Zeszyty Naukowe Wydziału
Elektroniki i Informatyki
Politechniki Koszalińskiej

ISSN:1897-7421
Nr 13, Listopad 2018
Str: 63-78

Piotr Ratuszniak
Wydział Elektroniki i Informatyki
Politechnika Koszalińska
ratusz@ie.tu.koszalin.pl

Radosław Łańcucki
Wydział Elektroniki i Informatyki
Politechnika Koszalińska
radek.lancucki@gmail.com

Andrzej Stasiak
isandrzej@gmail.com

Równoległa realizacja przykładowego algorytmu genetycznego z wykorzystaniem akceleratorów GPU

Słowa kluczowe: algorytm genetyczny, programowanie równoległe, akceleracja obliczeń, akceleratory GPU, CUDA, problem komiwojażera

1. Wstęp

Nadrzędnym celem wytworzonego oprogramowania w ramach prowadzonych badań naukowych, było porównanie wydajności sekwencyjnej oraz równoległej realizacji algorytmu genetycznego z wykorzystaniem techniki programowania procesora graficznego GPU (ang. Graphics Processing Unit), tj. GPGPU (ang. General-Purpose computing on Graphics Processing Units). Wykorzystana została dostępna moc obliczeniowa karty graficznej komputera, a w szczególności macierz jednostek wykonawczych w przetwarzaniu współbieżnym poddanego badaniu algorytmu. Jako problem do rozwiązania przez zaimplementowany algorytm genetyczny, oznaczono problem komiwojażera (ang. TSP – Travelling Salesman Problem). To typowe zadanie optymalizacyjne, polegające na jednokrotnym odwiedzeniu każdego miasta znajdującego się na mapie aktualnie rozwiązywanego wariantu. Każda mapa problemu komiwojażera składa się z miast wraz z zakodowanymi ich współrzędnymi. W równoległej realizacji oprogramowania wybrano technologię Nvidia CUDA, która należy do opisanej powyżej techniki programowania GPGPU.

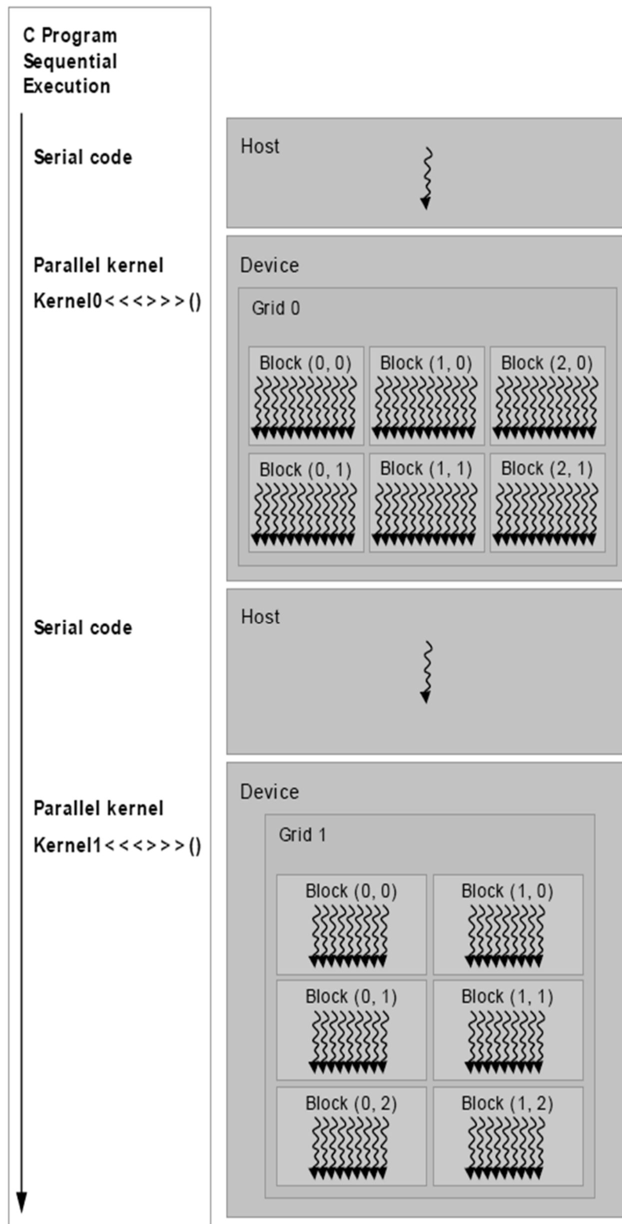
Wykonane oprogramowanie umożliwia rozwiązanie skonfigurowanego przez użytkownika algorytmu genetycznego w wersji sekwencyjnej, wykonywanej

całkowicie na procesorze komputera – CPU (ang. Central Processing Unit) oraz w wersji równoległej, gdzie obliczenia wykonywane są dodatkowo na karcie graficznej komputera. Jednym z założeń wykonanego oprogramowania była całkowita dowolność konfiguracji rozwiązywanego algorytmu genetycznego przez użytkownika, tj. dowolne ustawienie jego poszczególnych parametrów, czy też dowolna konfiguracja mapy rozwiązywanego przez niego problemu komiwojażera. W tym celu użyto biblioteki ManagedCUDA [1].

