Projekt CUDA

Rafał Czarnecki

Optymalizacje

Zaimplementowałem wszystkie optymalizacje wspomniane w jednej z prac wspomnianej w treści zadania:

- Zmniejszam rozmiar grafu usuwając iteracyjnie wierzchołki mające tylko jednego sąsiada
- Sortuję wierzchołki po czasie wejścia w algorytmie BFS, aby zwiększyć ilość aktywnych wątków w warpie
- Używam stride-CSR do reprezentacji grafu, aby odczyty pamięci nie były od siebie zbyt oddalone

Zmieniłem też kod usuwający wierzchołki z jednym sąsiadem względem tego, który jest podany w pracy, ponieważ mam wrażenie, że może w nim występić race-condition (jeżeli graf zawiera gdzieś podgraf o pojedynczej krawędzi `u - v`, to oba wątki mogą być w różnych warpach i mogą wejść oba do ciała `if (v != -1)`, lub tylko jeden z nich w zależności od tego czy jeden z nich nie wykonał `adj[tadj[p]] = -1;` zanim drugi zdążył sprawdzić warunek w ifie).

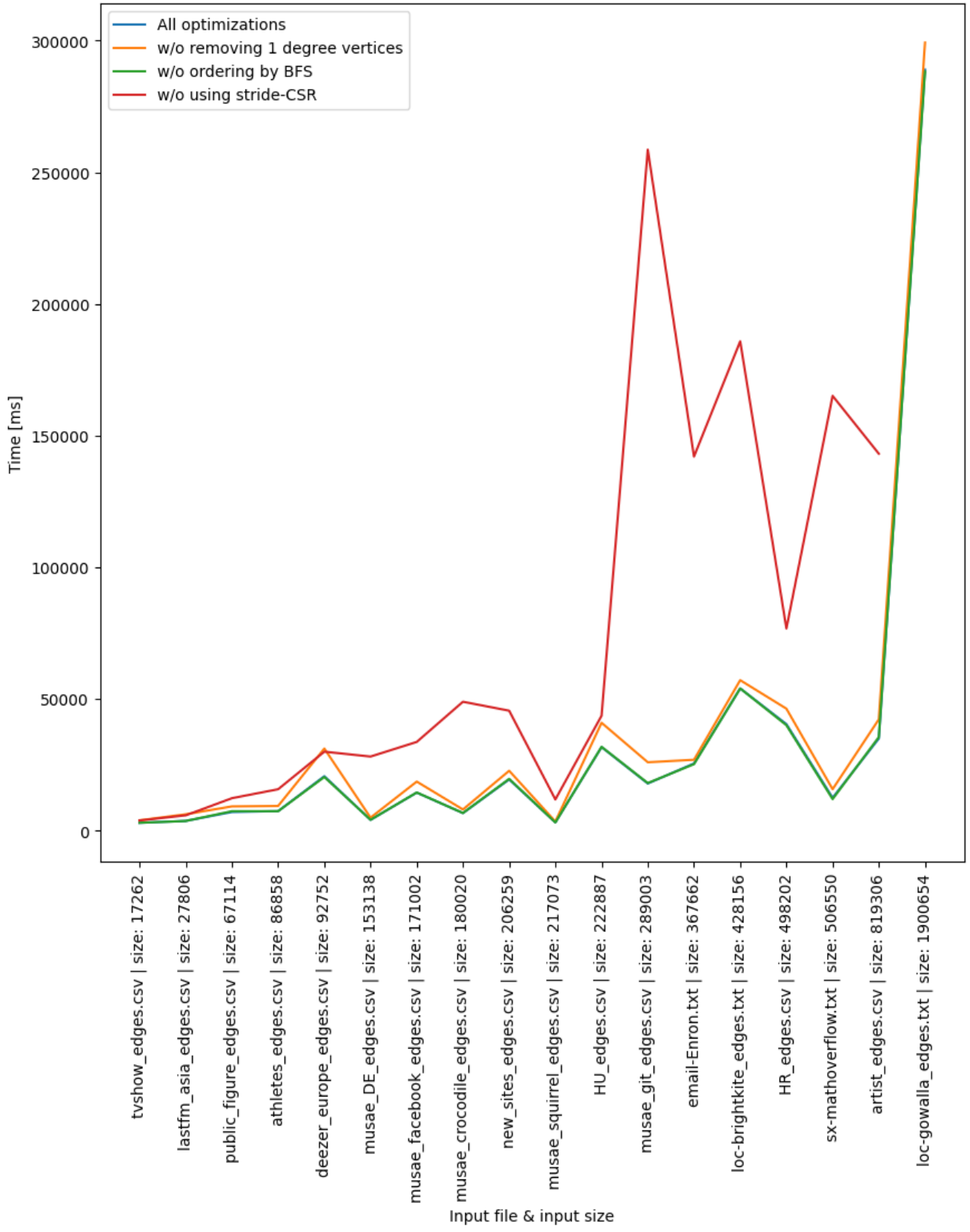
Kompilacja

`make` buduje najszybsza wersję, zgodną ze specyfikacją zadania. Dodatkowo na potrzeby testów można użyć następujących targetów:

- `test` - wypisuje postęp algorytmu i dodatkowe pomiary czasów (całość działania, działanie kerneli, działanie na hoście, czas transferów pamięci)
- `skip_1deg` - robi to co `test`, ale dodatkowo pomija usuwanie wierzchołków
- `skip_bfs` - robi to co `test`, ale dodatkowo pomija sortowanie wierzchołków po BFS
- `skip_stride` - robi to co `test`, ale dodatkowo pomija tworzenie wirtualnych wierzchołków i tworzenia stride'ów (czyli przechowuje graf w zwykłym CRS)

Testy

Przetestowałem wszystkie warianty na kilku grafach wziętych z http://snap.stanford.edu/data, a wyniki zamieszczam poniżej:



Z wyników wychodzi, że sortowanie po BFS nie pomaga zbyt mocno, ponieważ średnie przyspieszenie wynosi jedynie 1.0005.
Usuwanie wierzchołków daje lepszy efekt z przyspieszeniem 1.25.
Największy efekt dało użycie stride-CSR. Dla największego testu nie udało się go w ogóle wykonać, ze względu na limit czasu na entropy. Średnie przyspieszenie wynosi 4.4