová selekce je velmi podobná úměrné selekci. Používá se v případě, kdy je rozdíl



Obr. 2: Jednotková kružnice rulety [6]

kvality mezi jedinci příliš velký. Pořadová selekce přerozdělí jedince podle pořadí tak, že nejslabší jedinec bude ohodnocen číslem 1, silnější jedinec bude ohodnocen číslem 2, atd. až do nejsilnějšího jedince. Tak se jedinci stanou víceméně rovnoprávnými s podobnou šancí na výběr [3][4][7].

Turnaj je metoda selekce, která je prováděna formou soutěže, tzv. turnajem. Do soutěže se přihlásí několik jedinců a pouze nejlepší jedinec ze soutěže je připraven na reprodukci. Z generace jsou vybrány dva různé řetězce a do nové generace je vybrán nejsilnější z nich. Tento proces se několikrát opakuje [3][4][7].

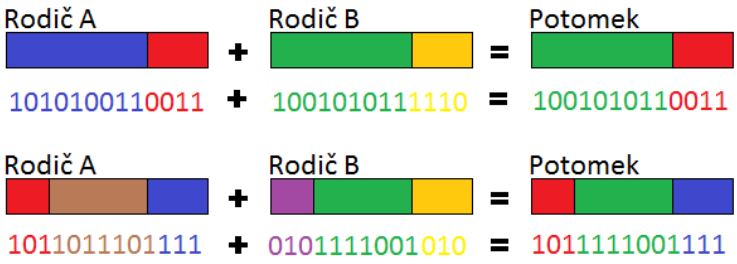
3.2 Mutace

Posledním krokem ve vytváření nové generace je mutace. Jedná se o změnu náhodných genů jedince. Operace je závislá na řešeném problému. Význam mutace spočívá v tom, že se může v dané generaci objevit vlastnost, kterou dosud žádný jedinec neměl a nemohl ji tedy předat potomkům. Tím je zabráněno uvíznutí v lokálním extrému [3][5][6][7].

3.3 Křížení

Operace spočívá ve výměně informací mezi jedinci. Výsledkem křížení je nový jedinec, který přejímá od každého rodiče nějaké vlastnosti. Metoda křížení je silně závislá na řešeném problému, např. jednobodové nebo dvoubodové křížení (Obr. 3) je založeno na kombinaci bloku genů.

Metodu nelze použít u problému, kde je chromozom zakódován jako permutace (nebezpečí vzniku permutace s opakováním) [3][5][6][7].



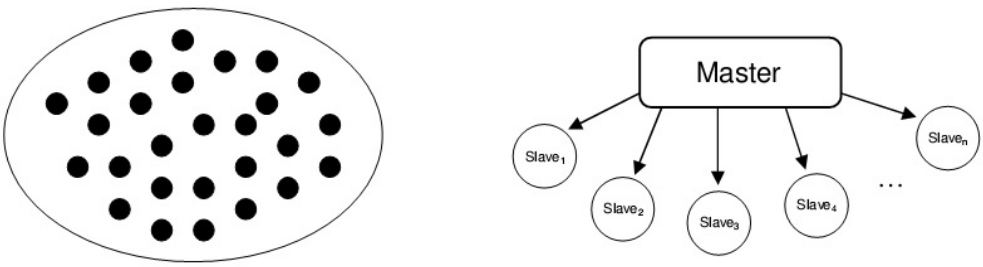
Obr. 3: Dvoubodové křížení [7]

4 Paralelní genetické algoritmy

Základní myšlenkou mnoha paralelních a distribuovaných programů je rozdělit úkol do dílčích oddílů (devide-and-conquer) a řešit je současně za pomocí většího počtu procesorů. Některé metody, jako je např. jemnozrnný paralelní genetický algoritmus, mohou využívat masivní paralelní architektury, zatímco jiné, např. hrubozrnný paralelní genetický algoritmus, se hodí lépe pro multi-počítače s vysokým výkonem, ale s menším počtem procesorů [1].

4.1 Globální paralelizace (Master-Slave model)

Podobně jako sekvenční GA v kontextu globální paralelizace je pouze jedna panmiktická populace, tzn. každý jedinec má šanci se spárovat s jiným jedincem. Chování algoritmu zůstává beze změny a globální GA má přesně stejné kvalitativní vlastnosti jako sekvenční GA. Nejběžnější paralelizovanou operací je vyhodnocení jednotlivců. Výpočet vhodnosti jedince je nezávislý od zbytku populace. Jeden hlavní uzel provádí GA (výběr, křížení a mutace) a další operace, např. ohodnocení jedinců je přiděleno podřízeným procesorům. Jednotlivé části populace jsou přiřazeny k dostupnému procesoru. Díky jejich centralizované hierarchické komunikaci jsou globální paralelní genetické algoritmy také známé jako single-master-slave populace. Populační struktura modelu master-slave je graficky znázorněna níže (Obr. 4). Panmiktický GA má všechny své jedince (černé skvrny) ve stejných populacích. Master proces ukládá populaci, provádí GA operace a distribuuje jednotlivce svým podřízeným. Podřízené procesy mají na starost výpočet vhodnosti. Globálně paralelní genetické algoritmy jsou poměrně snadně implementovatelné a mohou být docela efektivním způsobem paralelizace, pokud vyhodnocení vyžaduje značný výpočetní výkon [1][3].

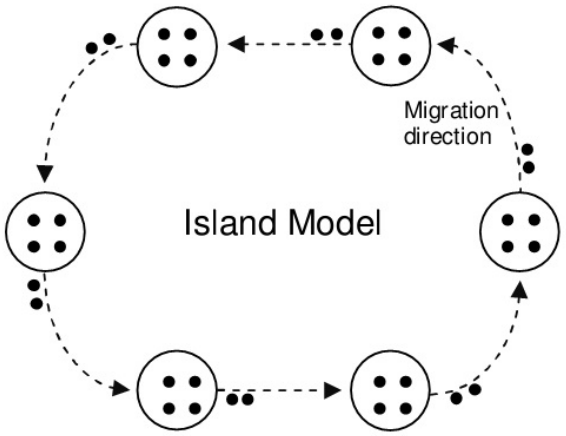


Obr. 4: Model Master-Slave [1]

4.2 Hrubozrnný paralelní genetický algoritmus (Coarse-Grained algorithm)

V případě hrubozrnných genetických algoritmů jsou populace rozděleny do několika sub-populací (ostrůvků). Sub-populace se vyvíjejí zcela nezávisle a jsou distribuovány mezi jednotlivé procesory, tzv. distribuovaný genetický algoritmus, který se většinou

používá s distribuovanou pamětí. Občas se během evolučního procesu provede výměna jednotlivců mezi sub-populacemi, tato operace se provádí každých n generací. Hrubozrnný paralelní genetický algoritmus (Obr. 5) reprezentují kruhy, kde jeden kruh je ostrov a jednotlivé kruhy spolu komunikují mezi GA. Hlavní myšlenkou hrubozrnného paralelního GA je, že relativně izolované chromozomy budou konvergovat k různým platným řešením [1][3].