2. Technologia Nvidia CUDA

Technologia Nvidia CUDA zadebiutowała w lutym 2007 roku i jest rozwijana po dziś dzień. Obecnie wszystkie istniejące na rynku układy graficzne firmy Nvidia: GeForce, Tesla, Quadro; w pełni ją wspierają. Każdy program uruchamiany na karcie graficznej określany jest mianem tzw. jądra obliczeniowego (ang. kernel), które wykonywane jest zarówno przez CPU jak i GPU komputera. Technologia CUDA dostarcza jasno zdefiniowany model operacyjny programowania GPU: a) procesor alokuje oraz inicjalizuje pamięć operacyjną dla uruchomionego programu, b) alokacja pamięci układu GPU, c) skopiowanie wszystkich wymaganych danych z pamięci operacyjnej do pamięci GPU. Następną czynnością jest wywołanie przez procesor komputera jądra obliczeniowego, który operuje na wcześniej przygotowanych danych. Przetwarzanie danych na karcie graficznej odbywa się w sposób asynchroniczny w stosunku do CPU. Stanowi to bez wątpienia wielką zaletę opisywanej technologii, gdzie CPU nie uczestniczy w procesie obliczeniowym. Po wykonaniu zadania przez GPU, dane wynikowe z pamięci GPU są synchronizowane, tj. kopiowane do pamięci operacyjnej CPU. Ostatnim zadaniem CPU jest zwolnienie zarezerwowanych obszarów pamięci urządzenia GPU – dealokacja. Wszystkie jądra obliczeniowe mogą być wykonywane równocześnie przez wiele bloków (ang. blocks) CUDA. Bloki te pogrupowane są w tzw. siatki bloków (ang. grids). Za ich pojedyncze wykonanie odpowiedzialne są natomiast równoległe wątki (tzw. threads). Współczesne karty graficzne są w stanie wykonać nawet do 1024 wątków na blok jednocześnie.

Na rysunku 1 ukazano schemat przykładowego programu realizowanego sekwencyjnie wraz z równoległym wykonaniem po stronie GPU.



Rys. 1. Schemat przykładowego wykorzystania CUDA [2]

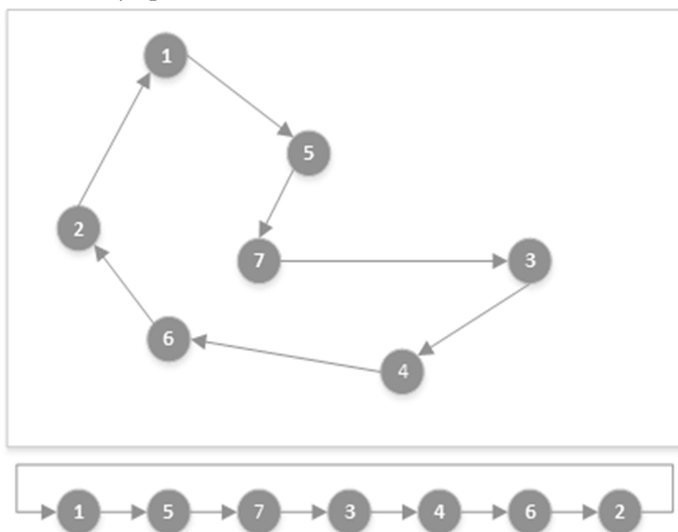
3. Algorytm genetyczny wykorzystany w rozwiązaniu problemu komiwojażera

Tak jak wspomniano we wstępie, utworzone oprogramowanie odpowiedzialne jest za rozwiązanie problemu komiwojażera. Problem ten jest zaliczany do grupy problemów NP-trudnych i jest on typowym problemem optymalizacyjnym. W związku z jego dużą złożonością $(n-1)!$, gdzie n to liczba wierzchołków miast na mapie – znalezienie optymalnego rozwiązania poprzez sprawdzenie wszystkich rozwiązań w przypadku większej liczby miast jest praktycznie niemożliwe z uwagi na bardzo długi czas działania programu. Zastosowanie w tym przypadku opisywanego algorytmu genetycznego pozwoli na znalezienie w rozsądnym czasie rozwiązania suboptymalnego, które jest w większości przypadków akceptowalne z uwagi na czas znalezienia tego optymalnego. Warunkiem zatrzymania zaimplementowanego algorytmu genetycznego jest osiągnięcie wprowadzonej przez użytkownika liczby generacji, bądź jego bezpośrednie zatrzymanie przez użytkownika programu. Poprzez generację algorytmu można rozumieć jeden pełny cykl algorytmu, czyli wykonanie działań oraz obliczeń dla jego całej populacji (wszystkich osobników). Na rysunku 2 przedstawiono schemat blokowy zaimplementowanego algorytmu genetycznego.

