Изследване на паралелизирани метаевристични алгоритми върху задачата за оцветяване на граф

Comparing parallelized metaheuristic algorithms for solving the graph coloring problem

Иван Даниелов Иванов
Ivan Danielov Ivanov
Американски колеж
София, 11-ти клас

Милен Младенов Ферев
Milen Mladenov Ferev
Американски колеж
София, 11-ти клас

Под ръководството на

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

Март 2017

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 задачата за оцветяване на граф"познатата NP-трудна задача за оцветяване на граф е решена, използвайки метаевристични алгоритми. Разработката включва генетичен алгоритъм, алгоритъм за оптимизация на мравешка колония, алгоритъм за симулирано закаляване и алгоритъм за табу търсене. Те се основават на съществуващи научни разработки, но при употребата им в проекта са паралелизирани и оптимизирани за работа на система за паралелно изчисление. За паралелизирането на алгоритмите се използва стандартният интерфейс за предаване на съобщения (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. Тестовете са проведени на стандартни x86 процесори от Intel и на едночипови системи.

Summary

With the increasing complexity of algorithms the demand for computational resources also increases. The classical model of serial computation does not have efficiency to solve the problems in a polynomial amount of time. The parallel computation model provides a better solution. In the project "Comparing parallelized metaheuristic algorithms for solving the graph coloring problem"the NP-complete graph coloring problem is solved, using metaheuristic algorithms. The project includes a genetic algorithm, an ant colony optimization algorithm, a simulated annealing algorithm, and a tabu search algorithm. They are based on existing scientific publications, but 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. The tests were performed on standard x86 Intel processing units and on System-On-Chip devices.

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 структуриран 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, алгоритъм за симулирано закаляване и алгоритъм за табу търсене са 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 отива за още. По този начин, феромоните се наслагват върху най-кратките пътища и мравките започват да използват само тях. Подобно на това, в нашата разработка различни процеси работят върху оцветяването на графа и при всяко намерено решение започват да се търсят нови, близки до него.

Алгоритъмът за симулирано закаляване наподобява процеса на охлаждане на метали в металургията. В началото, температурата е висока, което в някои случаи позволява текущото решение да бъде заменено с по-лошо от него. Така алгоритъмът не остава в локален минимум, което е предимство пред алчните алгоритми. Постепенно, подобно на охлаждащия се метал, температурата на алгоритъма намалява, с което намалява

и вероятността по-лошо решение да смени текущото.

Алгоритъмът за табу търсене наподобява алгоритъма за симулирано закаляване в метода на търсене на решение. Разликата между алгоритмите е, че вместо намаляване на температура, алгоритъмът за табу търсене създава списък със забранени решения. При смяна на текущото решение, новото се включва в списъка с забранени решения за определен брой итерации. Така алгоритъмът избягва оставане в локален минимум.

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.

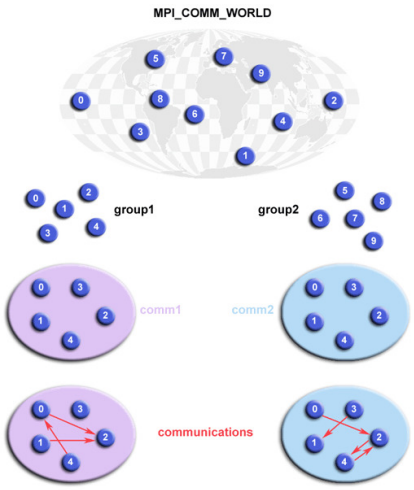
Изложението в текущата разработка е разделено на осем секции: **Интерфейс за предаване на съобщения MPI**, **Подход с генетичен алгоритъм**, **Подход с алгоритъм за оптимизация на мравешка колония**, **Подход със симулирано закаляване**, **Подход с табу търсене**, **Едночиповата система Parallella**, **Илюстратор**, **Тестове и резултати**, където в детайли ще обсъдим интефейса за предаване на съобщения, използвания генетичен алгоритъм, използвания алгоритъм за оптимизация на мравешка колония, използвания алгоритъм за симулирано закаляване, използвания алгоритъм за табу търсене, кратко описание на едночиповата система Parallella, която е използвана в разработката, разработения илюстратор, 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, срещнат 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 има широко разпространение [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 разработка 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.