Obr. 5: Island model [1]

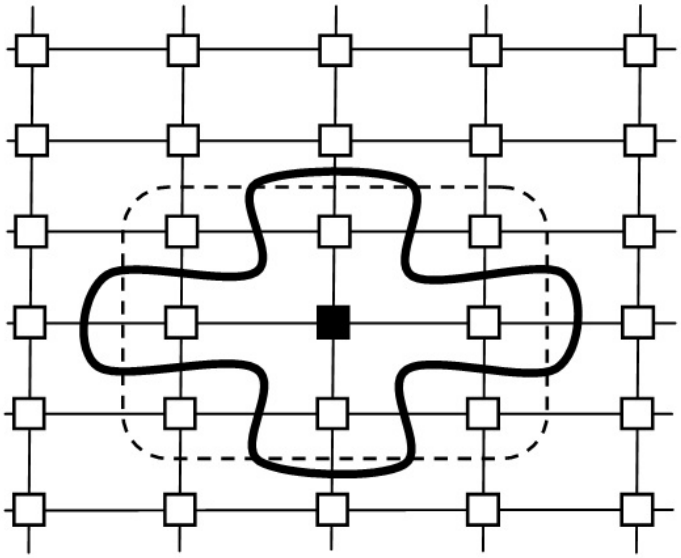
4.3 Jemnozrnný paralelní genetický algoritmus (Fine-Grained algorithm)

Model bývá někdy nazýván model sousedů (neighborhood model). Model namapuje jedince nebo velmi malé sub-populace jako mřížku mezi velký počet procesorů. V mřížce jsou všichni jedinci aktivní a hledají si své partnery na rekombinaci pouze mezi sousedními sub-populacemi (Obr. 6) [1][3].

4.4 Migrace

Migrace (synchronní a asynchronní) je důležitou operací u paralelních GA, která umožňuje výměnu jedinců mezi populacemi. Díky nezávislosti jednotlivých populací získáme uspokojivější a různorodější řešení rychleji než u bezmigračních algoritmů (globální GA). Migrace značně snižuje míru konvergence jednotlivých populací. Důležitými aspekty migrace jsou:

- správná definice topologie mezi sub-populacemi,
- rozhodnutí, který jedinec bude migrovat do které sub-populace,
- stanovení počtu, kolik jedinců bude migrovat,
- rozhodnutí, v jakých intervalech bude migrace prováděna [4][5][6].



Obr. 6: Struktura jemnozrnného algoritmu [1]

4.5 Problém nadměrného urychlení (Superlinear Speedups)

Jedním z kontroverzních témat, které se týká PGA, je problém nadměrného urychlení. Spočívá v rozporu mezi teoretickými předpoklady a empirickými výsledky.

Úvaha říká: pokud vyřešení libovolné úlohy trvá jedinému procesoru dobu t, pak je logické tvrdit, že n procesorům bude trvat řešení té samé úlohy minimálně dobu t/n. To znamená, že celý výpočetní proces se při použití n procesorů může urychlit maximálně n-krát.

Podle empirických výsledků dochází k většímu zrychlení a zkrácení výpočetního času.

Bylo vytvořeno několik pokusů ve snaze vysvětlit, kdy tento jev vzniká. Jedno z možných vysvětlení je to, že PGA provádějí méně činnosti než sériové GA. Redukce činnosti spočívá ve větším selekčním tlaku, který je způsoben migrací jedinců mezi populacemi. Jiným vysvětlením může být menší velikost populace. Tyto populace se mohou vejít do vyrovnávací paměti procesoru [4].

5 Projekt Cilk a jeho vývoj

Cilk, Cilk++ a Cilk plus jsou rozšíření pro jazyk C a C++, které nabízí rychlý, snadný a spolehlivý způsob, jak vylepšit výkon programů na vícejádrových procesorech [8].

5.1 Cilk

Cilk program vznikl a dále byl vyvíjen ve třech nezávislých projektech na MIT (Massachusetts Institute of Technology) vědeckých počítačových laboratořích:

- Theoretical work on scheduling multi-threaded applications,
- StarTech – a parallel chess program built to run on the Thinking Machines Corporation's Connection Machine model CM-5,
- PCM/Threaded-C – a C-based package for scheduling continuation-passing-style threads on the CM-5 [8].

V dubnu roku 1994 byly projekty spojeny do jednoho projektu Cilk. První verze, Cilk-1, byla vydána v září roku 1994. V době psaní této práce je současná verze implementace Cilk-5.3 [8].

5.2 Cilk++

V roce 2006 MIT licencovala Cilk Arts technologii Cilk s cílem rozvíjet komerční verzi C++ implementace. Cilk++ v1.0 byl vydán v prosinci 2008 s podporou pro Windows Visual Studio a pro GNU GCC/C++ compiler. Cilk++ se liší od Cilk-5:

- plnou podporou C++, včetně všech rozšíření,
- v C++ lze použít Cilk kód přímo, pokud je zkompilován Cilk compilerem,
- přejmenováním klíčových slov spawn a sync na cilk_spawn a cilk_sync,
- přidáním cilk_for smyčky pro paralelizaci,
- přidáním objektu reducer pro redukci races [8].

5.3 Intel Cilk plus

V roce 2009 získala licenci Cilk Arts Intel Corporation. Technologie Cilk byla sloučena s Array notací, aby technologie mohla poskytnout obsáhlé rozšíření jazyka na implementaci funkčního paralelismu (task parallelism) a vektorového paralelismu (vector parallelism). Intel Cilk plus je obohacen o kompatibilitu se standardními debugery [8].

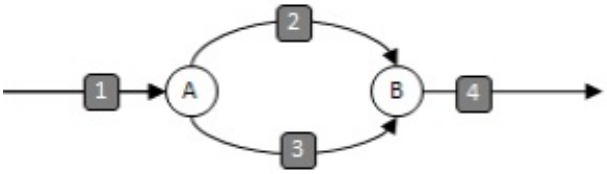
6 Výpočetní model Cilk

Průběh Cilk programu chápeme jako orientovaný acyklický graf (directed acyclic graph - DAG), kde jednotlivé hrany grafu reprezentují sériový průběh programu (cilk_spawn) a místo pro synchronizaci (cilk_sync).