Rys. 2. Schemat blokowy algorytmu genetycznego

3.1. Reprezentacja osobników

Ważnym elementem implementacji algorytmu genetycznego jest sposób reprezentacji osobników w populacji. W zaprogramowanej aplikacji zastosowano permutacyjne kodowanie osobników, gdzie w genach chromosomów zapisywana jest informacja o numerze wierzchołka miasta odpowiadającego jego numerowi na mapie problemu komiwojażera. Rysunek 3 wraz z zawartą przykładową mapą, przedstawia omówiony sposób kodowania.



Rys. 3. Przykład kodowania permutacyjnego

3.2. Funkcja oceny i operatory genetyczne algorytmu

Funkcja oceny osobników

Funkcja oceny osobnika jest to miara jakości rozwiązania reprezentowanego przez dowolnego osobnika w populacji. Premiuje ona rozwiązania najbardziej zbliżone do optymalnego rozwiązania problemu komiwojażera, a jej wartość opisana jest wzorem 1, gdzie długość trasy jest zakodowana w chromosomie osobnika poddawanego ocenie.

$$\frac{1}{długośćtrasy} \qquad [1]$$

Operatory genetyczne

W algorytmie genetycznym zaimplementowanym w aplikacji zastosowano następujące operatory genetyczne:

- <u>Krzyżowanie</u>: szansa wystąpienia operatora krzyżowania mieści się w przedziale od 10% do 100% i jest ona ustalana przez użytkownika. Im

większa jego wartość tym większe prawdopodobieństwo jego wystąpienia. W procesie krzyżowania bierze udział dwoje rodziców (dwa skrzyżowane ze sobą chromosomy), w wyniku czego powstają dwa chromosomy potomne. Krzyżowanie to polega na wybraniu pierwszego genu od drugiego rodzica, a następnie na wybraniu kolejnych genów następujących po poprzednio wybranych zarówno od pierwszego, jak i drugiego rodzica. Ponadto żaden z genów w chromosomie nie może się powtarzać. W zaimplementowanym algorytmie zastosowano mechanizm krzyżowania zachłannego, które różni się od standardowego tym, że podczas wybierania kolejnego genu następuje obliczenie długości ścieżki pomiędzy wierzchołkami zakodowanymi w tych genach. Zachłanność polega na tym, że wybrany zostaje ten gen, którego obliczona długość ścieżki jest krótsza. Zastosowanie tego mechanizmu pozwoli w niektórych przypadkach na zwiększenie zbieżności algorytmu.

- Mutacja: szansa wystąpienia operatora jest identyczna jak w przypadku krzyżowania i jest ona również ustalana przez użytkownika. Mutacja w algorytmie genetycznym polega na losowym wybraniu dwóch genów w chromosomie, a następnie zamiana ich miejscami.
- Metody selekcji:
  - elitarna; metoda polegająca na selekcji najlepszych (najlepiej dostosowanych) osobników w populacji, a następnie umieszczenie ich w nowej populacji,
  - rankingowa; polega na sporządzeniu rankingu osobników na podstawie funkcji oceny. Następnie na jego podstawie tworzone jest wirtualne koło ruletki, gdzie największy wycinek koła zajmuje osobnik z najwyższą pozycją w rankingu. Po całej procedurze następuje losowanie, a wylosowane osobniki trafiają do nowej populacji,
  - koła ruletki; utworzone w ramach tej metody koło ruletki symbolizuje sumę ocen przystosowania wszystkich osobników, a poszczególne wycinki koła symbolizują wartość przystosowania danego osobnika w stosunku do całej populacji. Pozostałe czynności wykonywane w ramach tej metody są takie same jak w opisanej wcześniej metodzie rankingowej.

W celu zachowania możliwości pełnej konfiguracji przez użytkownika algorytmu oraz rozwiązywanego przez niego problemu, dokonano zrównoleglenia funkcji celu algorytmu genetycznego. Żaden z wymienionych parametrów nie jest zadeklarowany w oprogramowaniu na stałe, a jego wartość miała być ustalana tak jak wspomniano przez użytkownika aplikacji. Wybór na równoległą realizację funkcji celu padł z uwagi na niewystępowanie w niej czynników losowych oraz stałą liczbę elementów – osobników poddawanych zrównolegleniu. Poprzez czynniki losowe można rozumieć szansę wystąpienia poszczególnych operatorów

genetycznych, co wpływa później bezpośrednio na liczbę osobników biorących udział w obliczeniach. Brak występowania wspomnianej losowości w funkcji celu czyni ją jednocześnie najbardziej podatną na proces zrównoleglenia obliczeń ze wszystkich występujących w algorytmie genetycznym.