Фигура 2: Логото на OpenMPI

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, подобен на генетичния модификатор, който намира, не задължително оптимално, оцветяване, 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.

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 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 Подход със симулирано закаляване

Алгоритъмът за симулирано закаляване следва метода на закаляване на метали чрез нагряване и охлаждане. Направен е по модела на Szymon Łukasik, Zbigniew Kokosiński и Grzegorz Świętoń.

5.1 Авторска разработка

В авторската разработка е представен алгоритъм за симулирано закаляване, който при дадено хроматично число k намира оцветяване на графа, при което няма конфликтни двойки върхове. В процеса на работа на авторския алгоритъм всеки от второстепенните процеси получава оцветяване от главния процес и генерира нови оцветявания, като се стреми да подобри текущото чрез премахване на конфликтите. Всяко оцветяване има своя оценка, определена от броя конфликти и използваните цветове. Второстепенните процеси периодично изпращат оцветяването с най-добра оценка на главния процес. Главният процес избира най-доброто от получените оцветявания и го изпраща на второстепенните процеси, където наново започва процесът на генериране и намиране на най-добро оцветяване. Алгоритъмът спира своята работа, след определен брой итерации или ако не може да бъде намерено по-добро решение от текущото.

5.1.1 Закаляване

Важен процес при сравняването на две решения е процесът на закаляване. При него се изчислява вероятността P решение с по-ниска оценка да бъде избрано като по-добро. По този начин, се предотвратява оставане в локален минимум при работата на алгоритъма.

$$P(\Delta_{cost,i}) = e^{-\frac{\Delta_{cost,i}}{T_i}}$$

Фигура 6: Функцията, с която се изчислява вероятността P

5.1.2 Температура

В алгоритъма температурата оказва ефект върху вероятността да се избере решение с по-ниска оценка като по-добро. При стартиране на алгоритъма температурата е висока като това позволява по-голяма вероятност. С броя итерации температурата намалява, като така намява и

вероятността решение с по-ниска оценка да бъде избрано. Температурата се намалява по метода: T = aT, където $0 < a < 1$.

5.2 Оптимизации

Алгоритъмът в авторската разработка намира правилно оцветяване според броя цветове k. Тъй като в задачата за оцветяване на граф се търси и най-малкият брой цветове k, авторският алгоритъм трябваше да бъде променен и оптимизиран.

5.2.1 Приближаващ модул

За да се намери горна граница за хроматичното число, към алгоритъма беше добавен приближаващ модул. Целта на приближаващия модул е да намери начално оцветяване с брой k на брой цветове, което да се използва като горна граница за хроматичното число от алгоритъма. Използваният **Приближаващ модул** беше взаимстван от разработения генетичен алгоритъм, поради неговата ефективност.

5.2.2 Метод на намиране на решение

За да може алгоритъмът да бъде приспособен към решаване на задачата за оцветяване на граф, методът на намиране на решение беше променен. При стартиране на алгоритъма всеки второстепенен процес намира горна граница за броя цветове чрез приближаващия модул, описан по-горе. След това, се изпълнява гореописания метод на симулирано закаляване, за да намери оцветяване. Периодично всеки второстепенен процес изпраща най-доброто намерено решение на главния процес, който намира текущото най-добро от получените и го изпраща обратно на вторичните процеси. Тези стъпки се повтарят даден брой итерации или докато не може да буде намерено по-добро решение от текущото.

6 Табу търсене

Алоритъмът за табу търсене е метаевристичен алгоритъм от класа алгоритми за локално търсене. Той работи посредством списък със забранени решения, или така наречен табу списък. Тази структура съдържа решения, които не трябва да бъдат достъпвани, като по-този начин се избягва оставане в локален минимум за решенията.

