Łukasz Napora
Kamil Żak

DOKUMENTACJA WSTĘPNA DO PROJEKTU

„Kolorowanie grafów metodą włączeń-wyłączeń. Implementacja na procesorach GPU.”

Spis treści

Tabela zmian			
Wersja	**Data**	**Imię i nazwisko**	**Opis zmiany**
0.1	2014-11-09	Kamil Żak	Utworzenie dokumentu.
0.2	2014-11-10	Łukasz Napora	Uwagi oraz naniesienie poprawek.
0.3	2014-11-11	Kamil Żak	Zmiana zawartości dokumentu, po wstępnych uwagach promotora.
0.4	2014-11-12	Kamil Żak	Poprawki do wersji zaakceptowanej

1. Informacje ogólne

Niniejszy dokument jest wstępną dokumentacją dla projektu dotyczącego stworzenia aplikacji rozwiązującej problem kolorowania grafów metodą włączeń-wyłączeń. Zawiera przydział zadań do poszczególnych członków zespołu w czasie trwania całego projektu, oraz opisuje formalny podział projektu na poszczególne etapy, wraz z wypunktowaniem artefaktów dostarczonych w każdym etapie.

2. Wstęp teoretyczny

Ważnym zagadnieniem algorytmiki jest zadanie kolorowania grafów (lub też ogólniej podział zbioru). Jest to problem NP-trudny. Dlatego też cały czas powstają różne modyfikacje istniejących algorytmów, lub zupełnie nowe algorytmy, które mają na celu zmniejszenie złożoności czasowej obliczeń. W 2009 roku panowie Björklund, Husfeldt i Koivisto zaproponowali nowy algorytm[1] o złożoności czasowej $O^*(2^n)$[1]. Opiera się on na prostej zasadzie włączeń-wyłączeń. Jest to technika zliczania, która uogólnia znaną z teorii zbiorów metodę wyznaczania ilości elementów w wielu zbiorach (niepowtarzających się):

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n}|A_i| - \sum_{1\le i\le j\le n}\left|A_i \cap A_j\right| + \cdots + (-1)^{n-1}|A_1 \cap A_2 \ldots \cap A_n|$$

Lub inaczej

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{\emptyset \neq J \subseteq \{1,2,\ldots,n\}} (-1)^{|J|-1}\left|\bigcap_{j\in J} A_j\right|$$

Idea algorytmu polega na wielokrotnym wykonywaniu podstawowych obliczeń, na niewielkiej części danych. Dlatego też spróbujemy zaimplementować algorytm[1] w środowisku CUDA, które służy do przetwarzania równoległego. Celem naszej pracy, jest sprawdzenie wpływu obliczeń równoległych na złożoność czasową oraz porównanie wyników z obliczeniami w sposób asynchroniczny.

3. Wstępne wymagania

2.1 Ogólny opis projektu

Nasza praca inżynierska jest próbą efektywnej implementacji algorytmu kolorowania grafów (ogólniej pewnego podziału zbioru) metodą włączeń-wyłączeń. Poprzez efektywną implementację rozumiemy tu program rozwiązujący dany problem w jak najkrótszym czasie.

[1] Poprzez gwiazdkę rozumiemy, że złożoność jest rzędu $w*2^n$, gdzie w to pewien wielomian

Spodziewamy się, że dany algorytm da się zaimplementować w środowisku NVIDIA CUDA, co pozwoli wykonanie wielu takich samych małych operacji równolegle na dużej ilości wątków.

2.2 Model projektu

Posłużymy się iteracyjnym modelem tworzenia projektów, ze względu na możliwość wydzielenia komponentów projektu. Każdy etap będzie oddaniem pewnej części funkcjonalności całego programu wraz z niezbędną dokumentacją.

2.3 Funkcjonalność projektu

W ramach danego projektu zostanie stworzona aplikacja pozwalająca na obliczanie podziału zbioru (kolorowanie grafu) o dużym rozmiarze wejściowym. Będzie to zarówno możliwe na zwykłym procesorze (CPU), jak i na procesorze graficznym dedykowanym dla obliczeń równoległych (GPU).

