Изследване на паралелизирани метаевристични алгоритми върху проблема за оцветяване на граф

Comparing parallellized metaheuristic algorithms solving the graph coloring problem

Иван Даниелов Иванов
Ivan Danielov Ivanov
Имейл: ivan@vankata.tk
Американски колеж
София, 11-ти клас

Милен Младенов Ферев
Milen Mladenov Ferev
Имейл: m.ferev18@gmail.com
Американски колеж
София, 11-ти клас

Под ръководството на

Др. Стоян Велев
Мениджър развитие в SAP Labs Bulgaria, София
Имейл: stoyan.vellev@sap.com

Декември 2016

Резюме

С нарастването на сложността на алгоритмите, нуждата за изчислителни ресурси се увеличава. Класическият модел на последователно изчисление не може да предостави нужната скорост за решаване на проблемите. Решение е предоставено от модела на паралелизирането изчисляване. В проекта "Изследване на паралелизирани метаевристични алгоритми върху проблема за оцветяване на граф"познатата NP-трудна задача за оцветяване на граф е решена, използвайки метаевристични алгоритми. Разработката включва генетичен алгоритъм и алгоритъм за оптимизация на мравешка колония. Те се основават на съществуващи алгоритми, но за разлика от авторските разработки, алгоритмите в проекта са паралелизирани и оптимизирани за работа на система за паралелно изчисление. За паралелизирането на алгоритмите се използва стандартният Message Passing Interface (MPI). Втората фаза на проекта е тестването на разработените алгоритми използвайки стандартизирани тестове от центъра по дискретна математика и теоритична компютърна наука (DIMACS) и представяне на резултатите. За по-реално тестване е избрана едночиповата система Parallella.

Summary

With the growing complexity of algorithms the need for computing resources also increases. The classical model of serial processing can not provide the speed needed for solving the problems. A solution is the parallel processing model. In the project "Exploring parallel metaheuristic algorithms on the graph coloring problem"the known NP-hard graph coloring task is solved, using metaheuristic algorithms. The project includes a genetic algorithm and an ant colony optimization algorithm. They are based on existing algorithms, but in comparison to the authors', the algorithms in the project are parallelized and optimized to work on a distributed computing system. For the parallelization the standard Message Passing Interface (MPI) is used. The second phase of the project is testing the algorithms using standardized tests from the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) and presenting the results.

Съдържание

1 Въведение

Идеята за паралелни изчисления се появява през 1954. Според нея една задача може се реши като се раздели на много по-малки подзадачи, които могат да бъдат решени по едно и също време вместо последователно. В сравнение с последователния изчислителен модел, паралелният модел позволява по-бързо решаване на задачата. За да може даден алгоритъм да бъде изпълнен на система за паралелно изчисление, той трябва да е моделиран така, че да може да се раздели на различни процеси, които да бъдат изпълнени едновременно.

Целта на тази научна разработка е да изследва различни метаевристични алгоритми, сравнявайки ефективността им при решаване на задачата за оцветяване на граф на паралелна изчислителна система. Задачата изисква граф да бъде оцветен чрез минимален брой цветове, така че да няма съседни върхове с еднакъв цвят. Разработки като генетичен алгоритъм и алгоритъм за колония на мравките са премоделирани, оптимизирани, тествани и класирани един срещу друг.

Генетичният алгоритъм имитира процеса на естествения подбор, което му позволява да намери решение на задачи като проблемът за оцветяване на граф.Тестваният алгоритъм е базиран на публикацията на *Реза Абасиан* и *Малек Моухуб* [1]. Той използва модела на главни и второстепенни процеси, характерен с това, че един от процесите разпределя задачите на другите процеси и управлява обмяната на информация между тях. По този начин изчислителното време се намалява, защото задачите са изпълнени едновременно.

Алгоритъмът за оптимизация на мравешка колония имитира поведението на реална мравешка колония. Мравките работят заедно, за да намерят източник на храна, като всяка от тях оставя следа от хормони, наречени феромони, след себе си. Когато някоя мравка намери такъв, тя се връща обратно в мравуняка, оставя храната и се отива за още. По този начин, феромоните се наслагват върху най-кратките пътища и мравките започват да използват само тях. Подобно на това, в нашето проучване различни процеси работят върху оцветяването на графа и когато се намери решение, всички процеси започват да търсят дали има по-добро решение, като се опитват да се приближат до предишното намерено.

Комуникацията между процесите се установява чрез интерфейса за предаване на съобщения (Message Passing Interface). Чрез него се обособява моделът на главни и второстепенни процеси, който се използва в текущата разработка. Интерфейсът позволява комуникация и предаване на информация между работещите процеси в рамките на локална мрежа.