Příklad DAG (Obr. 7 diagram) pro kód (Kód 1) spouští funkce do_work_1, do_work_2, do_work_3 a do_work_4. Funkce do_work_2 a do_work_3 se vykonávají paralelně. Funkce do_work_4 se začne vykonávat až po ukončení funkce do_work_2 a do_work_3 [11][12].

```
...
do_work_1(); //hrana 1
cilk_spawn do_work_2(); //hrana 2 a vytvoreni vrcholu A
do_work_3(); //hrana 3
cilk_sync; //vytvoreni vrcholu B
do_work_4(); //hrana 4
...
```

Kód 1: Jednoduchý cilk kód



Obr. 7: DAG diagram [12]

6.1 Reducer

Jazyk Cilk dodal metodu pro řešení problému, který se týká přístupu k ne-lokálním proměnným. Reducer je tedy proměnná, kterou lze použít více hranami při paralelním běhu programu. Reducer poskytuje každému vláknu privátní kopii proměnné bez nutnosti použití Lock. (Kód 2) Ukazuje použití reduceru při sčítání prvků vrácených funkcí compute [11][12].

```
...
   int n = 1000000;
   cilk::reducer_opadd<unsigned int> total;

   cilk_for (int i = 1; i <= n; ++i)
   {
       total += compute(i);
   }
...
```

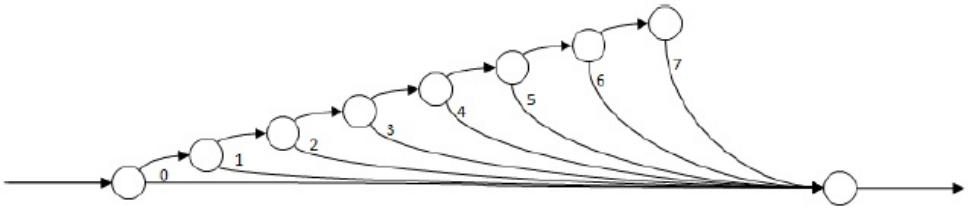
Kód 2: Jednoduchý reducer

7 Charakteristika jazyka Cilk

Programovací jazyk Cilk je charakterizován klíčovými slovy, pragmy, makry, prostředím s proměnnými a hlavičkovými soubory jazyka Cilk.

7.1 Klíčová slova

- **cilk_spawn** spouští zadanou funkci paralelně s volající funkcí. Funkce nemusí být paralelizována (záleží na prostředí Cilk) [11][12].
- **cilk_sync** udává bod synchronizace. Současná funkce nemůže pokračovat, dokud nejsou všechny funkce spuštěné příkazem cilk_spawn ukončeny (Obr. 8) [11][12].



Obr. 8: Průběh cilk_spawn bloků a jejich synchronizace_for [12]

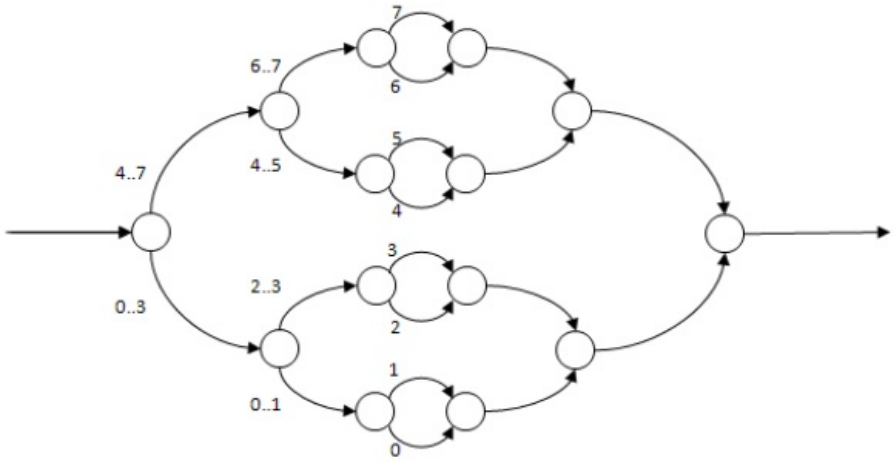
- **cilk_for** je náhradou klasické smyčky for. Smyčka cilk_for rozdělí své iterace do menších bloků (Obr. 9). Iterace jednotlivých bloků jsou vykonávány sériově, ale jsou volány příkazem cilk_spawn. Smyčka cilk_for musí být validní C/C++ for a nesmí měnit kontrolní proměnnou cyklu a ne-lokální proměnné (vzniká tak neintegrita paměti, které lze zabránit instancí reducer za cenu menšího paralelismu) [11][12].

7.2 Pragma

- **cilk grainsize** je proměnná, která udává maximální počet iterací bloku. Grainsize je nastavován automaticky prostředím Cilk podle počtu dostupných vláken (workers). Výchozí hodnota je vhodná pro většinu případů mimo ty, kdy se počet vláken při běhu programu mění [11][12].

7.3 Macro

- **__cilk** je jednoznačné určení verze jazyka Cilk. Makro je nastaveno automaticky compilerem [11][12].



Obr. 9: Průběh cilk_for [12]

7.4 Proměnná prostředí

- **CILK_NWORKERS** specifikuje počet vláken, která se budou podílet na paralelizaci [11][12].

8 Problém obchodního cestujícího

Zadáním problému je ohodnocený graf nebo také jednoduchý neorientovaný nebo orientovaný ohodnocený graf G. Graf G je uspořádaná trojice $G = (V, E, C)$, kde V je množina vrcholů, E je množina (dvouprvková podmnožina množiny V) hran a C je vzdálenost mezi vrcholy V.

Problém obchodního cestujícího je kombinatorická úloha patřící do skupiny NP-úplných problémů. Pro problém neexistuje žádný polynomiální k-aproximační algoritmus, tzn. neexistuje žádný polynomiální algoritmus, který by našel libovolné řešení, které je nejhůře k-násobkem optimálního řešení [13].

Cílem této úlohy je najít v ohodnoceném neorientovaném grafu co nejkratší Hamiltonovskou kružnici, která prochází všemi vrcholy grafu. Jedná se o nalezení nejkratší cesty mezi městy tak, aby jejich součet byl co nejmenší.

V práci je řešena symetrická varianta TSP, jedná se o neorientovaný graf, kde TSP má uloženou symetrickou matici vzdálenosti, kde v matici c platí - $c_{ij} = c_{ji}$.

9 Implementace

Bakalářská práce obsahuje program, kde jsou implementovány tři algoritmy, které řeší TSP problém prostřednictví GA, dva algoritmy jsou paralelní (Master-Slave model a Island model) a jeden je sériový. Program byl kompilován v Intel® C++ Compiler XE, který je součástí nekomerčního balíku Intel® Parallel Studio XE 2013 (volně stažitelný [1]).

Syntaxe pro program:

./program {IF} {NoO} {NoE} {MR} {OF} [MO]

Argumenty programu:

- IF udává cestu ke vstupnímu souboru,
- NoO udává velikost populace; velikost populace musí být minimálně 10 pro 1 jádro procesoru; v případě použití více procesorů je potřeba zvolit větší velikost populace,
- NoE udává počet evolucí,
- MR udává četnost migrace populace,
- OF udává jméno pro výstupní soubor; výstupní soubor je vždy uložen do složky output v adresáři s programem,
- MO udává jaký typ algoritmu má být spuštěn, v případě nezadání tohoto parametru jsou spuštěny všechny implementované algoritmy (Tab. 1).

9.1 Konfigurace

Testovací konfiguraci představuje čtyřjádrový procesor Intel®Core™ i5-3210M 2,5 GHz, 8GB RAM s operačním systémem Debian GNU/Linux 7 (wheezy).