3.3. Zrównoleglenie algorytmu

Funkcja celu w algorytmie genetycznym podczas rozwiązywania problemu komiwojażera polega na obliczeniu łącznej długości ścieżki pokonywanej przez każdego z osobników w populacji z osobna. Długość pojedynczego odcinka pokonywanego w ramach zakodowanej w chromosomie osobnika trasy obliczane jest zgodnie ze wzorem 2

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad [2]$$

gdzie: x_2, y_2, x_1, y_1; współrzędne kolejno aktualnie przeliczanego wierzchołka miasta oraz poprzedzającego go wierzchołka miasta.

Powyższe obliczenia wykonywane są dla każdej pary wierzchołków zakodowanych w chromosomie osobnika w populacji, łącznie z połączeniem ostatniego wierzchołka z pierwszym. Po wyliczeniu długości trasy zakodowanej w chromosomie, poddawana ona jest następnie opisanej już ocenie funkcji przystosowania.

Liczba osobników biorących udział w równoległych obliczeniach określona jest wzorem 3

$$liczbapopulacji + K + M \qquad [3]$$

gdzie:

K – liczba osobników powstałych w wyniku krzyżowania,

M – liczba osobników powstałych w wyniku mutacji. Liczba ta jest zmienna dla każdej pojedynczej generacji algorytmu.

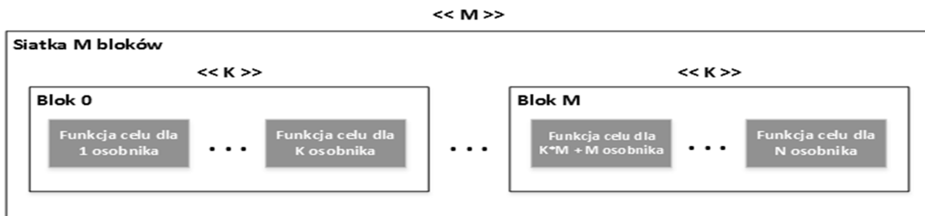
Jądro obliczeniowe programu wykonywanego na karcie graficznej GPU, odpowiedzialne za obliczenie opisywanej funkcji celu przyjmuje natomiast dwa parametry. Pierwszym z nich jest rozmiar bloku, którego wartość określona jest wzorem 4

$$\frac{liczbawątkównablokkartygraficznej}{8} \qquad [4]$$

natomiast drugim rozmiar siatki bloków, której wartość wyliczana jest ze wzoru 5

$$\frac{liczbaosobników+rozmiarbloku-1}{rozmiarbloku} \text{ w} \quad [5]$$

Powyższe parametry zostały w oprogramowaniu dobrane metodą empiryczną. Na rysunku 4 zaprezentowano schemat przedstawiający równoległe obliczenia wykonywane dla N osobników. Literą M oznaczono rozmiar siatki bloków, natomiast literą K rozmiar pojedynczego bloku.

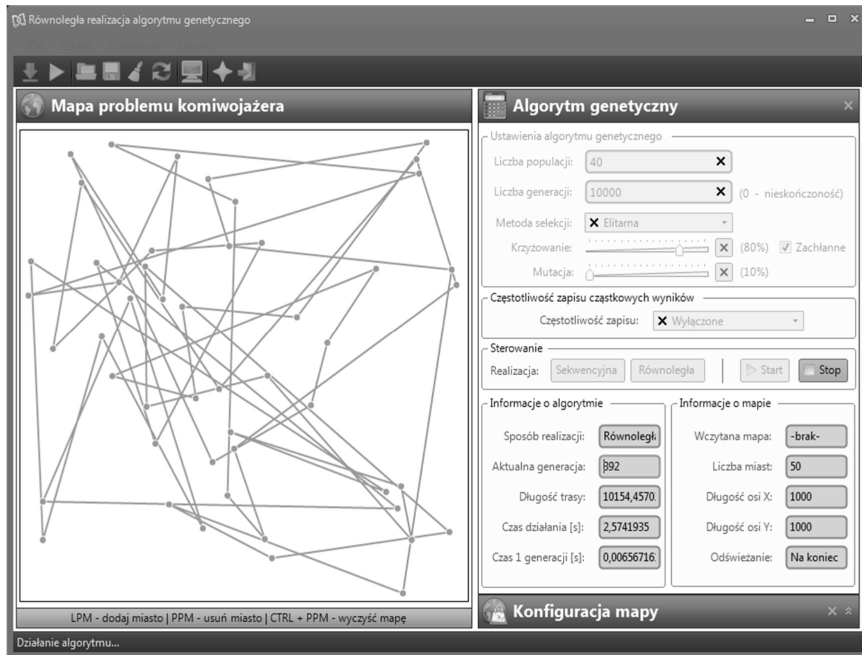


Rys. 4. Równoległa funkcja celu dla N osobników algorytmu genetycznego

Jak można łatwo zauważyć, obliczenia w ramach równoległej funkcji celu wykonywane są asynchronicznie w stosunku do siebie, co oznacza, że aby obliczyć funkcję celu dla 3 osobnika, nie jest konieczne wykonanie obliczeń dla dwóch poprzedzających go osobników. Za obliczenia odpowiedzialne są bezpośrednio rdzenie CUDA karty graficznej.