6.1 Авторска разработка

Алгоритъмът е разработен по модела на Алиан Херц и Доминик де Вера. Първоначално се създава решение на задачата по случаен принцип, което служи за шаблон. Върху него множество пъти се прилага операцията за подобрение. Тя е съставена от три главни части. Първо, създават се определен брой съседни решения на текущото, които не присъстват в списъка със забранени решения. От тях се избира най-доброто. Текущото става равно на най-доброто и то е добавено в списъка със забранени решения. След определен брой такива стъпки, алгоритъмът приключва своята работа и отпечатва текущото решение.

6.1.1 Намиране на съседни решения

Важна стъпка от алгоритъма за табу търсене представлява намирането на съседни решения на дадено такова. Това се осъществява като цветът на точно един връх от даденото бива променен на някой от другите цветове. По този начин новото решение се различава от текущото, но остава достатъчно близко до него, което гарантира, че няма да бъдат изпуснати решения в процеса на търсене.

6.1.2 Списък със забранени решения

Основният елемент на алгоритъма за табу търсене представлява списъкът със забранени решения. Той представлява съвкупност от решения, до които алгоритъмът няма право да достига. По този начин веднъж достигнал локален минимум, алгоритъмът няма да остане в него, а ще излезе, тъй като ще му бъде забранено да повтаря решенията от този минимум. Начинът по който функционира списъкът е следният. В момента, в който се достигне ново решение, ако списъкът е достигнал максималната си дължина, то най-старият елемент се премахва и текущото решение се добавя на негото място без да се променя дължината. Ако

максималната дължина не е достигната, то тогава не се премахва елемент. Максималната дължина на списъка е взета съобразно авторската разработка и е с големина от 7 решения.

6.2 Оптимизации

Тъй като алгоритъмът в авторската разработка е разработен за последователно изчисляване и намира решение спрямо предварително зададено хроматично число k, то за да може да бъде тестван в нашия случай, ние добавихме приближаващ модул и схема за паралелизация.

6.2.1 Приближаващ модул

С оглед на добрите резултати на приближаващия модул в разработката на Реза Абасиан и Малек Моухоуб, използвахме разработения в генетичния алгоритъм приближаващ модул и в алгоритъма за табу търсене. Приближаващият модул представлява алчен алгоритъм, който намира решение на дадената задача за кратко време и със сравнително малко цветове. Това помага на алгоритъма за табу търсене да започне търсенето с по-малка горна граница и по-този начин да има по-голяма успеваемост в намирането на хроматичното число на зададения граф. Повече за приближаващия модул може да прочетете в подсекцията **Приближаващ модул** на описания генетичен алгоритъм.