9.2 Reprezentace chromozomu

Pro reprezentaci chromosomu jsem zvolil permutační kódování. Permutační kódování je vzhledem k TSP nejpřirozenější, protože řešením je permutace měst. Chromozom je tvořen permutací z n prvků, kde n je počet procházených měst a reálným číslem R, kde R je součet vzdálenosti cest mezi městy. R udává kvalitu jedince.

[1] http://software.intel.com/en-us/non-commercial-software-development

hodnota	spuštěné algoritmy I (Island) M (Master-Slave) S (Serial)
0	$---$
1	$--S$
2	$-M-$
3	$-MS$
4	$I--$
5	$I-S$
6	$IM-$
7	IMS

Tabulka 1: Módy programu

9.3 Funkce vhodnosti

Vhodnost jedince udává pořadí cest mezi městy tvořícími permutaci. Stačí pouze sečíst n čísel udávajících délku cesty. Pro výpočet je na začátku běhu programu přečten vstupní soubor, který udává souřadnice jednotlivých měst a po přečtení souřadnic jsou vypočítány vzdálenosti mezi jednotlivými městy. Vzdálenosti mezi městy jsou uloženy do matice $n \times n$, kde první dimenze udává město startovní a druhá dimenze udává cílové město (Kód 3).

```
float Chrom::calcFitness( float ** map )
{
   this->fitness = 0;
   for(int i = 0; i < (*chromSize)-1; i++)
      this->fitness += map[ chromosome[i] ][ chromosome[i+1] ];
   this->fitness += map[ chromosome[0] ][ chromosome[(*chromSize)-1] ];

   this->fitness
}
```

Kód 3: Funkce vhodnosti

9.4 Selekce jedinců

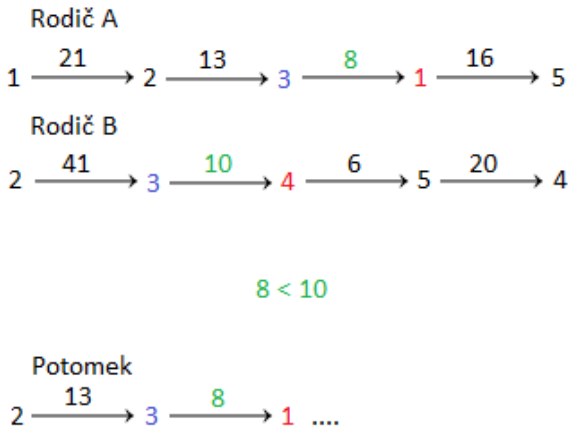
Selekce jedinců probíhá pomocí pořadové selekce. Selekce je normalizována tak, aby i slabší jedinci měli šanci na reprodukci. Možnost výběru slabších jedinců by měla zaručit větší rozmanitost populace a možná i nalezení lepších jedinců v případě, kdy křížení nejlepších jedinců již nevytváří silnější jedince (Kód 4).

```
int Serial::ruletteWheel( int bot, int top)
{
    int arit = ( (top-bot+1) * (bot+top) ) / 2;
    int n = rand_r( &seed ) % arit;

    for(int i = bot; i <= top; i++)
    {
        if ( n >= arit)
            return top - i;
        arit -= i;
    }

    return top;
}
```

Kód 4: Úměrná selekce jedinců

9.5 Křížení jedinců

Křížení jedinců je realizováno sekvenčním konstruktivním křížením (The sequential constructive crossover SCX) (Obr. 10) [2]. Metoda je založena na kombinaci nejlepších genů rodičů tak, aby vznikl silnější potomek. V případě TSP je výhodné kombinovat kratší cesty mezi městy. Metoda porovnává cestu rodiče P1 z města A do města B a rodiče P2 z města A do města B, kde SCX metoda vybere kratší cestu. Algoritmus má jistou šanci, že vytvoří permutaci měst, kde se budou některá města opakovat. Proto je zapotřebí zaznamenávat již navštívená města a v případě opakování stejných měst nahradit města za nová, ještě nepoužitá. Nepoužité město k opravě permutace je zvoleno metodou lokálního vyhledávání (local-search), kde je vybráno nejbližší možné město.



Obr. 10: SCX

9.6 Mutace jedinců

Mutace byla provedena záměnou pořadí měst v permutaci. V případě TSP je lepší vyměňovat města, která jsou v permutaci velmi blízko. Mutace mění pozici jednoho města až jednoho procenta z celkového počtu měst. Mutace je použita pouze v případě nalezení totožných jedinců v populaci (Kód 5).

```
void Chrom::mutate(unsigned int seed)
{
    int x, y, temp;
    int limit = 0;
    int nMutations;
    int tempChromSize = *chromSize;
    int pivot = rand_r( &seed ) % (*chromSize);

    while(tempChromSize)
    {
        tempChromSize /= 10;
        limit ++;
    }

    nMutations = limit == 1 ? 1 : rand_r( &seed ) % (limit - 1) + 1 ;

    while(nMutations--)
    {
        x = rand_r( &seed ) % (limit + limit + 1) - limit ;
        y = rand_r( &seed ) % (limit + limit + 1) - limit ;

        temp = chromosome[(x + *chromSize) % *chromSize];
        chromosome[(x + *chromSize) % *chromSize] = chromosome[(y + *chromSize) % *
            chromSize];
        chromosome[(y + *chromSize) % *chromSize] = temp;
    }
}
```

Kód 5: Mutace jedinců

9.7 Nahrazovací strategie

Nahrazovací strategie udává novou formu nové generace rozhodnutím, kteří jedinci přežijí do dalšího evolučního cyklu a kteří budou nahrazeni. V projektu je nahrazováno 60% populace. Po vytvoření nových jedinců jsou noví jedinci sloučeni se současnou populací a následně setříděni podle vhodnosti, pak je opět 60% populace odstraněno, tzn. odstranění nejslabší části populace (Kód 6).

```
...
    for(int q = 0; q < populSize; q++)
        temp[q] = popul[q];

    for(int q = populSize; q < populSize + newPopulSize; q++)
        temp[q] = newPopul[q − populSize];

    this−>sort(temp, 0, populSize + newPopulSize−1);

    for(int q = 0; q < populSize; q++)
        popul[q] = temp[q];
...
```

Kód 6: Výběr jedinců pro další populaci a nahrazování jedinců

10 Sériový model GA

Operace výběru jedinců, křížení, mutace a nahrazování populace jsou prováděny sériově. Sériový GA (Kód 7) je implementován především pro porovnání výsledného nejlepšího řešení a výpočetního času mezi jednotlivými modely.

```
...
    for(int i = 0; i < tempEvol; i++)
    {
        for(int j = 1; j < populSize; j++)
        {
            if (popul[j-1].getFitness() == popul[j].getFitness())
                if (popul[j-1] == popul[j])
                {
                    popul[j].mutate(seed);
                    popul[j].calcFitness( map );
                }
        }

        for(int j = 0; j < newPopulSize; j++)
        {
            int a = this->ruletteWheel(0, populSize-1);
            int b = this->ruletteWheel(0, populSize-1);
            while(a == b) b = this->ruletteWheel(0, populSize-1);

            int* indiv = crossover( popul[a], popul[b] );
            Chrom c(indiv, &chromSize);
            c.calcFitness( map );
            newPopul[j] = c;
        }

        for(int q = 0; q < populSize; q++)
            temp[q] = popul[q];

        for(int q = populSize; q < populSize + newPopulSize; q++)
            temp[q] = newPopul[q - populSize];

        this->sort(temp, 0, populSize + newPopulSize-1);

        for(int q = 0; q < populSize; q++)
            popul[q] = temp[q];

        solution[i] = popul[0].getFitness();
        if (output) printf ("%d. evolution ended.\n", i+1);
    }
...
```