4. Aplikacja

Opisaną w artykule aplikację wykonano w celu porównania wydajności sekwencyjnej i równoległej wersji algorytmu genetycznego. Porównaniu poddany został czas realizacji algorytmu wykonanego w całości na procesorze CPU testowanego komputera (sekwencyjna wersja) oraz z zastosowaniem karty graficznej, która będzie odpowiedzialna za obliczenie funkcji celu (równoległa wersja) opisywanego algorytmu. Na rysunku 5 przedstawiono główne okno aplikacji.



Rys. 5. Główne okno aplikacji podczas działania algorytmu

W aplikacji można skonfigurować w pełni algorytm genetyczny poprzez ustawienie odpowiednio liczby populacji, liczby generacji algorytmu, metodę selekcji, szansę na wystąpienie operatora krzyżowania, czy też mutacji. Ponadto można w pełni skonfigurować mapę problemu komiwojażera, która ma zostać rozwiązana przez algorytm genetyczny. Można tego dokonać poprzez ustawienie: liczby miast na mapie, bądź też zmianę długości osi X oraz Y mapy. Użytkownik decyduje jaka realizacja algorytmu genetycznego ma zostać uruchomiona przez aplikację – sekwencyjna, czy też równoległa. Podczas działania algorytmu genetycznego w miejscu panelu konfiguracji mapy wyświetlany jest w aplikacji panel informacyjny o postępach jego pracy oraz drugi panel pokazujący informacje o mapie uruchomionego aktualnie problemu komiwojażera.

Aplikację wykonano w obiektowym języku programowania C#. Podczas implementacji opisywanej aplikacji wykorzystano między innymi następujące narzędzia i technologie: Microsoft Visual Studio 2012 Professional, pakiet CUDA Toolkit 6.0, bibliotekę programową ManagedCUDA (implementacje CUDA na platformie .NET), isku .NET, biblioteki: OpenHardwareMonitorLib (odczytywanie parametrów komputera), ComponentFactory.Krypton.Toolkit (wygląd aplikacji), OxyPlot (rysowanie wykresów).

Utworzony program dostarcza szereg funkcjonalności, w tym między innymi: wyświetlenie specyfikacji sprzętu komputera, zapis uzyskanych rezultatów, konfiguracja parametrów algorytmu genetycznego, określenie częstotliwości zapisu wyników cząstkowych, wybór trybu pracy – sekwencyjny/równoległy, natychmiastowe zatrzymanie algorytmu genetycznego, skonfigurowanie mapy problemu komiwojażera, wyświetlanie informacji o aktualnie uruchomionym algorytmie genetycznym oraz informacji o mapie problemu komiwojażera.

```
1  string resource = "ParallelGeneticAlgorithm.Kernels.kernel.ptx";
2  Streamstream =
   Assembly.GetExecutingAssembly().GetManifestResourceStream(resource);
3  CudaKernelrouteLengthCuda = cudaContext.LoadKernelPTX(stream,
   "RouteLength");
4
5  staticFunc<float[], float[], int[], int, float[]>cudaRouteLength =
   (host_mapx, host_mapy, host_route, N) =>
6  {
7  CudaDeviceVariable<float>device_mapx = host_mapx;
8  CudaDeviceVariable<float>device_mapy = host_mapy;
9  CudaDeviceVariable<int>device_route = host_route;
10 CudaDeviceVariable<float>device_length = newCudaDeviceVariable<float>(N);
11 float[] host_length;
12 routeLengthCuda.Run(device_mapx.DevicePointer, device_mapy.DevicePointer,
   device_route.DevicePointer, N, device_length.DevicePointer);
13 host_length = device_length;
14 device_mapx.Dispose();
15 device_mapy.Dispose();
16 device_route.Dispose();
17 device_length.Dispose();
18 returnhost_length;
19 };
20 float[] result = cudaRouteLength(MapX, MapY, Genes, Chromosomes.Count);
```

Rys. 6. Wywołanie jądra obliczeniowego z poziomu języka C#

Z uwagi na główne założenia wykonanego oprogramowania, wykorzystano zewnętrzną bibliotekę obsługującą technologię Nvidia CUDA w środowisku .NET., tj. ManagedCUDA. Biblioteka ta korzysta z plików PTX, które są skompilowanym przez *vncc*kodem maszynowym programu (kernel-a) napisanego w języku CUDA

C. Program ten obsługiwany jest przez funkcje biblioteki ManagedCUDA, która komunikując się z GPU przez jego sterownik graficzny, przekazuje kernel do GPU oraz uruchamia proces wykonawczy programu na rdzeniach CUDA.

