Konrad Lisiecki 291649

Zadanie 2 - Algorytm Brandesa

Programowanie współbieżne i Rozproszone

Semestr letni 2013/14

Warszawa, 2014-04-27

Spis treści

Rozdział 1

Wprowadzenie

Algorytm Brandesa Celem zadania jest zaimplementowanie Algorytmu Brandesa. Algorytm ten jest używany w zagadnieniach związanych z analizą sieci, a dokładniej rzecz ujmując, z identyfikacją ważnych wierzchołków w sieciach. Może on być stosowany do wyszukiwania istotnych elementów w sieciach nieważonych, tzn. takich dla których nie przypisujemy wag dla połączenia dwóch danych składników sieci. Jest to istotne założenie, ponieważ dzięki temu algorytm brandesa osiąga znacznie lepszą złożoność obliczeniową ($O|V||E|$) w porównaniu do najszybszych algorytmów działających na sieciach ważonych ($O|V|^3$ dla alg. Floyda-Warshalla). Jednak mimo tego ograniczenia algorytm Brandesa świetnie najdaje się do analizy wielu powszechnie występujących sieci, jak np. sieci społecznościowych.

Użyte jednostki obliczeniowe Przedstawiony powyżej algorym został zaimplementowany na dwóch jednostkach obliczeniowych, mianowicie na procesorze graficznym GPU i procesorze jednostki centralnej CPU. Dla implementacji na GPU zostanie użyta technologia OpenCL.

Struktura pracy Na początku zostanie przedstawiony schemat algorytmu na GPU wraz z wyszczególnionymi kernelami. Następnie, przejdę do sposobu w jaki budowałem algorytm i jakie stosowałem optymalizacje w poszczególnych krokach. Po tym nastąpi porównanie otrzymanych wyników na GPU i CPU dla różnych grafów wejściwoych. Na samym końcu zostanie przedstawiona tabela z dokładnymi wynikami.

Rozdział 2

Schemat algorytmu na GPU

Kernele Aby przenieść algorytm z CPU na GPU należy stworzyć kilka kerneli obliczeniowych. Są one uruchamiane sekwencyjnie dla każdego wierzchołka (wirtualnego) grafu. Zostały zaimplementowane następujące kernele:

1. InitArray - inicjalizuje tablice
2. Forward - wykonuje fazę forward algorytmu
3. Middle - wypełnienie tablicy delta
4. Backward - wykonanie fazy backward algorytmu
5. Final - uaktualnienie tablicy wynikowej

Schemat algorytmu wygląda następująco:

Algorithm 1 Schemat algorytmu z wyszczególnieniem użytych kerneli

```
1: for all u ∈ V do                          ▷ Dla każdego wierzchołka w grafie
2:     run Initarray(u)
3:     run Forward
4:     run Middle
5:     run Backward
6:     run Result(v)
7: end for
```

Maksymalna liczba sąsiadów wierzchołka Ponadto w algorytmie doświadczalnie wyznaczono maksymalną liczbę sąsiadów dla implementacji z wirtualnymi wierzchołkami. Liczba ta wynosi 32.

Rozdział 3

Optymalizacje szybkości działania algorytmu oraz GPU

Kroki podjęte podczas implementacji algorytmu Poniższe zestawienie zawiera najważniejsze kroki podjęte podczas implementacji i optymalizacji algorytmu:

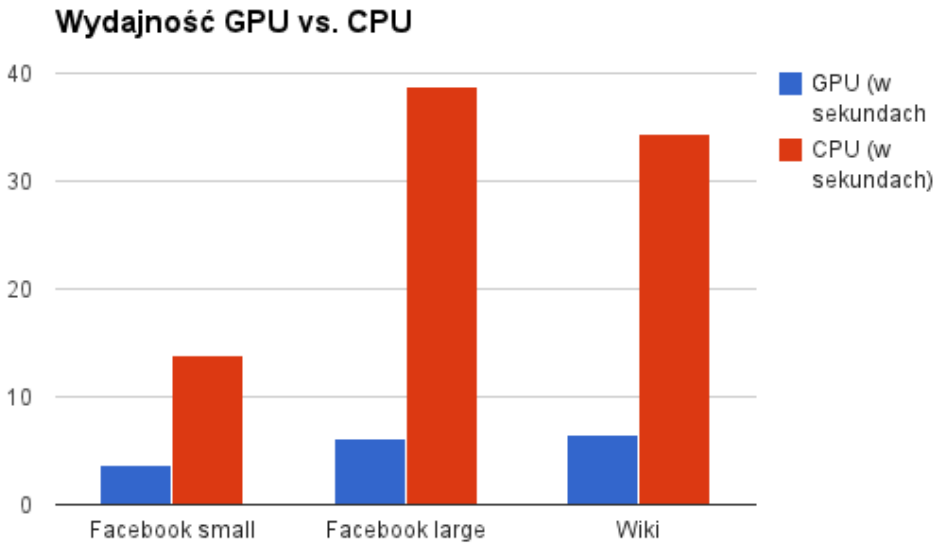
1. Implementacja bazowej wersji algorytmu na CPU
2. Przeniesienie algorytmu z CPU na GPU
3. Redukcja liczby kerneli
4. Próba skompresowania grafu
5. Implementacja wersji rozszerzonej algorytmu (o wirtualne wierzchołki) na CPU
6. Dodanie do bazowego algortymu na GPU wirtualnych wierzchołków (zauważenie, że wierzchołki o dużej liczbie sąsiadów spowalniają działanie algorytmu, stąd znaczne przyspieszenie działania)
7. Dostrajanie algorytmu przy pomocy zmiany maksymalnej liczby sąsiadów dla danego wierzchołka
8. Końcowe optymalizcjie na GPU, np. cachowanie wartości globalnych w pętlach występujacych w kernelach (aby zmniejszyć liczbę odwołań do pamięci)