Kód 7: Průběh sériového genetického algoritmu

11 Master-Slave model GA

Vytváření nových jedinců, selekce a nahrazování populace jsou prováděny paralelně, operace mutace se provádí sériově z důvodu nebezpečí problému souběhu. Master-Slave algoritmus (Kód 8) narozdíl od sériového algoritmu zpracovává několik jedinců najednou. Nevýhodou přístupu je možný vznik problému souběhu. Problém souběhu spočívá v neurčitosti stavu proměnné vlivem přepisování proměnné více než jedním procesem.

```
...
   for(int i = 0; i < tempEvol; i++)
   {
      for(int j = 1; j < populSize; j++)
      {
         if (popul[j-1].getFitness() == popul[j].getFitness())
            if (popul[j-1] == popul[j])
            {
               popul[j].mutate(seed[nworkers]);
               popul[j].calcFitness( map );
            }
      }

       cilk_for (int j = 0; j < newPopulSize; j++)
      {
         int a = this->ruletteWheel(0, populSize-1);
         int b = this->ruletteWheel(0, populSize-1);
         while(a == b) b = this->ruletteWheel(0, populSize-1);

         int* indiv = crossover( popul[a], popul[b] );
         Chrom c(indiv, &chromSize);
         c.calcFitness( map );
         newPopul[j] = c;
      }

       cilk_for (int q = 0; q < populSize; q++)
         temp[q] = popul[q];

       cilk_for (int q = populSize; q < populSize + newPopulSize; q++)
         temp[q] = newPopul[q - populSize];

      this->sort(temp, 0, populSize + newPopulSize-1);

       cilk_for (int q = 0; q < populSize; q++)
         popul[q] = temp[q];

      solution[i] = popul[0].getFitness();
      if (output)  printf ("%d. evolution ended.\n", i+1);
   }
...
```

Kód 8: Průběh paralelního genetického algoritmu (Master-Slave model)

12 Island model GA

Island model spouští několik instancí sériového GA s občasnou výměnou nejlepších jedinců mezi instancemi. Výměna jedinců probíhá podle parametru četnosti migrace.

```
...
  for(int i = 0; i < nEvol;i++)
  {
     cilk_for (int j = 0; j < this->nworkers; j++)
     {
        island[j]->run(nMigration);
        for(int k = 0; k < nMigration; k++)
           memcpy((solution[j])+index, island[j]->getSolution(), sizeof(float) * nMigration);
     }
     index += nMigration;

     for(int q = 0; q < nworkers; q++)
     {
        Chrom* populA = island[q]->getPopulation();
        Chrom* populB = island[travelMap[q]]->getPopulation();
        for(int k = 0; k < migrateSize; k++)
        {
           Chrom c;
           c = island[q]->getPopulation()[k];
           populA[k] = populB[k];
           populB[k] = c;
        }
     }
     if (output)  printf ("%d. evolution ended.\n", index);
  }

  if (rest)
     cilk_for (int j = 0; j < this->nworkers; j++)
     {
        island[j]->run(rest);
        for(int k = 0; k < nMigration; k++)
           memcpy((solution[j])+index, island[j]->getSolution(), sizeof(float) * rest);
     }
...
```

Kód 9: Průběh paralelního genetického algoritmu (Island model)

13 Testování

Testování modelů proběhlo na třech na sobě nezávislých problémech z knihovny TSPLIB[2] vzorových příkladů problému TSP. Byly vybrány tři problémy - kroA100 se 100 vrcholy, d2103 s 2103 vrcholy a rl5915 s 5915 vrcholy. Tyto příklady byly testovány s parametry:

- **velikost populace** udává kolik jediců tvoří populaci,
- **počet evolucí** udává kolika evolucí projde populace jedinců,
- **četnost migrace** udává jak často budou jedinci migrovat,
- **velikost migrující populace** - velikost migrující populace byla stanovena na pět jedinců (polovina minimální velikosti populace).

Výsledky testů jsou zobrazeny ve formě tabulek (Tab. 2, 3, 4, 5, 6, 7, 8, 9, 10), kde jsou uvedeny vstupní parametry: velikost populace, počet evolucí, četnost migrace a parametry výstupní: nalezené řešení, odchylka, čas a zrychlení.

- **nalezené řešení** udává kvalitu nejlepšího jedince v poslední generaci,
- **odchylka** udává procentuální rozdíl kvality řešení mezi kvalitou nejlepšího jedince a optimálním řešením problému,
- **čas** udává jak dlouho trval běh programu v sekundách,
- **zrychlení** udává zrychlení paralelního algoritmu podle:

$$zrychlení = \frac{Čas_{sériová\_implementace}}{Čas_{paralelní\_implementace}}$$

Jednotlivé testy proběhly na sobě nezávisle desetkrát a do tabulek byly zapsány aritmetické průměry naměřených hodnot.

[2] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

kroA100				
model	Sériový			
opt. řešení	21282			
velikost populace	počet evolucí	čas [s]	nalezené řešení	odchylka [%]
40	1	0.000809	62423	193.3
40	5	002881	24348	14.4
40	100	0.017265	23183	8.9
40	500	0.066628	22860	7.4
40	1000	0.125307	22635	6.4
100	1	0.001820	60004	181.9
100	5	0.008000	23870	12.2
100	100	0.054431	22825	7.3
100	500	0.233979	22672	6.5
100	1000	0.493943	22418	5.3
250	1	0.005324	58955	177.0
250	5	0.021852	23739	11.5
250	100	0.142826	22660	6.5
250	500	0.667834	22350	5.0
250	1000	1.289720	22156	4.1
500	1	0.009572	57833	171.7
500	5	0.048112	23373	9.8
500	100	0.308244	22609	6.2
500	500	1.110872	22216	4.4
500	1000	2.259791	21974	3.3

Tabulka 2: Testy sériového modelu provedené pro kroA100

d2103				
model	Sériový			
opt. řešení	80450			
velikost populace	počet evolucí	čas [s]	nalezené řešení	odchylka [%]
40	1	0.131544	1050703	1206.0
40	5	0.580450	87931	9.3
40	100	2.690647	84900	5.5
40	500	5.780171	84872	5.5
40	1000	11.020681	84786	5.4
100	1	0.336959	1041452	1194.5
100	5	1.495703	86616	7.7
100	100	10.569761	84580	5.1
100	500	15.976181	84461	5.0
100	1000	26.808895	84505	5.0
250	1	0.830118	1032356	1183.2
250	5	4.327034	86276	7.2
250	100	24.254995	84294	4.8
250	500	49.354542	84123	4.6
250	1000	71.176109	84257	4.7
500	1	1.678159	1026375	1175.8
500	5	7.373616	85947	7.3
500	100	59.29393	83964	4.4
500	500	98.925522	83920	4.3
500	1000	149.330704	84062	4.5