Rysunek 6 przedstawia przykład obsługi jądra obliczeniowego (kernel) przez bibliotekę ManagedCUDA. Wspomniane jądro obliczeniowe odpowiedzialne jest za obliczenie długości ścieżki pokonywanej przez każdego kolejnego osobnika w populacji i uruchamiane ono jest w każdej generacji algorytmu genetycznego. Poprzez generację można rozumieć jeden pełny cykl pracy algorytmu genetycznego. Opisywane jądro obliczeniowe uruchamiane jest w każdej z generacji z innymi parametrami – rozmiarem bloku oraz rozmiarem bloku siatki z uwagi na zmienną liczbę elementów (osobników) biorących udział w równoległych obliczeniach.

Na rysunku 6 zaprezentowano przykładową programową obsługę procesów architektury Nvidia CUDA z poziomu języka programowania C#. Standardowy zestaw procesów niezbędny podczas uruchomienia jądra obliczeniowego oraz jego kontrola przy użyciu biblioteki ManagedCUDA, wygląda następująco:

1. Alokacja oraz inicjalizacja pamięci operacyjnej; (Rys. 6 wiersz 20)
2. Alokacja pamięci układu GPU oraz skopiowanie danych z pamięci operacyjnej do pamięci urządzenia; (Rys. 6 wiersze 7, 8, 9, 10 oraz 11)
3. Wywołanie jądra obliczeniowego; (Rys. 6 wiersz 12)
4. Skopiowanie rezultatu wykonania jądra z pamięci GPU do pamięci operacyjnej; (Rys. 6 wiersz 13)
5. Zwolnienie zarezerwowanej pamięci układu GPU; (Rys. 6 wiersze 14, 15, 16 oraz 17).

5. Wyniki

Badanie czasu wykonania zaimplementowanego algorytmu genetycznego realizowano na różnych wybranych platformach sprzętowych. W tabeli 1 przedstawiono specyfikację sprzętową komputerów, na których wykonano badania pomiarowe.

Tabela 1. Specyfikacja sprzętowa platform testowych

Podzespół	Nazwa parametru	Wartość parametru		
		I platforma	II platforma	III platforma
CPU	Nazwa:	Intel Core i5-4670K	Intel Core 2 Duo P7350	Intel Xeon E5506
	Liczba rdzeni:	4	2	4
	Taktowanie:	3800MHz	1995MHz	2130MHz
GPU	Nazwa:	GeForce GTX 760	GeForce GT 130M	Quadro K5000
	Wersja CUDA:	3.0	1.1	3.0
	Rdzenie CUDA:	1152 (6x192)	32 (4x8)	1536 (8x192)
	Wątki na blok:	1024	512	1024
	Taktowanie rdzenia:	1020MHz	1500MHz	705MHz
	Taktowanie pamięci:	1500MHz	800MHz	2700MHz
	Wielkość pamięci:	2048MB	512MB	4096MB
Pamięć RAM	Wielkość pamięci:	8192MB	4096MB	4096MB

W tabeli 2 przedstawiono zestawy parametrów algorytmu genetycznego, zgodnie z którymi przeprowadzono badania pomiarowe.

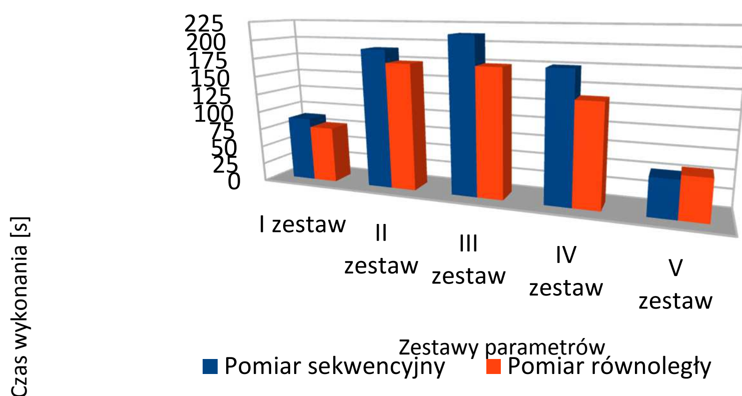
Tabela 2. Zestawy parametrów testowych algorytmu genetycznego