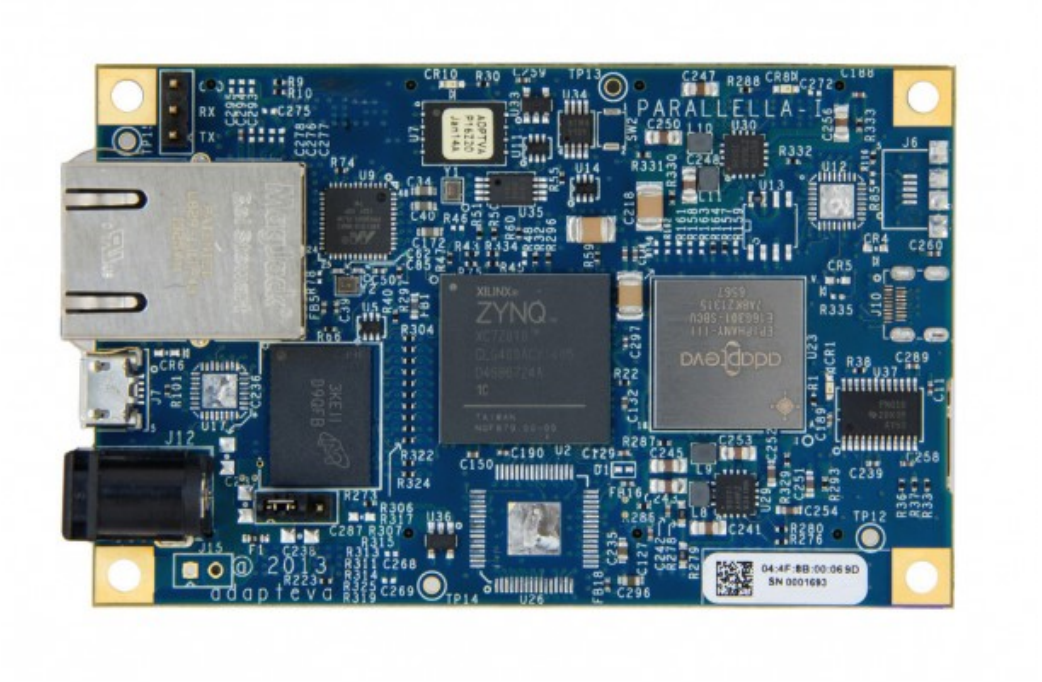
6.2.2 Паралелизация

Съществуват три основни вида паралелизации за алгоритми от класа на табу търсене. Първият вид изисква да се изпълнят едновременно n на брой напълно независими процеси на алгоритъма и накрая да се избере най-добрият получен отговор. При този вид, от една страна, няма комуникация между процесите и поради това времето за изпълнение е намалено, но от друга страна, без комуникация решенията могат да достигнат до един и същи резултат накрая, което прави изпълнението им паралелно без резултат. Вторият вид изисква да се изпълнят едновременно **n** на брой процеси на алгоритъма, като след всяка стъпка процесите да обменят решенията си и текущото на всеки процес да става най-доброто намерено досега. Този вариант ще позволи на процесите да не достигат до едно и също решение, а да подобряват вече намерено. Но допълнителното време на изпълнение, което ще отнеме комуникацията, и това, че процесите ще имат напълно същата информация на всяка стъпка ще ограничи броят решения, които ще бъдат разгледани. Третият

вид паралелно изпълнение е смесица от първите два: n на брой процеси от алгоритъма се изпълняват, като на всеки T стъпки, те обменят намерените резултати и започват ново търсене с най-доброто решение намерено до този момент. Този вид паралелно изпълнение е използван и в нашата разработка, тъй като избягва недостатъците на предишните два и по този начин е по-ефективен от тях.

7 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 7.

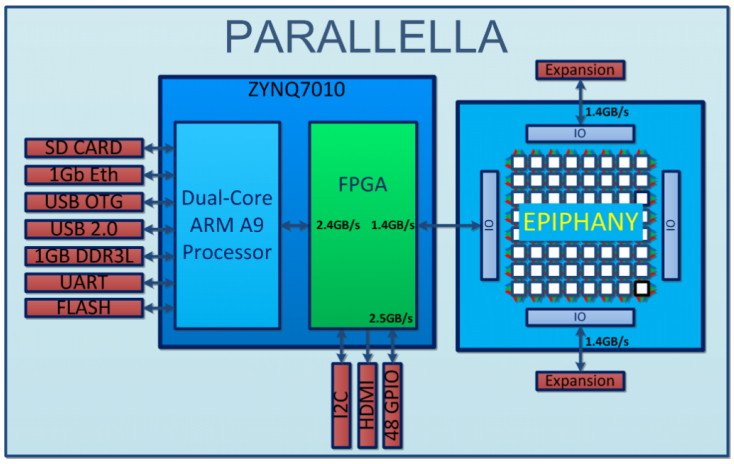


Фигура 7: Стандартния модел на едночиповата система с предназначение за сървър

7.1 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 8.