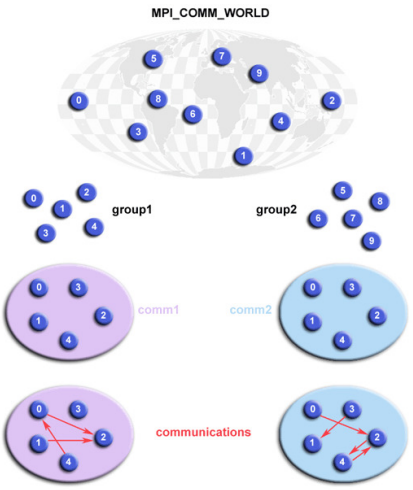
Изложението в текущата разработка е разделено на пет секции: **Интерфейс за предаване на съобщения MPI**, **Подход с генетичен алгоритъм**, **Подход с алгоритъм за оптимизация на мравешка колония**, **Илюстратор**, **Тестове и резултати**, където в детайли ще обсъдим интефейса за предаване на съобщения, използвания генетичен алгоритъм, използвания алгоритъм за оптимизация на мравешка колония, разработения визуализатор, и ще представим тестовете и резултати, последвани от кратък анализ.

2 Интерфейс за предаване на съобщения MPI

В тази секция ще бъде обяснен интерфейсът за предаване на съобщения, който е стандартът използван за паралелизиране на алгоритмите.

2.1 Описание

Интерфейсът за предаване на съобщения е система, превърнала се в стандарт, която се използва за паралелни изчисления. Разработката ѝ започва през 1991. Тя предоставя защитен TCP-базиран протокол за комуникация между процесите. Комуникацията се установява чрез протокола SSH, който трябва да бъде инсталиран на изчислителната машина. Главното предимство на интерфейса е, че е независим от архитектурата на процесора, което позволява процесори с различни архитектури да работят заедно в една хетерогенна система. Това беше проблем, срешнат при разработката на проекта и е обяснен в секцията **Тестове и резултати**. Интерфейсът организира паралелното изчисляване като разпределя различни индентификационни номера на работещите процеси. Тези номера се използват за комуникация с процеса със съответния номер. Интерфейсът предоставя предавателен механизъм, който подрежда процесите в топология. Този механизъм може да раздели изчислителната система на по-малки групи, всяка от които има зададена задача и резултатите се обединяват, за да се формира крайния резултат, както е показано на фигура 1.



Фигура 1: Разделението на системата в по-малки групи

2.2 Имплементация

Първоначално интерфейсът за предаване на съобщения се разработва за езиците C, C++ и Fortran, които се поддържат от повечето имплементации. Първата имплементация е MPICH, която използва стандарт MPI 1.x. MPICH2 е втората версия, която използва стандарт MPI 2.x и има модификации като едностранна комуникация. Някои имплементации като MPICH не поддържат разнородни системи. Стандартът е ефективен за еднородна система, но разнородното ограничение прави тази имплементация не толкова добра за разработката. Сегашната разработка се използва OpenMPI. OpenMPI е подобна на MPICH, но поддържа разнородни системи и се използва в много от най-добрите суперкомпютри [11]. Има други имплементации на MPI стандарта, като тези на IBM и Intel, но техните дистрибуции са ориентирани към търговци и нямат важни предимства пред OpenMPI. Поради тези причини в текущата научна раз-

Фигура 2: Логото на OpenMPI

работка се използва OpenMPI. Следователно, вариантите за програмен език са Fortran, C и C++. Разликата в производителността на тези езици е малка, но C++ предоставя много повече удобства със C++11 и C++17 стандартите. Поради тези причини, C++ се използва като програмен език в текущата разработка. Освен основните имплементаци на стандарта MPI има имплементации за Java и Python, но те не са езици на ниско ниво и времето им за изпълнение е по-голямо. Това ги прави лош избор за разработката, защото те биха увеличили времето за изпълнение на алгоритмите, което ще има ефект върху крайните резултати.

3 Подход с генетичен алгоритъм

В тази част ще бъде описан използваният генетичен алгоритъм и ще бъдат обяснени направените по него оптимизации.

3.1 Подходът на Абасиан и Моухуб

Алгоритъмът в проекта е базиран на този, описан от Абасиан и Моухуб в тяхната публикация [1]. Те предлагат решение на проблема за оцветяване на граф, използвайки генетичен алгоритъм.

3.1.1 Архитектура

Използваната за паралелизиране архитектура е комбинция между главни и второстепенни процеси и островен модел на паралелен генетичен алгоритъм. Според този дизайн всеки процес трябва да работи върху собствена популация. Този модел е ефективен в системи с голям брой изчисляващи ядра, защото много популации могат да бъдат обработвани едновременно, което увеличава шанса да бъде намерено решение.