2.4 Historyjki użytkownika

1. **Jako** użytkownik chcący obliczyć problem podziału danego zbioru**,**
   chcę móc wprowadzić dane poprzez graficzny interfejs**,**
   aby móc łatwo obsłużyć aplikację**.**
2. **Jako** użytkownik chcący obliczyć problem podziału danego zbioru**,**
   chcę móc sprawdzić statystyki obliczonego zadania dla podanych danych wejściowych**,**
   aby móc sprawdzić szczegółowe dane osiągniętego wyniku (takie jak czas wykonywania)**.**
3. **Jako** użytkownik korzystający z aplikacji**,**
   chcę widzieć prezentowane dane wejściowe i wyjściowe w graficznym UI**,**
   aby móc je zapisać jako obraz w celu np. wydrukowania**.**
4. **Jako** użytkownik korzystający z aplikacji**,**
   chcę mieć możliwość wyboru procesora obliczeniowego (CPU i GPU),
   aby porównać poprawę czasową obliczenia w różnych modelach obliczeniowych**.**
5. **Jako** użytkownik korzystający z aplikacji**,**
   chcę mieć możliwość zapisu danych wejściowych wraz z wynikiem w pliku tekstowym
   aby móc przechowywać historię obliczeń**.**
6. **Jako** użytkownik wprowadzający dane do aplikacji**,**
   chcę być poinformowany o błędnych wprowadzonych danych,
   aby wiedzieć, że odpowiednie dane muszą zostać wprowadzone ponownie**.**
7. **Jako** użytkownik korzystający z aplikacji**,**
   chcę mieć możliwość zobaczenia przykładowych danych,
   aby wiedzieć jak wyglądają prawidłowo skonstruowane dane wejściowe.
8. **Jako** użytkownik korzystający z aplikacji**,**
   chcę mieć możliwość zobaczenia przykładowych danych,
   aby wiedzieć jak wyglądają prawidłowo skonstruowane dane wejściowe.
9. **Jako** użytkownik korzystający z aplikacji**,**
   chcę być poinformowany przed rozpoczęciem obliczeń o długim czasie obliczenia problemu,
   aby wiedzieć czy będę musiał długo czekać na otrzymanie wyniku.

4. Terminarz

4.1 Ogólny harmonogram pracy

Termin	Wymagania	Artefakty	Osoby
12.11	Dokumentacja wstępna zawierająca podział prac w trakcie projektu, oraz wymagania wstępne dla projektu.	1. Terminarz 2. Wymagania wstępne	1. Łukasz Napora 2. Kamil Żak
27.11	Kompletna implementacja problemu na procesorach CPU, wraz ze wszystkimi niezbędnymi dokumentacjami. Wstępna implementacja problemu na procesorach graficznych (GPU) oraz wstępna dokumentacja techniczna dla wersji GPU.	1. Dokumentacja techniczna (CPU) 2. Kod źródłowy (CPU) 3. Instrukcja użytkownika (CPU) 4. Testy jednostkowe (CPU) 5. Dokumentacja techniczna (GPU) 6. Kod źródłowy (GPU) (wersja 0.1)	1. Łukasz Napora 2. Łukasz Napora, Kamil Żak 3. Łukasz Napora 4. Kamil Żak 5. Kamil Żak 6. Łukasz Napora, Kamil Żak
11.12	Dalsza implementacja problemu na GPU, modyfikacje mające na celu doprowadzenie do stworzenia wersji stabilnej. Uzupełnienie dokumentacji technicznej, instrukcji użytkownika, oraz pokrycie nowych funkcjonalności testami jednostkowymi.	1. Dokumentacja techniczna – naniesienie poprawek (tabela zmian) 2. Kod źródłowy (wersja alpha) 3. Instrukcja użytkownika – naniesienie poprawek (tabela zmian) 4. Testy jednostkowe – testy nowych funkcjonalności	1. Łukasz Napora 2. Łukasz Napora, Kamil Żak 3. Kamil Żak 4. Łukasz Napora
08.01	Dalsza implementacja problemu na GPU, modyfikacje implementacyjne ulepszające złożoność czasową problemu. Uzupełnienie dokumentacji technicznej, instrukcji użytkownika, oraz pokrycie nowych funkcjonalności testami jednostkowymi.	1. Dokumentacja techniczna – naniesienie poprawek (tabela zmian) 2. Kod źródłowy (wersja beta) 3. Instrukcja użytkownika – naniesienie poprawek (tabela zmian) 4. Testy jednostkowe – testy nowych funkcjonalności	1. Kamil Żak 2. Łukasz Napora, Kamil Żak 3. Łukasz Napora 4. Kamil Żak
22.01	Finalna wersja aplikacji w wersji GPU (wraz z niezbędnymi dokumentacjami). Porównanie wersji CPU i GPU oraz wyciągnięcie wniosków. Opisanie otrzymanych wyników.	1. Dokumentacja techniczna – wersja końcowa (tabela zmian) 2. Kod źródłowy – wersja final (1.0) 3. Instrukcja użytkownika – wersja końcowa (tabela zmian) 4. Testy jednostkowe – wersja końcowa (tabela zmian) 5. Dokument podsumowujący osiągnięte wyniki	1. Łukasz Napora 2. Łukasz Napora, Kamil Żak 3. Kamil Żak 4. Łukasz Napora 5. Kamil Żak