Фигура 8: 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 9.

```
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 9: Скорости на писане и четене в различните памети от всеки от процесорите

7.2 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.

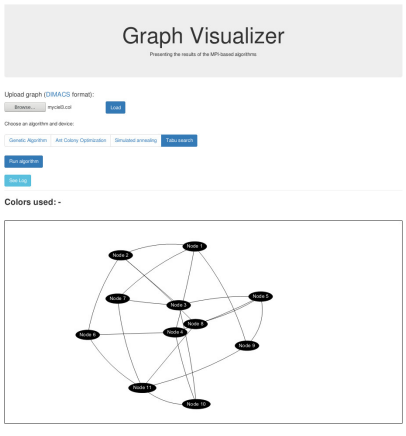
8 Илюстратор

С цел демонстрация на работата на алгоритмите и по-лесно възприемане на задачата за оцветяване на граф, в допълнение към имплементацията на гореописаните алгоритми сме разработили и уеб илюстратор.

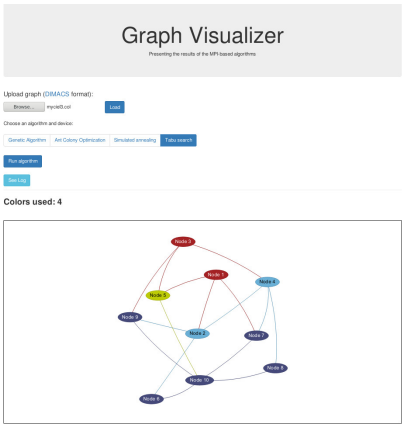
8.1 Описание

Илюстраторът е имплементиран чрез библиотеката за илюстриране на обекти в *Javascript*, *vis.js*, издадена под MIT лиценз. Той се състои от front-end и back-end част. Front-end частта представлява уеб страница, в която потребителят изпраща тестов пример и избира алгоритъм, с който да бъде изпълнени операциите. След като бъде намерено решение, то автоматично се появява на страницата и потребителят може да го разгледа подробно благодарение на интерактивната среда. В допълнение е достъпно и поле с подробната работа на алгоритъма. За дизайн е използван Bootstrap. Back-end частта се състои от уеб сървър-а имплементиран под Node.JS. Той отговаря за изпълнението на алгоритмите при заявка от front-end часта и за пренасочване на техния резултат обратно към нея. За комуникацията между back-end и front-end се използва библиотеката socket.io, издадена под MIT лиценз. Снимки от илюстратора може да видите на Фигура 10, Фигура 11 и Фигура 12.

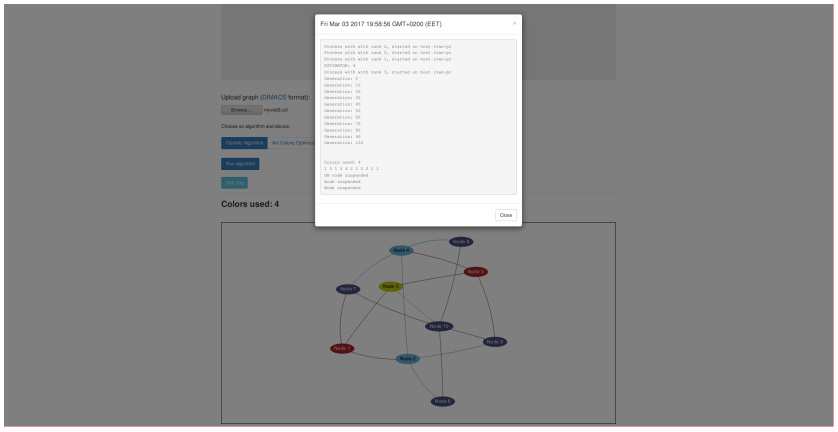
8.2 Примерни илюстрации



Фигура 10: Изображение, представящо илюстратора, когато тест myciel3.col от DIMACS е зареден



Фигура 11: Изображение, представящо илюстратора, когато тест myciel3.col от DIMACS е зареден и оцветен



Фигура 12: Изображение, представящо илюстратора, когато се разглежда подробният изход от генетичния алгоритъм, след като тест myciel3.col от DIMACS е оцветен

9 Тестове и резултати

9.1 Тестове върху x86 процесори от Интел

Представени са тестове на алгоритмите върху стандартни x86 процесори от Интел: двуядрен Intel Core i3 6098P @3.6GHz с 4 нишки и 12GB DDR4 RAM @2133MHz и Intel Core i7 4790K @4.0Ghz с 8 нишки.

Test	Vertices	Edges	Chromatic number	Algorithm	Threads	Answer	Time
myciel7.col	191	2360	8	Genetic	4	8	1:16s
myciel7.col	191	2360	8	Genetic	8	8	33.9s
myciel7.col	191	2360	8	Ant colony	4	8	2.7s
myciel7.col	191	2360	8	Ant colony	8	8	2s
myciel7.col	191	2360	8	Simulated annealing	4	8	0.5s
myciel7.col	191	2360	8	Simulated annealing	8	8	0.4s
myciel7.col	191	2360	8	Tabu search	4	8	6.8s
myciel7.col	191	2360	8	Tabu search	8	8	4s
queen7_7.col	49	476	7	Genetic	4	9	6s
queen7_7.col	49	476	7	Genetic	8	9	2.6s
queen7_7.col	49	476	7	Ant colony	4	8	0.2s
queen7_7.col	49	476	7	Ant colony	8	7	0.16s
queen7_7.col	49	476	7	Simulated annealing	4	9	0.5s
queen7_7.col	49	476	7	Simulated annealing	8	8	0.6s
queen7_7.col	49	476	7	Tabu search	4	8	1.9s