3.1.2 Търсене на решение

Авторите предлагат решението да се търси, като първоначално се стартира приближаващ модул, който определя горна граница за брой използвани цветове K. По-нататък всеки остров от архитектурата работи върху намиране на решение с по-малък брой от намерения от приближаващия модул. Тоест, първия остров търси решение с K-1 цветове, втория с K-2 и т.н. Ако успее да намери такова, то това става новата горна граница и търсенето започва отначало с новото K. Ако не успее, то K е минималния отговор, намереното оцветяване с K се записва във файл и алгоритъмът свършва своята работа. Това число K е също известно като хроматичното число на съответния граф.

3.1.3 Хромозом

Общата популация в алгоритъма се състои от различни хромозоми. Хромозомът е представен като масив, съдържащ схема за оцветяване на върховете на графа. Всеки хромозом има своя жизнеспособност, която зависи от броя на конфликтните върхове в съответния хромозом. Хромозом преминават през мутация, размножаване и някои от тях са заменени от нови генетично модифицирани индивиди.

3.1.4 Първоначална популация

Популацията се генерира при всяка проверка за решение. Тя се състои от индивиди, всеки от които представя възможно оцветяване за графа.

3.1.5 Размножаване и кръстосване

За размножителната фаза в алгоритъма се избират произволни 30% от най-здравите индивиди и произволни 10% от цялата популация. Кръстосването става, когато за два индивида от избраните се генерират и произволен брой точки на кръстосване. Гените от двата родителя, разделени от точките на кръстосване, се пренасят в новия елемент. След като новият индивид се генерира, той замества произволно избран индивид от популацията.

3.1.6 Мутация

В алгоритъма има два метода за мутация. При първия метод произволно се променя оцветяването на индивида. При втория метод индивидът се променя така, че конфликтните върхове са сведени до минимум. Това означава, че ако бъде намерен конфликт, един от цветовете в конфликтната двойка върхове се заменя с произволен цвят. В разработката на алгоритъма всеки индивид има еднакъв шанс за мутация. На всяка генерация 20% от популацията е избрана да мутира, като 10% от хромозома на всеки индивид е подложен на промяна. Шансът да се избере първата мутация е 33%, а втората 66%. По този начин голяма част от гените в популацията остават, но се въвеждат и нови гени.

3.1.7 Генетична модификация

Алгоритъмът включва генетичен модификатор, който създава нови индивиди за популацията. При стартиране модификаторът използва подредените по степен върхове и ги пренарежда по ниво на зависимост, което се определя от броя съседни върхове, имащи влияние над оцветяването на текущия връх. Зависимостта между два върха се определя от степента и изчисленото ниво на зависимост на всеки от двата върха. По този начин се решава дали връх А зависи от връх Б или връх Б - от връх А. След като върховете се подредят по ниво на зависимост модификаторът използва алчен алгоритъм, за да оцвети графа. Когато връх бъде оцветен с даден цвят, той се премахва от цветовото множество на съседните върхове. Възможно е някои върхове да останат неоцветени, при което модификаторът ги оцветява с произволно избран цвят.

3.1.8 Приближаващият модул

Приближаващият модул представлява алчен алгоритъм, много подобен на генетичния модификатор, който намира решение, не задължително оптимално, чиято цел е да определи крайна граница за списък от върховете на графа, подредени по степен. При инициализация модулът създава празен граф A, добавя върха с най-висока степен към A и го маркира като оцветен. След това модулът вмъква съседните върхове на добавения по-горе връх в A и използвайки алчен алгоритъм определя как да ги оцвети. Следвайки тази последователност от стъпки, приближаващият модул намира оцветяване на граф с нисък брой цветове K. Авторите изчисляват, че естиматорът има сложност от $O(V^2)$.

3.2 Модифициран подход

В тази част се покриват оптимизациите по всяка от главните функции на генетичния алгоритъм.

3.2.1 Архитектура

В текущата разработка архитектурата е променена само на главни и второстепенни процеси поради ограничените ресурси. Това означава, че главният процес има популацията, а второстепенните извършват операции като размножаване и мутация и връщат резултатите на главния процес. По този начин натоварването е намалено и позволява на алгоритъма да работи на по-слаби изчислителни машини.

3.2.2 Търсене на решение

Вместо предложеният от авторите приближаващ модул, имплементацията използва двоично търсене. Горната граница на двоичното търсене се установява използвайки теоремата на Брукс, която гласи, че хроматичното число на ненасочен свързан граф е най-високата степен Δ в графа. Ако графът представлява цикъл с нечетна дължина или е свързан, хроматичното число е равно на $\Delta+1$. Тъй като установяването на типа на графа отнема време, горната граница на двуичното търсене е зададена на $\Delta+1$. При голяма разлика между намерения от приближаващия модул отговор и крайния отговор, двоичното търсене ще е по ефективен подход в сравнение с приближаващия модул.