Parametr	Wartość parametru				
	I zestaw	II zestaw	III zestaw	IV zestaw	V zestaw
Liczba populacji	2500	4000	16500	12500	100
Liczba generacji	2000	2000	1000	1000	10000
Metoda selekcji	rankingowa	koła ruletki	elitarna	elitarna	elitarna
Krzyżowanie	40% (zachłanne)	50% (zachłanne)	80% (zachłanne)	50% (zachłanne)	80%
Mutacja	10%	15%	10%	10%	10%
Liczba miast	60	50	40	55	100
Wczytana mapa	mapa_zestaw_1	mapa_zestaw_2	mapa_zestaw_3	mapa_zestaw_4	mapa_zestaw_5

Dobór parametrów wpływa w sposób dominujący na wyniki obliczeń algorytmu genetycznego. Duża liczba miast oraz populacji może spowodować przetwarzanie jednej generacji nawet na poziomie kilkunastu sekund, co spowoduje oczywiście bardzo długi czas uzyskania przez algorytm zbieżności. Losowość działania poszczególnych operatorów genetycznych algorytmu genetycznego również ma wpływ na działanie samego algorytmu oraz na czasy poszczególnych uruchomień nawet dla tej samej konfiguracji sprzętowej.

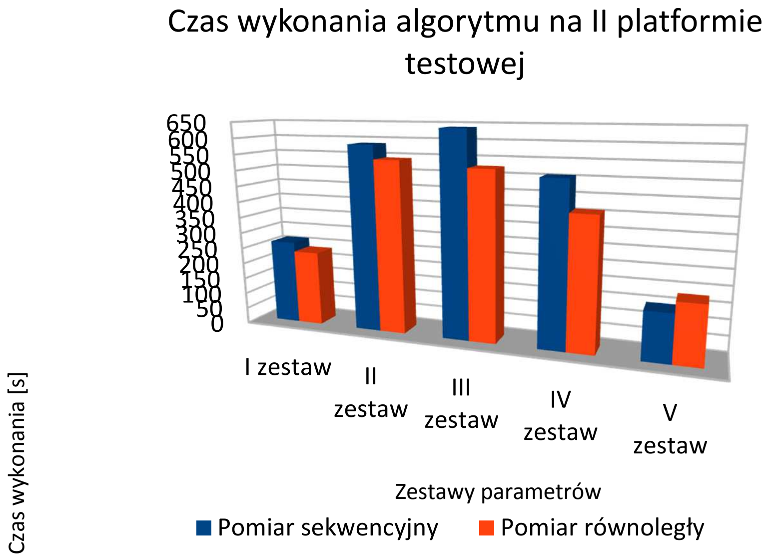
Na wykresach rysunku 7 kolorem niebieskim oznaczono czas pomiaru sekwencyjnego, natomiast pomarańczowym pomiar równoległej realizacji algorytmu. Należy zwrócić uwagę na różnicę w czasie wykonania pomiędzy obiema realizacjami algorytmu genetycznego dla poszczególnych zestawów parametrów testowych. Rezultaty czasu działania przedstawione na tym rysunku uzyskano na I platformie testowej.

Czas wykonania algorytmu na I platformie testowej



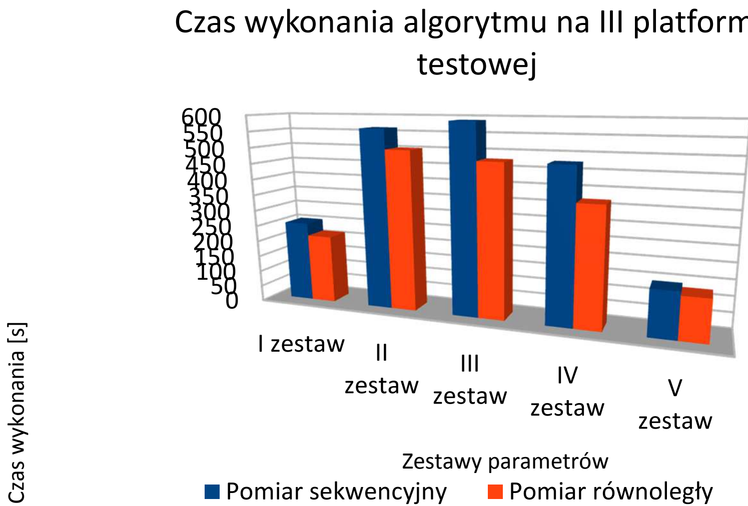
Rys. 7. Wykres czasu wykonania algorytmu genetycznego na I platformie testowej

Na rysunku 8 przedstawiono wykres czasu wykonania sekwencyjnego oraz równoległego na II platformie testowej.



Rys. 8. Wykres czasu wykonania algorytmu genetycznego na II platformie testowej

Na rysunku 9 przedstawiono rezultaty uzyskane na III platformie testowej.