queen7_7.col	49	476	7	Tabu search	8	8	1.1s
miles1000.col	128	3216	42	Genetic	4	43	39s
miles1000.col	128	3216	42	Genetic	8	43	17s
miles1000.col	128	3216	42	Ant colony	4	42	1s
miles1000.col	128	3216	42	Ant colony	8	42	0.8s
miles1000.col	128	3216	42	Simulated annealing	4	42	1.1s
miles1000.col	128	3216	42	Simulated annealing	8	42	0.8s
miles1000.col	128	3216	42	Tabu search	4	42	11.2s
miles1000.col	128	3216	42	Tabu search	8	42	6.05s
anna.col	138	493	11	Genetic	4	11	24s
anna.col	138	493	11	Genetic	8	11	10s
anna.col	138	493	11	Ant colony	4	11	1s
anna.col	138	493	11	Ant colony	8	11	0.9s
anna.col	138	493	11	Simulated annealing	4	11	0.2s

anna.col	138	493	11	Simulated annealing	8	11	0.2s
anna.col	138	493	11	Tabu search	4	11	1.2s
anna.col	138	493	11	Tabu search	8	11	0.87s
mulcol.i.3.col	184	3916	31	Genetic	4	31	1:28s
mulcol.i.3.col	184	3916	31	Genetic	8	31	41s
mulcol.i.3.col	184	3916	31	Ant colony	4	31	2.3s
mulcol.i.3.col	184	3916	31	Ant colony	8	31	1.8s
mulcol.i.3.col	184	3916	31	Simulated annealing	4	31	0.8s
mulcol.i.3.col	184	3916	31	Simulated annealing	8	31	0.7s
mulcol.i.3.col	184	3916	31	Tabu search	4	31	10s
mulcol.i.3.col	184	3916	31	Tabu search	8	31	5.2s

9.2 Тестове върху едночипови системи

Представени са тестове на алгоритмите върху популярни едночипови системи и процесор Intel Atom N455, с изчислителна мощност близка до тази на останалите системи.

Test	Vertices	Edges	Chromatic number	Board	Threads	Answer	Time
myciel6.col	95	755	7	Raspberry Pi 3	4	7	1.8s
myciel6.col	95	755	7	Parallella ARM CPU	2	7	2.2s
myciel6.col	95	755	7	Parallella Epiphany CPU	16	7	56s
myciel6.col	95	755	7	Intel Atom N455	2	7	1.8s
queen7_7.col	49	476	7	Raspberry Pi 3	4	8	0.5s
queen7_7.col	49	476	7	Parallella ARM CPU	2	8	0.8s
queen7_7.col	49	476	7	Parallella Epiphany CPU	16	9	6s
queen7_7.col	49	476	7	Intel Atom N455	2	9	0.5s
huck.col	74	301	11	Raspberry Pi 3	4	11	1s
huck.col	74	301	11	Parallella ARM CPU	2	11	0.3s
huck.col	74	301	11	Parallella Epiphany CPU	16	11	31s
huck.col	74	301	11	Intel Atom N455	2	11	1s

9.3 Анализ на резултатите

9.3.1 Анализ на резултатите от Интел процесорите

Резултатите от проведените тестове показват, че всички имплементирани алгоритми намират решение за максимум 1 минута и 20 секунди. Това време е много добро за пример със 191 върха и 2360 ребра. Също така, получените резултати съвпадат с хроматичното число на дадения граф, или се различават с максимум 1 от него, което говори за ефективността на алгоритмите. По-детайлно се забелязва, че генетичният алгоритъм има най-дългото време за изпълнение. Но той обхожда и най-много решения и по този начин има най-голям шанс да намери хроматичното число. Алгоритъмът за оптимизация на мравешка колония има сравнително малко време за изпълнение, достига максимум до 6 секунди, и има доста голяма успеваемост с намирането на отговора. Но тъй като той си служи с алчен алгоритъм е възможно да се намерят примери, при които той ще е неефективен. Алгоритъмът за симулирано