Tabulka 3: Testy sériového modelu provedené pro d2103

rl5915				
model	Sériový			
opt. řešení	565530			
velikost populace	počet evolucí	čas [s]	nalezené řešení	odchylka [%]
40	1	1.730189	13304376	2252.5
40	5	6.836732	680117	20.3
40	100	51.308697	662706	17.2
40	500	67.667793	655795	16.0
40	1000	93.883896	655862	16.0
100	1	3.932068	13172240	2229.2
100	5	18.922842	678807	20.0
100	100	144.050552	658737	16.5
100	500	253.442291	652105	15.3
100	1000	272.829376	650745	15.1
250	1	9.395555	13134155	2222.5
250	5	42.328911	679326	20.1
250	100	342.755402	659179	16.6
250	500	665.814819	646626	14.3
250	1000	759.494873	647538	14.5
500	1	18.607746	13092309	2215.1
500	5	73.641754	676062	19.5
500	100	679.215027	658199	16.4
500	500	1509.345947	645302	14.1
500	1000	1729.942246	645969	14.2

Tabulka 4: Testy sériového modelu provedené pro rl5915

kroA100					
model	Master-Slave				
opt. řešení	21282				
velikost populace	počet evolucí	čas [s]	zrychlení	nalezené řešení	odchylka [%]
40	1	0.001062	0.8	59590	180.0
40	5	0.001886	1.5	24234	13.9
40	100	0.019348	0.9	22646	6.4
40	500	0.081532	0.8	22101	3.8
40	1000	0.171545	0.7	22002	3.4
100	1	0.000934	1.9	58604	175.4
100	5	0.004062	2.0	23827	12.0
100	100	0.041600	1.3	22515	5.8
100	500	0.190879	1.2	22042	3.6
100	1000	0.361685	1.4	21980	3.3
250	1	0.002127	2.5	57843	171.8
250	5	0.009674	2.3	23648	11.1
250	100	0.100196	1.4	22392	5.2
250	500	0.439313	1.5	22030	3.5
250	1000	0.860434	1.5	21860	2.7
500	1	0.003984	2.4	55823	162.3
500	5	0.018897	2.5	23293	9.4
500	100	0.195157	1.6	22376	5.1
500	500	0.868010	1.3	22037	3.5
500	1000	1.705216	1.3	21852	2.7

Tabulka 5: Testy Master-Slave modelu provedené pro kroA100

d2103					
model	Master-Slave				
opt. řešení	80450				
velikost populace	počet evolucí	čas [s]	zrychlení	nalezené řešení	odchylka [%]
40	1	0.058111	2.3	1048490	1203.3
40	5	0.262374	2.2	87775	9.1
40	100	1.415255	2.0	84796	5.4
40	500	2.707411	2.1	84875	5.5
40	1000	4.413104	2.5	84522	5.1
100	1	0.157386	2.1	1035933	1187.7
100	5	0.718918	2.1	86772	7.9
100	100	3.987010	2.7	84489	5.0
100	500	6.753477	2.4	84324	4.8
100	1000	10.912425	2.5	84133	4.6
250	1	0.377907	2.2	1026820	1176.3
250	5	1.795707	2.4	86278	7.2
250	100	11.627910	2.1	84139	4.6
250	500	20.030087	2.5	83956	4.4
250	1000	28.829977	2.5	83770	4.1
500	1	0.756236	2.2	1022875	1171.4
500	5	3.552372	2.1	85878	6.7
500	100	26.190100	2.3	84054	4.5
500	500	41.530231	2.4	83909	4.3
500	1000	62.172157	2.4	83497	3.8

Tabulka 6: Testy Master-Slave modelu provedené pro d2103

rl5915					
model	Master-Slave				
opt. řešení	565530				
velikost populace	počet evolucí	čas [s]	zrychlení	nalezené řešení	odchylka [%]
40	1	0.714197	2.4	13281738	2248.5
40	5	3.145887	2.2	684203	21.0
40	100	22.465605	2.3	659698	16.7
40	500	29.177896	2.3	656456	16.1
40	1000	37.998417	2.5	656006	16.0
100	1	1.797174	2.2	13271288	2246.7
100	5	7.804014	2.4	678772	20.0
100	100	67.733009	2.1	659280	16.6
100	500	115.457115	2.2	650906	15.1
100	1000	118.267197	2.3	646391	14.3
250	1	4.208253	2.2	13136659	2222.9
250	5	18.613026	2.3	677995	19.9
250	100	172.576187	2.0	659167	16.6
250	500	309.688934	2.1	647663	14.5
250	1000	419.714447	1.8	642727	13.7
500	1	7.7669596	2.4	13064804	2210.2
500	5	36.149044	2.0	674625	19.3
500	100	342.979218	2.0	658344	16.4
500	500	748.227661	2.0	643681	13.8
500	1000	856.967651	2.0	641061	13.4

Tabulka 7: Testy Master-Slave modelu provedené pro rl5915

kroA100						
model	Island					
opt. řešení	21282					
velikost populace	počet evolucí	četnost migrace	čas [s]	zrychlení	nalezené řešení	odchylka [%]
40	100	20	0.016471	1.0	23026	8.2
40	500	20	0.062577	1.1	23026	8.2
40	1000	20	0.118114	1.1	22857	7.4
100	100	20	0.033883	1.6	22816	7.2
100	500	20	0.144864	1.6	22752	6.9
100	1000	20	0.25965	1.9	22683	6.6
250	100	20	0.082369	1.7	22763	7.0
250	500	20	0.335155	2.0	22413	5.3
250	1000	20	0.649260	2.0	22689	6.6
500	100	20	0.156988	2.0	22843	7.3
500	500	20	0.694249	1.6	22132	4.0
500	1000	20	1.362737	1.7	22033	3.5
40	100	40	0.017624	1.0	22458	5.5
40	500	40	0.058168	1.1	22458	5.5
40	1000	40	0.102089	1.2	22458	5.5
100	100	40	0.034012	1.6	22987	8.0
100	500	40	0.143304	1.6	22845	7.3
100	1000	40	0.283741	1.7	22744	6.9
250	100	40	0.077538	1.8	22744	6.9
250	500	40	0.333779	2.0	22707	6.7
250	1000	40	0.705223	1.8	22382	5.2
500	100	40	0.154631	2.0	22858	7.4
500	500	40	0.693486	1.6	22539	5.9
500	1000	40	1.338829	1.7	21906	2.9