4 Подход с алгоритъм за оптимизация на мравешка колония

Алгоритъмът за оптимизация на мравешка колония работи следвайки модела на реална мравешка колония. Направен е по модел на алгоритъма ANTCOL, описан в публикацията на Д. Коста и А. Херц [6].

4.1 Следи от феромони

Характерно за алгоритмите на мравешка колония са следите от феромони. В природата, феромоните са хормони, които мравките оставят след себе си, когато минат по даден път. Те са много полезни в това да се види колко пъти дадена мравка е минавала през този път, което води до избор на текущата мравка къде да отиде в следващия ход.

В алгоритъма следите от феромони влият на вероятността даден връх да бъде оцветен с съответен цвят. При стартиране на алгоритъма се създава матрица в която се пази колко е било добро оцветяването когато два несъседни върха са били оцветени в един цвят. Първоначално всички стойности в матрицата са равни на едно, но впоследствие се променят.

$$M_{rs} := \rho \cdot M_{rs} + \Delta M_{rs} \; \forall [v_r, v_s] \notin E;$$

Фигура 3: Обновяване на матрицата с феромоните

4.2 Алгоритъмът ANTCOL

Алгоритъм ANTCOL служи за намиране на хроматичното число на дадения граф, използвайки следите от феромони, описани по-горе. При инициализация стойността на всяка следа от феромони става равна на едно, което маркира еднакво наличие на феромони навсякъде. На всеки цикъл на алгоритъма, графът се оцветява използвайки алчния алгоритъм Recursive Largest First. Ако бъде намерено оцветяване с по-малък брой цветове, той става текущия най-добър резултат. След това се определя промяната за всяка следа от феромони, която зависи от намереното решение. Във всяка клетка на матрицата с феромони с координати x и y се добавя реципрочното на намерения брой цветове, което означава, че при по-малко използвани цветове, толкова по-силна ще е следата. Освен това, на всеки ход, 0.5 процента от феромоните се изпаряват с цел много стари решения да не оказват влияние.

Table 2 The algorithm ANTCOL

$M_{rs} := 1\ \forall [v_r, v_s] \notin E$; (initialization of the trail between pairs of non adjacent vertices)
$f^\circ := \infty$; (number of colours in the best colouring reached so far)
For *ncycles* := 1 **to** $ncycles_{\max}$ **do**
  $\Delta M_{rs} := 0; \forall [v_r, v_s] \notin E$;
  For $a := 1$ **to** *nants* **do**
    colour the graph by means of a constructive method;
    let $s = (V_1, V_2, \ldots, V_q)$ be the colouring obtained
    If $q < f^\circ$ then $s^\circ := (V_1, \ldots, V_q); f^\circ := q$;
    $\Delta M_{rs} := \Delta M_{rs} + \frac{1}{q}\ \forall [v_r, v_s] \notin E, \{v_r, v_s\} \subseteq V_j, j = 1, \ldots, q$;
  $M_{rs} := \rho \cdot M_{rs} + \Delta M_{rs}\ \forall [v_r, v_s] \notin E$;

Фигура 4: Псевдокод на ANTCOL представен от авторите

4.3 Алгоритъмът Recursive Largest First (RLF)

Алгоритъмът RLF оцветява графа, групирайки върховете в зависимост от цвета, с който те могат да бъдат оцветени. При стартиране алгоритъмът избира връх v от множеството W, състоящо се от върхове, които могат да бъдат оцветени в цвят q. Избраният връх е включен в множеството V_q, състоящо се от върхове, които могат да се оцветят със цвят q. След това, докато монжестовто W не бъде изпразнено, съседите на върха v се включват в множеството B, състоящо се от върховете, които не могат да бъдат включени във V_q и се изключват от множеството W. След това се избира нов връх v от W, който се включва във V_q и горепосочените стъпки се повтарят. Когато множеството W се изпразни, елементите от B се включват в W и се премахват от B. След това се създава ново множество V_{q+1} и цикълът се повтаря докато не бъдат оцветени всички върхове.

Table 4 Procedure ANT_RLF (Σ, Ω) with $\Sigma \in \{1, 2\}$ and $\Omega \in \{1, 2, 3\}$

$q := 0$; (number of colours used)
$W := V$; (uncoloured vertices which can be included in the stable set under construction)
$k := 0$; (number of coloured vertices)
While $k < |V|$ **do**
  $k := k + 1$; $q := q + 1$;
  $B := \varnothing$; (uncoloured vertices which can no longer belong to the stable set V_q)
  select a first vertex v according to one of the two strategies Σ:
  Σ: $\begin{cases} (1)\ v = \arg\max\{\deg_W(v') | v' \in W\}; \\ (2)\ v$:chosen randomly in W;$ \end{cases}$

