CUDA - Raport

Władek Pałucki
wp418359@students.mimuw.edu.pl

28 listopada 2024

1 Podstawowe informacje

Napisałem wersję algorytmu z podanej pracy. Zdecydowałem się na podejście *Graph orientation* w wersji *Vertex-centric*. Wykorzystuję technikę optymalizacji *Binary Encoding of Induced Sub-graphs* opisaną w artykule, oraz trochę mniejszych usprawnień wykorzystujących pamięć *shared*.

2 Kontrola programu

2.1 Wczytanie i obróbka wejścia

Ten krok wykonywany jest w całości na CPU. Po wczytaniu wejścia w standardowy sposób przenumerowuję wierzchołki, tak, aby ich numery były z zakresu 0 - $(n-1)$ (gdzie n to liczba wierzchołków). Następnie, aby otrzymać graf skierowany, w którym stopień wierzchołków jest nie większy niż 1024 orientuję krawędzie tak, aby były skierowane z wierzchołka z o niższym stopniu do tego o wyższym stopniu. Dodatkowo, sortuję listy krawędzi, będzie to potrzebne przy tworzeniu indukowanego podgrafu.

2.2 Uruchamianie kernela

Kernel obliczający liczbę klik jest uruchamiany zaraz po przerzuceniu wyników poprzedniego punktu do pamięci karty graficznej. Za liczbę bloków odpowiada stała `MAX_INDUCED_SIZE`. W każdym bloku uruchamiana jest siatka dwuwymiarowa wątków o rozmiarach `GROUP_SIZE` × `(MAX_THREADS / GROUP_SIZE)` . Dodatkowo, z racji na to, że moja implementacji używa więcej pamięci *shared* niż standardowa dostępna wielkość, przed uruchomieniem kernela wykonywana jest instrukcja `cudaFuncSetAttribute` aby odpowiednio zwiększyć dostępną pamięć.

2.3 Struktura podziału pracy

Ponieważ cały indukowany podgraf dla każdego z wierzchołków nie zmieściłby się w pamięci, nawet przy kodowaniu binarnym, wykorzystuję optymalizację pamięci o której wspomniane

jest w sekcji 3.6 artykułu. Polega ona na tym, że na raz działa pewna liczba bloków (ściślej `MAX_INDUCED_SIZE bloków`), każdy z nich przetwarza indukowany podgraf zaczynający się z jednego wierzchołka, a po zakończeniu obliczeń dla tego kawałka grafu każdy blok dostaje kolejny wierzchołek, aż nie zostanie przetworzony cały graf.
W obrębie jednego bloku, wydzielamy grupy robocze, w każdej po `GROUP_SIZE` wątków. Jedna grupa przetwarza poddrzewo zakorzenione w jednym z sąsiadów wierzchołka bloku. Ponieważ sąsiadów wierzchołka startowego może być znacznie więcej niż maksymalna liczba grup, to grupy, podobnie jak blok na wyższym poziomie, są reużywane po skończeniu swoich obliczeń.

2.4 Działanie kernela

Działanie bloku dla pojedynczego wierzchołka dzieli się na dwa etapy:

1. Stworzenie podgrafu indukowanego
   Ta część kodu jest wydzielona do funkcji `induced_subgraph_extraction`.
   Na tym etapie wszystkie wątki działają oddzielnie, bez podziału na grupy, aby stworzyć podgraf. Jeden wątek ma przypisany jeden wierzchołek sąsiadujący z wierzchołkiem startowym bloku, i tworzy podgraf indukowany przez listę sąsiedztwa tego wierzchołka i wierzchołka startowego bloku. Cały graf indukowany jest zakdowany binarnie i zapisywany w pamięci globalnej karty graficznej.

2. Liczenie klik
   Po tym jak stworzony zostanie podgraf indukowany (zakodowany binarnie) każda grupa bloków zajmuje się przetwarzaniem kolejnych sąsiadów wierzchołka startowego. Podobnie jak opisane to zostało w artykule rekurencja zastąpiona jest stos. Na stosie w pamięci lokalnej wątku przechowywane są informacje potrzebne do wybierania kolejnych wierzchołków z listy sąsiedztwa. Na stosie w pamięci *shared* przechowywane jest obecny zbiór wierzchołków na którym pracuje grupa (I' w pseudokodzie w artykule). Przecięcie dwóch list sąsiedztwa realizowane jest przez grupę wspólnie. Każdy wątek w grupie oblicza operację `AND` na jednej pozycji zakodowanej listy sąsiedztwa i zapisanego na stosie zbioru wierzchołków. Zależnie od tego czy po tej operacji zbiór wierzchołków jest niepusty grupa przechodzi na niższy albo wyższy poziom rekurencji. Wyniki zbierane są w pamięci *shared* i dodawane do globalnego licznika wyników po skończeniu zliczania dla danego wierzchołka przez grupę.

3 Zastosowane Optymalizacje

- Binarne zakodowanie podgrafów
  Zakodowanie list sąsiedztwa jako liczb typu `int_32` znacznie zmniejsza użycie pamięci, dodatkowo pozwala na bardzo szybkie i czytelne przecięcie dwóch zbiorów, przez zwykłą binarną operację `AND`. Tą optymalizację realizuje praktycznie dokładnie tak, jak zostało to opisane w artykule. W pamięci globalnej zaalokowane jest statycznie miejsce na cały podgraf, a potem w to miejsce wpisywany jest obliczany przez bloki podgraf.

- Reużywanie bloków
  Tak jak wspomniałem wyżej wykorzystuję optymalizację pamięci polegającą na reużywaniu bloków do obliczania wielu wierzchołków, która opisana jest w sekcji 3.6 artykułu. W ten sposób ilość pamięci potrzebnej na zapisanie podgrafu jest stała, ponieważ na raz przetwarzane jest tylko stała liczba wierzchołków.

- Użycie pamięci *shared* przy obliczaniu podgrafów
  Przy obliczniu indukowanego podgrafu danego wierzchołka często potrzebna jest lista jego sąsiadów. W mojej implementacji ta lista jest zapisana w pamięci *shared* jako tablica `start_edges`. Lista jest przez wszystkie wątki razem do tej pamięci a potem, używana jest do obliczania indukowanych podgrafów.

- Użycie pamięci *shared* do przechowywania części informacji o rekurencji.
  Podczas symulowania rekurencji przy pomocy stosu, na każdym jej poziomie obliczany jest nowy zbiór wierzchołków na którym obecnie pracuje grupa. Ten zbiór zapisywany jest do tablicy `sub_graph_adj` w pamięci *shared*, ponieważ informacje tam się znajdujące będą często używane.

- Zapisywanie wyników cząstkowych w pamięci *shared*
  Aby uniknąć zapisywania wyników do globalnego licznika, (operacja ta wymaga użycia dodawania atomowego) wyniki każdej grupy zapisywane są do tablicy `results` w pamięci *shared* i dodawane do globalnego wyniku tylko po skończeniu obliczeń dla danego wierzchołka.