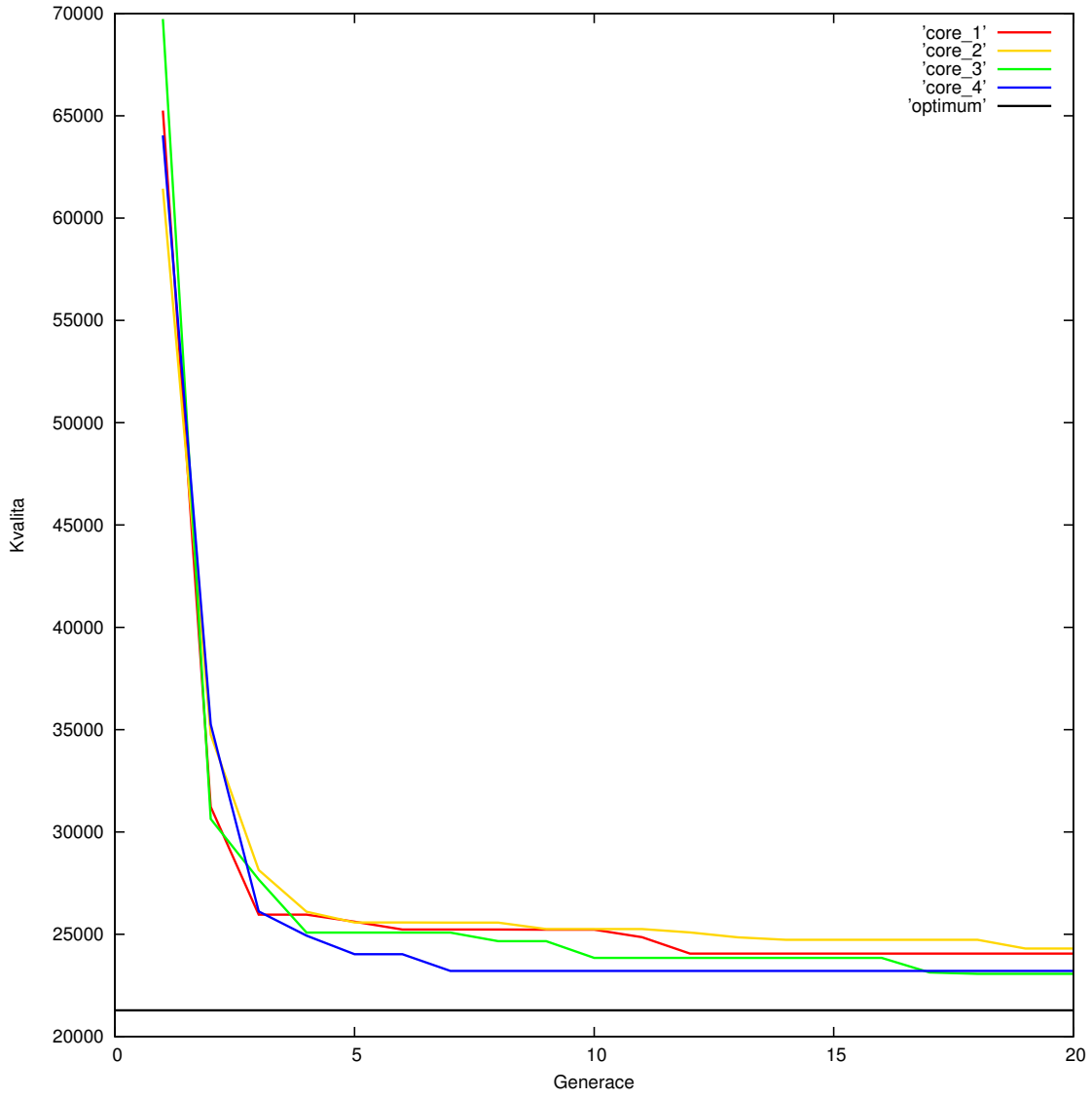
Tabulka 8: Testy Island modelu provedené pro kroA100

d2103						
model	Island					
opt. řešení	80450					
velikost populace	počet evolucí	četnost migrace	čas [s]	zrychlení	nalezené řešení	odchylka [%]
40	100	20	2.046422	1.3	85181	5.9
40	500	20	5.969123	1.0	85090	5.8
40	1000	20	5.179857	2.1	84764	5.4
100	100	20	4.011249	2.6	84507	5
100	500	20	8.503023	1.9	84388	4.9
100	1000	20	12.211344	2.2	84402	4.9
250	100	20	13.942272	1.7	84540	5.1
250	500	20	19.165918	2.6	84907	5.5
250	1000	20	29.852114	2.4	84377	4.9
500	100	20	22.786779	2.6	84570	5.1
500	500	20	53.362679	1.9	84005	4.4
500	1000	20	63.020695	2.4	84199	4.7
40	100	40	1.695576	1.6	85174	5.9
40	500	40	9.134838	0.6	85432	6.2
40	1000	40	6.305458	1.7	84051	4.5
100	100	40	4.450207	2.4	84970	5.6
100	500	40	6.930356	2.3	85298	6
100	1000	40	12.348001	2.2	84591	5.1
250	100	40	8.661070	2.8	84988	5.6
250	500	40	22.441833	2.2	84157	4.6
250	1000	40	31.083084	2.3	83915	4.3
500	100	40	20.257805	2.9	84472	5.0
500	500	40	39.131798	2.5	84346	4.8
500	1000	40	57.227074	2.6	84271	4.7

Tabulka 9: Testy Island modelu provedené pro d2103

rl5915						
model	Island					
opt. řešení	565530					
velikost populace	počet evolucí	četnost migrace	čas [s]	zrychlení	nalezené řešení	odchylka [%]
40	100	20	21.806271	2.4	670652	18.6
40	500	20	36.474152	1.9	666727	17.9
40	1000	20	132.161606	0.7	656377	16.1
100	100	20	67.409531	2.1	655636	15.9
100	500	20	91.314590	2.8	651531	15.2
100	1000	20	105.943825	2.6	655110	15.8
250	100	20	137.583221	2.5	654208	15.7
250	500	20	199.554184	3.3	651792	15.3
250	1000	20	293.962921	2.6	645618	14.2
500	100	20	367.437927	1.8	662361	17.1
500	500	20	657.422546	2.3	650264	15.0
500	1000	20	967.549072	1.8	651591	15.2
40	100	40	21.157806	2.4	664091	17.4
40	500	40	76.326752	0.9	666647	17.9
40	1000	40	105.294853	0.9	666650	17.9
100	100	40	54.853306	2.6	651770	15.2
100	500	40	104.371895	2.4	654576	15.7
100	1000	40	102.745094	2.7	651560	15.2
250	100	40	167.891739	2.0	659081	16.5
250	500	40	333.387695	2.0	651447	15.2
250	1000	40	302.998535	2.5	654258	15.7
500	100	40	366.755066	2.0	655699	15.9
500	500	40	761.551270	2.0	653170	15.5
500	1000	40	716.512695	2.4	643002	13.7

Tabulka 10: Testy Island modelu provedené pro rl5915



Obr. 11: Vývoj dvaceti generací problému kroA100 Island modelu

14 Závěr

Cílem bakalářské práce bylo vytvořit základní přehled sériových a paralelních genetických algoritmů, navrhnout projekt a implementovat paralelní verzi algoritmu pomocí technologie Cilk++, včetně jeho testování.