  $V_q := \{v\}$;
  While $W \backslash (N_W(v) \cup \{v\}) \neq \varnothing$ **do**
    $k := k + 1$;
    $B := B \cup N_W(v)$; $W := W \backslash (N_W(v) \cup \{v\})$;
    Let $s[k-1] = (V_1, \ldots, V_q)$ be the current partial solution. Choose $v \in W$ with probability $p_{it}(k, v)$ where $\tau_1(s[k-1], v) = \tau_2(s[k-1], v, q)$ and $\eta_1(s[k-1], v)$ is defined according to one of the three strategies Ω:
    Ω: $\begin{cases} (1)\ \eta_1(s[k-1], v) = \deg_B(v); \\ (2)\ \eta_1(s[k-1], v) = |W| - \deg_W(v); \\ (3)\ \eta_1(s[k-1], v) = \deg_{W \cup B}(v); \end{cases}$
    $V_q := V_q \cup \{v\}$;
  $W := B \cup N_W(v)$;

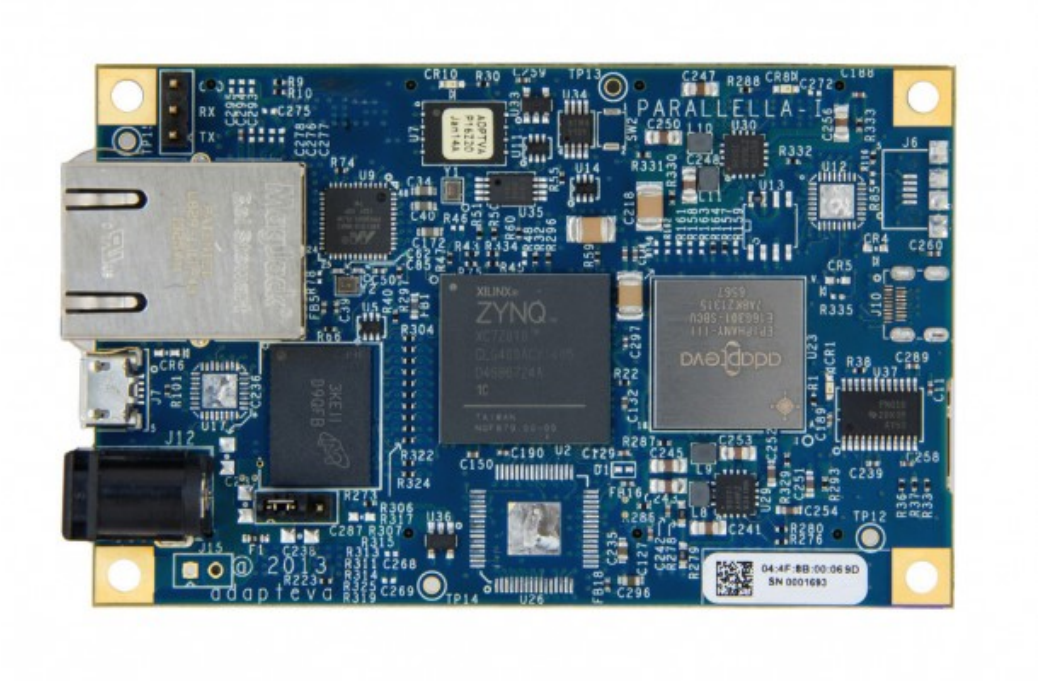
Фигура 5: Псевдокод на RLF, представен от авторите

4.4 Архитектура на паралелизиране

В алгоритъма е използвана архитектурата главен и второстепеннени процеси. На всеки ход на алгоритъма главният процес изпраща матрицата с феромонните следи на всички второстепенни. След това всеки второстепенен процес, имащ ролята на мравка, оцветява графа, използвайки алгоритъма Recursive Largest First (RLF). В края на всеки цикъл главният процес получава всички оцветявания и променя стойностите на следите от феромони.

5 Едночиповата система Parallella

С цел тестването на алгоритъмите на реална многоядрена система, сме предоставили и тестове върху едночиповата система Parallella. За момента за нея има имплементация единствено на алгоритъма за мравешка колония. За да е максимално представителен резултатът, направихме малки промени по алгоритъма, с което го пригодихме за оптимална работа на едночиповата система. Снимка на едночиповата система използвана в разработката е представена на 6.