4.2 Szczegółowy opis tworzenia konkretnych artefaktów:

1. Dokumentacja techniczna – w ramach tego artefaktu przypisana osoba stworzy dokument o strukturze takiej jak w niniejszym dokumencie. Zostanie także opisana architektura stworzonego komponentu, diagram klas, wykorzystane środowisko, przykładowe zadanie oraz przykładowy wynik. W razie potrzeby osoba doda również przypadki użycia.
2. Kod źródłowy – w ramach tego artefaktu przypisana osoba tworzy czysty kod podzielony na klasy, bez nadmiarowej ilości linijek kodu. W przypadku wersji CPU będzie to kod w języku C# (Visual Studio), a w opcji GPU będzie to język C++ (Visual Studio). Ponadto każda klasa, metoda czy właściwość będą dokładnie opisane (dokumentacja kodu)
3. Instrukcja użytkownika – w ramach tego artefaktu przypisana osoba stworzy dokładny opis użytkowania danego komponentu wliczając w to: wymagania sprzętowe oraz systemowe, informacja o ograniczeniu rozmiaru rozwiązywanego problemu, opis uruchomienia programu oraz wprowadzenia danych wejściowych, posługiwanie się interfejsem, oraz poprawne odczytanie wyniku z aplikacji.
4. Testy jednostkowe – w ramach tego artefaktu przypisana osoba stworzy kod zawierający testy jednostkowe, które pokrywają min. 80% kodu źródłowego (w tym całą funkcjonalność).

4.3 Szczegółowy harmonogram tworzenia kodu źródłowego (podział tygodniowy)

Ad.1: W nawiasie podany jest język w jakim dana funkcjonalność jest zaimplementowana

17-23.11.14	UI (C#), zaimplementowanie wczytywania danych z pliku tekstowego do odpowiednich struktur w programie (C#)
24-30.11.14	Cały algorytm w wersji CPU (C#), połączenie algorytmu CPU z UI, dla wersji GPU wczytywanie danych wejściowych i ich podział (wraz z przesłaniem) na wątki (C++)
1-7.12.14	Implementacja algorytmu w wersji GPU (C++)
8-14.12.14	Podłączenie algorytmu GPU do środowiska równoległego – każdy wątek wywołuje dany algorytm (C++)
15-21.12.14	Zebranie wyników ze wszystkich wątków GPU i zapisanie wyniku do odpowiedniej struktury oraz pliku tekstowego (C++)
22-28.12.14	TERMIN ZAPASOWY
…-4.01.15	Stworzenie dll dla wersji GPU (C++) i przygotowanie do jej wykonania w UI (C#)
5-11.01.15	Stworzenie wersji CPU (C++) oraz wyodrębnienie z niej dll, podłączenie otrzymanego dll do UI (C#)
12-18.01.15	Dodanie opcji wyboru algorytmu (wykonanie odpowiedniej dll), zebranie danych i wyświetlenie statystyk (C#)
19-25.01.15	TERMIN ZAPASOWY

5. Bibliografia

1. Lawler, E.L. (1976), "A note on the complexity of the chromatic number problem", Information Processing Letters 5 (3): 66–67
2. Björklund, A.; Husfeldt, T.; Koivisto, M. (2009), "Set partitioning via inclusion–exclusion", SIAM Journal on Computing 39 (2): 546–563
3. NVIDIA GPU Programming Guide: https://developer.nvidia.com/nvidia-gpu-programming-guide