V teoretické části jsem zpracoval přehled o základních charakteristikách sériových a paralelních genetických algoritmů používaných v informačních technologiích. Nedílnou součásti teoretické části je seznámení s technologií Cilk++, včetně vývoje projektu, výpočetního modelu, hlavních charakteristik jazyka Cilk a možnostmi implementace GA algoritmů za použití této technologie.

V projektové části jsem řešil Problém obchodního cestujícího, kde jsem navrhl a implementoval tři verze genetického algoritmu (sériový model GA, Master-Slave model GA a Island model GA). Algoritmy byly spuštěny s různými parametry, tj. velikost populace, počet evolucí a četnost migrace a s různými vstupními soubory (kroA100 - 100 vrcholů, d2103 - 2103 vrcholů, rl5915 - 5915 vrcholů) nad různými TSP problémy.

Výsledky, které jsou přehledně uvedeny v tabulkách (Tab. 2, 3, 4, 5, 6, 7) a grafu (Obr. 11) ukazují na velmi rychlý vývoj jedinců při prvních pěti evolucí, další evoluce je značně pomalejší s občasným zlepšením jedinců. Při zvyšování parametru počtu evolucí a velikosti populace se konečné řešení zlepšovalo, ale zvyšoval se i výpočetní čas potřebný k vyřešení problému. Hlavním důvodem nárustu výpočetního času bylo zvyšování velikosti populace, vlivem zvýšení velikosti populace jedinci konvergovali pomaleji a tím bylo nalezeno lepší řešení.

U Master-Slave modelu došlo k průměrnému zrychlení, u menšího problému kroA100 1.5 (min. 0.7 - max. 2.5), u středního problému d2103 2.3 (min. 2.0 - max. 2.7) a u velkého problému rl5915 2.2 (min. 1.8 - max. 2.5). Kvalita řešení u Master-Slave modelu a sériového modelu se liší vlivem stochastické části GA, kvalita řešení by měla být stejná nebo velmi podobná řešení sériového GA. U modelu Island dochází k průměrnému zrychlení, u menšího problému kroA100 1.6 (min. 1.0 - max. 2.0), u středního problému d2103 2.1 (min. 0.6 - max. 2.9) a u velkého problému rl5915 2.1 (min. 0.7 - max. 2.8) a kvalita řešení se oproti Master-Slave modelu příliš neliší. Výhoda Island modelu ve srovnání Master-Slave modelem je vývoj několika nezávislých populací, které na sebe mohou, ale nemusí být konvergentní.

Testování ukázalo, že navržené algoritmy ukázaly u menších instancí najít řešení velmi blízké globálnímu optimu, přičemž s rostoucí složitostí instancí kvalita řešení klesá, a to vlivem většího prohledávaného prostoru.

Paralelizace modelů pomocí Cilk++ byla u modelu Master-Slave provedena u třídícího algoritmu přidáním klíčového slova cilk_spawn a cilk_sync a jednoduchým přepsáním klíčových slov for na cilk_for. U větších projektů, kde přepsání klíčových slov for není optimální pro vysoký počet klíčových slov for ve zdrojovém kódu, je možné re-definovat klíčové slovo for, ale je potřeba dávat pozor na kritické body programu (nebezpečí souběhu) a upravit tyto části tak, aby byla zaručena stabilita programu. Paralelizace Island modelu byla implementována pomocí instancí sériového GA, pro každé jádro procesoru jedna instance, kde byly průběhy evoluce instancí sériových GA přerušeny podle parametru četnosti migrace s následnou migrací.

Rozšíření jazyka C++ jazykem Cilk++ podporuje datový i funkční paralelismus pro procesory z rodiny Intel Xenon®Phi™.

15 Reference

[1] AFFENZELLER, Michael, WINKLER, Sephan, WAGNER Stefan, BEHAM, Andreas. *Genetic Algorithms and Genetinic Programming* Modern Concepts and Practical Applications. Boca Raton, Fla.: Chapman & Hall/CRC Press, 2009. 2-21. ISBN 978-1-58488-629-7

[2] AHMED, Zakir H. *Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator*. International Journal of Biometrics & Bioinformatics (IJBB). 2010. 96-105.

[3] HAUPT, Daniel. *Paralelizace genetických algoritmů*. Diplomová práce. Brno. Vysoké učení technické v Brně. 2011. 9-36.

[4] POŠÍK, Petr. *Paralelní genetické algoritmy*. Diplomová práce. Praha. České vysoké učení technické v Praze. 2001. 2-9, 14-17.

[5] POŠÍK, Petr. *Paralelní genetické algoritmy*. cyber.felk.cvut.cz [online]. 20. 2. 2000. [cit 15. 1. 2013]. Dostupné z: http://labe.felk.cvut.cz/~posik/pga/theory/pga-theory.htm

[6] ZELINKA, Ivan, OPLATKOVÁ, Zuzana, ŠEDA, Miloš, OŠMERA, Pavel, VČELAŘ František. *Evoluční výpočetní techniky* Principy a aplikace. Praha: BEN - Technická literatura, 2009. 26-28, 173-187. ISBN 978-80-7300-218-3

[7] OBITKO, Marek. *Introduction to genetic algorithms*. www.obitko.com [online]. 1998. [cit. 16. 12. 2012]. Dostupné z: http://www.obitko.com/tutorials/genetic-algorithms/index.php

[8] *A Brief History of Cilk* [online]. [cit. 10. 4. 2013]. Dostupné z: cilkplus.org/cilk-history

[9] *Cilk Plus Tutorial* [online]. [cit. 22. 2. 2013]. Dostupné z: cilkplus.org/cilk-plus-tutorial

[10] LUNER, Petr. *Jemný úvod do genetických algoritmů*. cgg.mff.cuni.cz [online]. [cit. 15. 1. 2013]. Dostupné z: http://cgg.mff.cuni.cz/~pepca/prg022/luner.html

[11] *Intel® C++ Compiler XE 13.1 User and Reference Guides*. [online] [cit. 12. 3. 2013]. Dostupné z: http://software.intel.com/sites/products/documentation/doclib/stdxe/2013/composerxe/compiler/cpp-win/index.htm

[12] *Intel® Cilk++ SDK Programmer's Guide*. [online]. [cit. 12. 3. 2013]Dostupné z: www.clear.rice.edu/comp422/resources/Intel_Cilk++_Programmers_Guide.pdf. 31-68.

[13] *www.algoritmy.net*. [online]. [cit. 12. 3. 2013] Dostupné z: http://www.algoritmy.net/article/5407/Obchodni-cestujici.

A Obsah přiloženého CD

- text bakalářské práce
- zdrojové kódy
- testované TSP problémy kroA100, d2103 a rl5915
- výstupní soubory testů