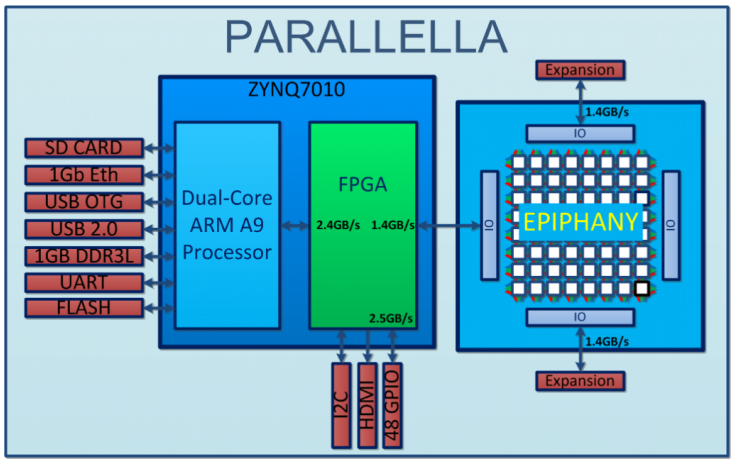


Фигура 6: Стандартния модел на едночиповата система с предназначение за сървър

5.1 Хардуер

Едночиповата система Parallella има да процесора. Първият е стандартен двуядрен ARMv7 Zunq процесор с ядро A9 и с честота на работа 800 Mhz. Отделно има и акселератор във формата на 16-ядрен Epiphany III RISC процесор с максимална производителност 32 GFLOPS. Двата процесора си комуникират посредством FPGA буфер. Използвания модел Parallella в разработката няма други функционалности като GPIO, HDMI и USB изходи. Схема на устройствотот на системата е представена

на фигура 7.



Фигура 7: Схема на начина на работа на едночиповата система

Системата разполага с 1GB DD3L RAM памет към основния процесор, от които 32 MB са споделени с Epiphany процесора по стандартната конфигурация. Отделно всяко ядро от Epiphany процесора разполага с 32 KB памет за изпълнимия код на ядрото си и променливите. Скоростта на адресирането на паметта от всеки процесор е представена на фигура 8.

```
SRAM = Internal memory
ERAM = External memory

Host -> SRAM: Write speed =   14.62 MBps
Host <- SRAM: Read speed  =   17.85 MBps
Host -> ERAM: Write speed =  100.71 MBps
Host <- ERAM: Read speed  =  135.42 MBps

Using memcpy:
Core -> SRAM: Write speed =  504.09 MBps clocks = 9299
Core <- SRAM: Read speed  =  115.65 MBps clocks = 40531
Core -> ERAM: Write speed =  142.99 MBps clocks = 32782
Core <- ERAM: Read speed  =    4.19 MBps clocks = 1119132

Using DMA:
Core -> SRAM: Write speed = 1949.88 MBps clocks = 2404
Core <- SRAM: Read speed  =  480.82 MBps clocks = 9749
Core -> ERAM: Write speed =  493.21 MBps clocks = 9504
Core <- ERAM: Read speed  =  154.52 MBps clocks = 30336
```

Фигура 8: Скорости на писане и четене в различните памети от всеки от процесорите

5.2 Софтуер

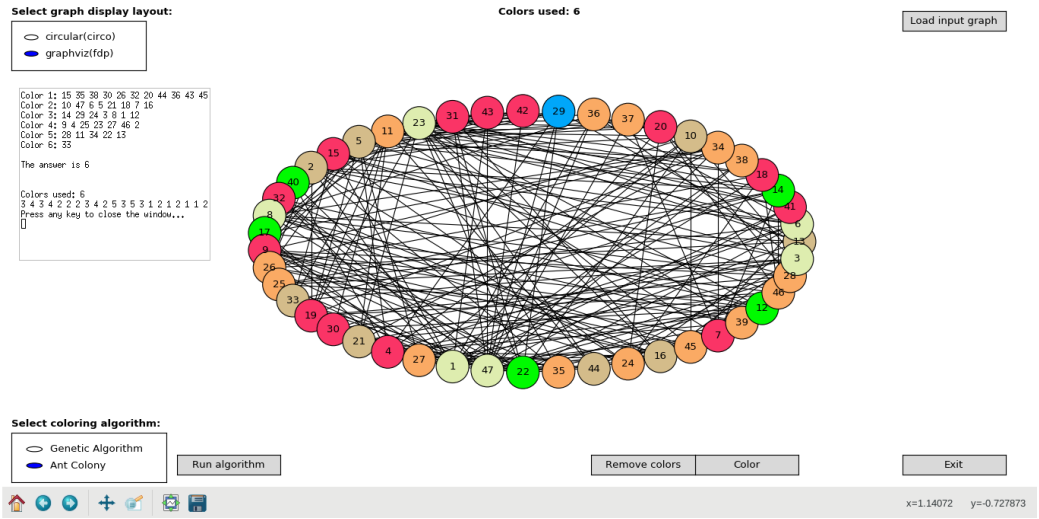
От горепоказаните резултати се вижда, че тъй като Epiphany процесорът има по-голяма изчислителна мощност, то с него трябва да се правят тежките изчисления в алгоритмите. Но малката скорост за четене от споделената памет е основен лимитиращ фактор. Също така, освен споделената памет набора от иструменти за софтуерна разработка не предлага друг начин за комуникация между двата процесора. Затова ние използваме наш протокол за комуникация, който се основава с това, че в дадени сектори от паметта, обозначени по име пазим графа като списък на съседите и матрицата с феромоните. Отделно с помощта на едно 32-битово число се пази статус на всяко ядро. И това служи за изпращане на случайния сийд и на получаване на адреса на отговора в паметта. За още по-бързо четене от паметта използваме DMA канали в Epiphany ядрата. Освен лимитациите от споделената памет, такива налага и паметта на ядрата. Както е ясно, C++ добавя много повече удобства и посредством свои функции предотвратява някои проблеми от C. Но това се отразява в по-голям по размер изпълнителен код, който не може да се събере в 32-те KB памет на всяко ядро. Това налага превеждане на C++ кода на C. Това забранява използване на стандартната библиотека и налага наша имплементация на повечето контейнери предоставени от нея и използвани от нас. Като цяло, с премоделирането на кода за едночиповата система Parallella, в текущата научна разработка сме се успели да разрешим главните проблеми описани по-горе и получаваме един реален резултат как би работил алгоритъма на една реална многонишкова система.