Rozdział 4

Porównanie wyników implementacji algorytmu na GPU i CPU

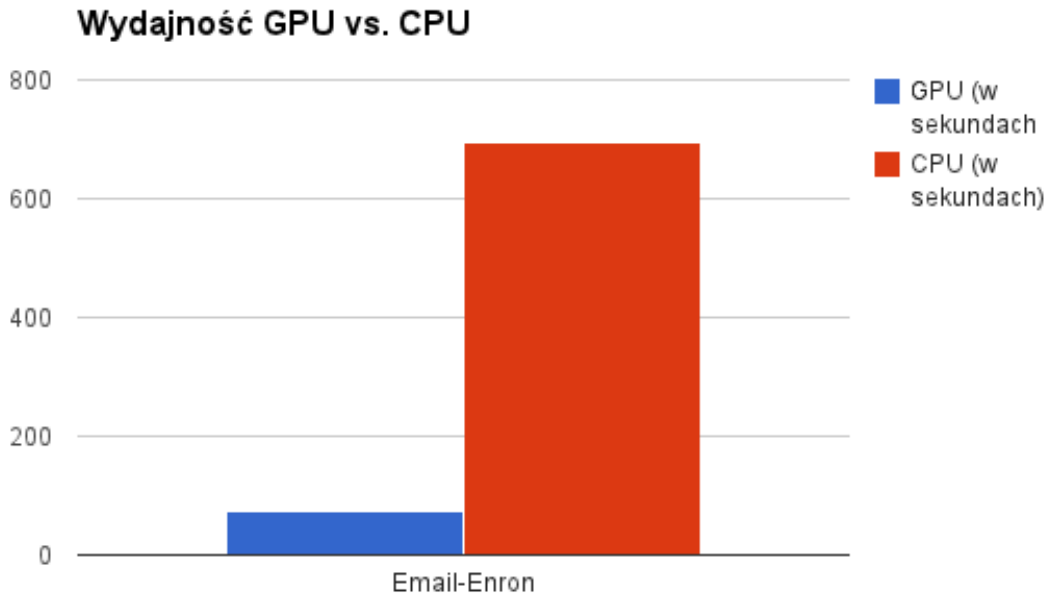
GPU vs. CPU W niniejszym rozdziale zostaną przedstawione wyniki z wykonania testów dla obydwu implementacji algorytmu. Testy te były wykonywane na karcie graficznej wspierającej technologię OpenCL **nVidia gtx 660**. Jest to karta graficzna gwarantująca przyspieszenie w stosunku do karty graficznej dostępnej na serwerach **nvidia1** oraz **nvidia2** rzędu **1,5-2x** (czas wykonania algorymtu Brandesa dla tej karty na danych **Facebook Small** to 3.669s, natomiast na kartach serwerów nvidia to 7.132s). Z kolei testy na układzie CPU zostały wykonane na procesorze **Intel Core i5-3230M**. Jest to procesor o częstotliwości taktowania zegara rzędu **2.60GHz-3.20GHz**.

Wyniki algorytmu na danych średniej wielkości Na rysunku nr 4.1 został przedstawiony wykres porównujący czasy wykonania algorytmów dla obu platform obliczeniowych. Wynika z nich, że algorytm działa od 4 do 8 razy szybciej na karcie graficznej **nVidia gtx 660** (dla kart kraficznych na serwerze wydziałowym **nvidia** oznacza to przyspieszenie 2-4 krotne).



Rysunek 4.1: Porównanie szybkości algorytmu Brandesa na GPU i CPU dla średnich grafów

Wyniki algorytmu na danych dużej wielkości Z kolei na obrazku nr 4.2 przedstawiono wykres porównujący czasy wykonania algorytmów dla obu platform obliczeniowych, ale tym razem na dużych grafach wejściowych. Z przeprowadzonych testów wynika, że dla takich dancyh uzyskano przyspieszenie około 10-krotne. Obrazek został przedstawiony poniżej:



Rysunek 4.2: Porównanie szybkości algorytmu Brandesa na GPU i CPU dla dużych grafów

Rozdział 5

Tabela z wynikami

Tabela z czasami Poniższa tabela przedstawia czasy (w sekundach), jakie osiągnięto podczas przeprowadzania testów na algorytmie Brandesa:

Facebook small GPU (wszystkie wyniki w tej tabeli zostały podane w sekundach)	3.669s
Facebook small CPU	13.862s
Facebook large GPU	6.151s
Facebook large CPU	38.69s
Wiki GPU	6.545s
Wiki CPU	34.445s
Email-Enron GPU	75.332
Email-Enron CPU	696.304s

Tabela 5.1: Porównanie wyników algorytmu Brandesa na poszczególnych grafach

Pliki z danymi wejściowymi Wszystkie powyższe testy będą dostępne do 30 czerwca 2014 roku na stronie internetowej http://students.mimuw.edu.pl/~kl291649/pwir

Rozdział 6

Zawartość rozwiązania

Pliki z kodami źródłowymi Dołączony do opracowania folder z kodami źródłowymi zawiera:

1. Wersję algorytmu na GPU z zaimplementowanymi wirtualnymi wierzchołkami (pliki brandes*)
2. Wersję algorytmu na CPU z zaimplementowanymi wirtualnymi wierzchołkami (pliki cpubrandes*)
3. Raport z wyniki w formaci PDF
4. Plik Makefile