Rys. 9. Wykres czasu wykonania algorytmu genetycznego na III platformie testowej

Czasy uzyskiwane przez równoległą realizację algorytmu genetycznego w większości przypadków były lepsze (tj. krótsze) w porównaniu

z jej sekwencyjnym odpowiednikiem. Opisywana równoległa realizacja okazywała się gorsza jedynie w przypadkach V zestawu parametrów, zarówno na I jak i II platformie testowej. Na wynik pomiaru w przypadku V zestawu parametrów miał wpływ bez wątpienia niski dobór liczby populacji algorytmu genetycznego, a co za tym idzie mała liczba elementów podlegających zrównolegleniu w funkcji celu. Na III platformie testowej czas uzyskany przez równoległą realizację był w każdym badanym przypadku lepszy, a miał na to bez wątpienia wpływ bardzo dobry akcelerator graficzny – Quadro K5000, charakteryzujący się największą liczbą rdzeni obliczeniowych.

6. Podsumowanie

Zgodnie z otrzymanymi rezultatami, zastosowanie technologii Nvidia CUDA pozwoliło przyspieszyć obliczenia algorytmu genetycznego dla przykładowo wybranego problemu. Chęć zachowania możliwości pełnej elastyczności w konfiguracji algorytmu genetycznego oraz mapy problemu komiwojażera jaka ma być przez niego rozwiązana bez ingerencji w kod programu, skutecznie ograniczyła pole manewru odnośnie integracji opisywanej aplikacji ze wspomnianą technologią CUDA. Czynnikami mającymi na to wpływ była przede wszystkim losowość towarzysząca algorytmowi genetycznemu, a co za tym idzie zmienne rozmiary tablic na których operuje się podczas jego działania, czy też zmienna liczba elementów poddawanych przetwarzaniu. Powyższa argumentacja może świadczyć o tym, że technologia GPGPU, do której zalicza się technologia CUDA, posiada swoje ograniczenia i wydaje się być bardziej efektywna dla wybranej grupy problemów. Pomimo świadomości istnienia wymienionych powyżej ograniczeń, na podstawie zrealizowanych badań można stwierdzić, że technologia Nvidia CUDA spełnia pokładane w niej oczekiwania i realizuje swoje zadanie przyspieszając w większości przypadków działanie aplikacji. Przyspieszenie to jest bardziej widoczne dla większej liczby osobników w populacji algorytmu genetycznego.

Zaprogramowana aplikacja dzięki zastosowaniu modułowej budowy oraz interfejsów może być w przyszłości z powodzeniem rozwijana. Głównymi kierunkami jej rozwoju, lecz z pewnością nie jedynymi są:

- dalsza optymalizacja obliczeń wykonywanych po stronie karty graficznej;
- dodanie kolejnych jąder obliczeniowych wykonujących obliczenia z innych jak obecnie obszarów algorytmu genetycznego;
- implementacja kolejnych algorytmów genetycznych.

Wymienione powyżej kierunki rozwoju aplikacji są oczywiście wolnymi sugestiami. Mają one za zadanie głównie zobrazowanie jak rozbudowany i czasochłonny w implementacji jest poruszany problem algorytmów genetycznych

oraz samej technologii GPGPU do której zalicza się Nvidia CUDA, a jej efektywność zależy od zaimplementowanego algorytmu oraz również od doświadczenia projektanta/programisty aplikacji.

Bibliografia

1. http://managedcuda.codeplex.com/documentation,
2. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, CUDA C Programming Guide,
3. Sanders J.; Kandrot E., CUDA by Example, Addison-Wesley, 2011, ISBN 0-13-138768-5,
4. Karaszewski M., http://problem-komiwojazera.cba.pl/,
5. SinhHoaNguyen, Algorytmy ewolucyjne, http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad10/w10.htm,
6. Lubiński T., Algorytmy genetyczne, http://www.algorytm.org/kurs-algorytmiki/algorytmy-genetyczne.html,
7. Lapthorn B., A Simple C# Genetic Algorithm, http://www.codeproject.com/Articles/3172/A-Simple-C-Genetic-Algorithm,
8. http://en.wikipedia.org/wiki/CUDA

Streszczenie

W artykule zaprezentowano praktyczną implementację aplikacji rozwiązującej przykładowy algorytm genetyczny z wykorzystaniem akceleratorów GPU. W tym przypadku zdecydowano się na rozwiązanie za pomocą algorytmu genetycznego typowego problemu optymalizacyjnego, jakim jest problem komiwojażera. Dodatkowo w celu wykorzystania mocy karty graficznej w tworzonej aplikacji wykorzystano technologię programowania na karcie graficznej – technologię Nvidia CUDA.

Abstract

The paper presents a practical implementation of a local desktop application that solves exemplary genetic algorithm with the use of GPU accelerators. In this case decided with the use of genetic algorithm to solve typical optimization problem which is travelling salesman problem. Additionally used Nvidia CUDA programming technology in order to use power of GPU in created application.

Keywords: genetic algorithm, parallel programming, computing acceleration, GPU, CUDA, travelling salesman problem