6 Илюстратор

В тази секция ще обясним детайлно приложението, с който графът се визуализира за по-лесно интерпретиране от публиката.

6.1 Описание

За да може намерените решения да бъдат представени по подобаващ начин беше разработен визуализатор на Python. Той е разработен така, че с него от браузър за файлове се избира входен файл спрямо стандарта в DIMACS. Входният файл се илюстрира като граф от неоцветени върхове. След това потребителят си избира алгоритъм, с който да оцвети графа, следи работата му в конзолен прозорец. И след това, графът може да бъде оцветяван в различни цветове, но пак запазвайки правилото да няма два съседни върха в един и същи цвят. Илюстратора също така поддържа уголемяване и местене на графа, което е полезно при много големи графи. Потребителят има избор от два начина на илюстрация - циклична и като мрежа. Примерна илюстрация е представена на фугура 9.



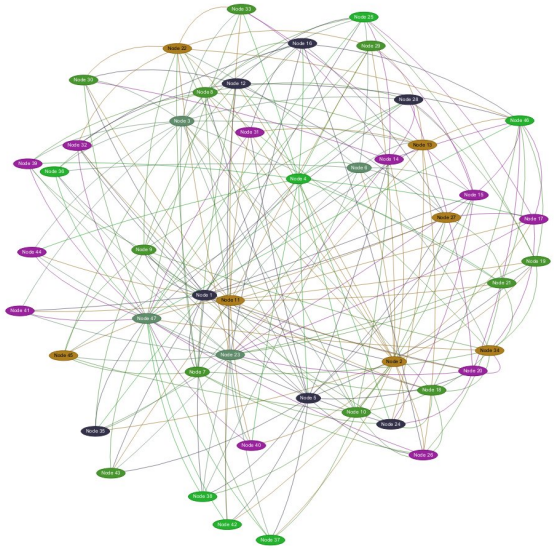
Фигура 9: Изображение, представящо илюстрирането на графа от myciel3.col от DIMACS

6.2 Имплементация

За визуалните ефекти на илюстратора се използва GTK toolkit. Представянето на графа се осъществява чрез граф класа в Python и Graphiz библиотеката. Графът се изобразява използвайки библиотеката за изобразяване на мрежи NetworkX и Graphiz оформление.

6.3 Уеб разработка на илюстратора

Към проекта е включена и уеб разработка на илюстратора, направена с да се представи алтернатива, при която решението да бъде достъпно за повече клиенти. За back-end на разработката е използван HTTP сървър, написан на Node.js. За комуникацията между back-end и front-end се използва библиотеката socket.io, издадена под MIT лиценз. Намереното решение и тестовия пример се изпращат към front-end чрез сокет на socket.io. При получаване на информацията, front-end използва решението за да генерира цветове и за да оцвети върховете. Тестовият пример за използва за реконструиране на графа, който след това се визуализира с помощта на библиотеката vis.js, издадена под MIT лиценз. Уеб разработката на илюстратора е във фаза на разработване и много от функциите на илюстратора на Python не са имплементирани. Изображение направено от уеб разработката е представено на фигура 10.



Фигура 10: Изображение, представящо илюстрирането на графа от myciel5.col от DIMACS с уеб илюстратор

7 Тестове и резултати

Представени са тестове на двата алгоритма, които са проведени на двуядрен Intel Core i5 3230M @2.6GHz с 4 нишки и 8GB DDR3 RAM @1600MHz, едночиповата система Parallella и Intel Core i7 4790K @4.0Ghz с 8 нишки.