закаляване също има много късо време за изпълнение и голяма успеваемост с отговора. Но този алгоритъм основно разчита на горната граница, която му е дадена от приближаващия модул. Ако тя е твърде голяма, то няма как за определените итерации да бъде намерен отговор близък до хроматичното число. Алгоритъмът за табу търсене има по-дълго време за изпълнение в сравнение с алгоритмите за оптимизация на мравешката колония и симулирано закаляване, но по-късо от това на генетичния алгоритъм. Успеваемостта му в намирането на отговор също е доста голяма. Но той отново зависи пряко от горната граница, но е по-ефективен в това да я свали ако тя е твърде голяма.

9.3.2 Анализ на резултатите от едночиповите системи

С цел да сравним работата на алгоритмите върху едночипови системи сме предоставили и тестове на алгоритъма за мравешка колония. Избрахме този алгоритъм, тъй като изисква най-малко ресурси и дава много добри резултати на тестовете със стандартни x86 процесори. С резултатите от тези тестове целим да покажем, че имплементираните алгоритми могат да бъдат изпълнявани върху такива устройства за средно големи графи.

Въпреки своята мощност и брой ядра, Epiphany процесора на едночиповата система Parallella дава по-лоши резултати от останалите едночипови системи. Това 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 алгоритъмът се паралелизира правилно и работи правилно на устройството.

За разлика от Epiphany процесора на едночиповата система Parallella, ARM процесорите на системите Raspberry Pi 3 и Parallella успяват да намерят решение за по-малко време. Това се дължи на по-голямата скорост на писане и четене от паметта, както и на по-високата честота на процесорните ядра. ARM процесорите се справят с алгоритъма за време, подобно на процесора Intel Atom N455. Това се дължи на факта, че Atom серията процесори са разработени предимно за устройства, които не се нуждаят от много изчислителна сила.

10 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, алгоритъм за оптимизация на мравешка колония, алгоритъм за симулирано закаляване и алгоритъм за табу търсене. 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 при решаването на задачата за оцветяване на граф, и че те са в основата на развитието на бъдещите технологии.

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

- Оптимизация на имплементациите
- Проучване и разработване на алгоритъм за оптимизиране в рояк от частици
- Проучване и разработване на алгоритъм за гравитационно търсене

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] *Hertz, Alain,* and *Dominique de Werra.* "Using tabu search techniques for graph coloring."Computing 39.4 (1987): 345-351. *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.

[12] *Łukasik, Szymon, Zbigniew Kokosiński, and Grzegorz Świętoń.* "Parallel simulated annealing algorithm for graph coloring problem."International

Conference on Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, 2007.