Генетичен алгоритъм

Test	Answer	i5 answer	i5 time	i7 answer	i7 time
myciel3.col	4	4	1.1s	4	0.2s
myciel4.col	5	5	3.7s	5	2.2s
myciel5.col	6	6	23.6s	6	8.6s
myciel6.col	7	7	1m11s	7	26.9s
myciel7.col	8	8	10m2s	8	4m8s

Алгоритъм за мравешка колония

Test	Answer	i5 answer	i5 time	Parallella answer	Parallella time
myciel3.col	4	4	0.72s	4	0.87s
myciel4.col	5	5	0.78s	5	6.7s
myciel5.col	6	6	0.88s	6	1m01s
myciel6.col	7	7	1.74s	7	10m54s
myciel7.col	8	8	10.5s	-	-

7.1 Анализ на резултатите

7.1.1 Анализ на резултатите от Интел процесорите

Резултатите от проведените тестове показват, че и двата алгоритъма успяват да намерят решение в приемлив времеви интервал. В сравнение с методи като Brute Force, разработените алгоритми намират решение много по-бързо. Намерените отговори от алгоритмите съвпадат с хроматичното число на дадения граф, което говори за правилната работа на алгоритмите. Също така, тестовете показват, че алгоритъмът за мравешка колония намира решение много по-бързо от генетичния алгоритъм. Причина за това е алчният алгоритъм RLF, който намира решение доста бързо при тези примери.

7.1.2 Анализ на резултатите от едночиповата система Parallella

Въпреки своята мощност и броя ядра, едночиповата система Parallella дава по-лоши резултати от Интел i5 процесора. Това може да се дължи на бавното четене и писане в паметта както и на имплементираните от

нас заместители на контейнерите от стандартната библиотека. Но важното в този случай е, че се достига до правилен отговор, защото това означава, че алгоритъма се паралелизира правилно и работи правилно на устройството.

8 Заключение

В текущата научна разработка са представени тестове за производителност на паралелизирани метаевристични алгоритми решаващи проблемът за оцветяване на граф. В лицето на представените метаевристични алгоритми са паралелизираните и оптимизираните генетичен алгоритъм и алгоритъм на мравешка колония. Както можете да видите в предишната секция, след тестове се забелязва, че тези алгоритми са написани правилно и решават задачата в доста приелив времеви интервал. Това още веднъж показва, че паралелните алгоритми работят много по-ефективно от последователните, и че те са в основата на развитието на бъдещите технологии.

Но тази разработка има още много места за подобрение.Те включват:

- Подобряване на алгоритъма за едночиповата система Parallella
- Манипулиране на константите на генетичния алгоритъм за да се намери случай, в който времето за намиране на отговор е най-малко
- Оптимизиране на работата на алгоритъм на мравешката колония с множества, което сега забавя много алгоритъма
- Подобряване на уеб илюстратора
- Намиране на нови метаевристични алгоритми, имплементирането, оптимизирането и тестването им

Литература

[1] *Abbasian, Reza,* and *Malek Mouhoub.* "An efficient hierarchical parallel genetic algorithm for graph coloring problem."Proceedings of the 13th annual conference on Genetic and evolutionary computation. ACM, 2011.

[2] Adapteva. "Epiphany SDK Reference."Parallella.org. Adapteva, 2013. Web. 31 Sept. 2016.

[3] Adapteva. "Epiphany Architecture Reference."Parallella.org. Adapteva, 2013. Web. 31 Sept. 2016.

[4] Adapteva. "Parallella-1.x Reference Manual."Parallella.org. Adapteva, 2014. Web. 31 Sept. 2016.

[5] *Barney, Blaise.* "Introduction to parallel computing."Lawrence Livermore National Laboratory 6.13 (2010): 10.

[6] *D. Costa* and *A. Hertz.* "Ants can colour graphs."Journal of the operational research society 48.3 (1997): 295-305.

[7] *Sörensen, Kenneth,* and *Fred W. Glover.* "Metaheuristics."Encyclopedia of operations research and management science. Springer US, 2013. 960-970.

[8] *Grama, Ananth,* et al. "Principles of parallel algorithm design."Introduction to Parallel Computing. 2nd ed. Harlow: Addison Wesley (2003).

[9] *Shekhawat, Anirudh, Pratik Poddar,* and *Dinesh Boswal.* "Ant Colony Optimization Algorithms : Introduction and Beyond."Artificial Intelligence Seminar. 2009. Lecture.

[10] *Hindi, Musa,* and *Roman V. Yampolskiy.* "Genetic Algorithm Applied to the Graph Coloring Problem."MAICS. 2012.

[11] *Graham, Richard L.,* et al. "Open MPI: A high-performance, heterogeneous MPI."2006 IEEE International Conference on Cluster Computing. IEEE, 2006.