Autor: Michał Bartosz Naruniec
Data: 9.01.2017r.
Przedmiot: Programowanie współbieżne, semestr zimowy 2016-2017
Uczelnia: Uniwersytet Warszawski
Zadanie: Zadanie nr 2, algorytm Brandesa

Raport dotyczy współbieżnej implementacji algorytmu Brandesa w C++14.

Algorytm został opisany w treści zadania.
Implementacja wykorzystuje klasę std::fstream do pobierania danych z i zapisywania do plików, klasę std::thread do manipulacji wątkami, klasę std::vector jako substytut tablic i list oraz klasy std::queue i std::stack do przeprowadzania przeszukiwań grafu. Poza zaaplikowaniem wprost udostępnionego algorytmu przy pomocy standardowych konstrukcji języka C++14 częścią zadania było również wprowadzenie własnych optymalizacji.

Pierwszą optymalizacją jest przeindeksowanie wierzchołków grafu. Ponieważ wejściowe indeksy są niesekwencyjne oraz mogą być bardzo duże, do zapamiętania ich potrzebna jest struktura słownikowa. Używana podczas obliczeń spowalniałaby je, więc podczas wczytywania wejścia mapuję identyfikatory wierzchołków z pierwszymi n liczbami naturalnymi, żeby móc używać wektorów. Po zakończeniu obliczeń przechodzę mapę iteratorem, aby odzyskać oryginalne identyfikatory i wypisać wyniki w odpowiedniej kolejności.

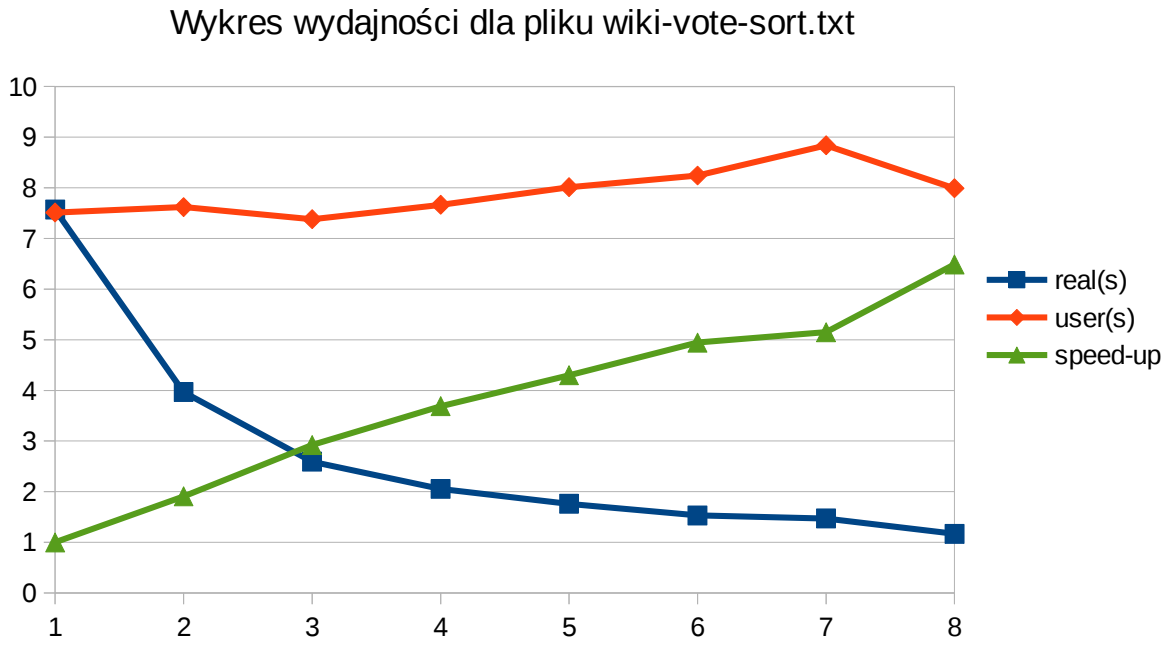
Wierzchołki, z których powinien startować dany BFS, przypisuję wątkom dynamicznie. Pozwala to na lepsze zrównoważenie obciążenia wątków, niż przypisanie statyczne, w którym jedne wątki mogłyby mieć dużo więcej do zrobienia niż inne ze względu na budowę grafu.

Z uwagi na wolne działanie mutexów i blokowanie jednych procesów przez inne oraz to że, zazwyczaj wierzchołków jest istotnie więcej niż wątków, każdy wątek posiada lokalny wektor BC, do którego wpisuje wyniki z przetwarzanych przez siebie samego wierzchołków. Dopiero kiedy proces zakończy obliczenia, przywłaszcza sobie mutex i dopisuje wyniki do tablicy globalnej. Wykorzystuję jedno zajęcie mutexa dla całej tablicy, żeby uniknąć częstych operacji na nim. Jest to skuteczniejsze, niż użycie wielu mutexów.

Do powyższych optymalizacji dochodzi również to, że wątki unikają tworzenia struktur bądź resetowania ich, jeżeli nie pozostał już do przetworzenia żaden wierzchołek lub aktualny start BFSa nie ma sąsiadów.

Oto tabela średniej wydajności dla pliku wejściowego wiki-vote-sort.txt na platformie students przy 3 próbach na każdą liczbę wątków m:

	1	2	3	4	5	6	7	8
real(s)	7,569	3,966	2,592	2,053	1,761	1,531	1,470	1,167
user(s)	7,513	7,620	7,380	7,663	8,013	8,243	8,840	7,990
sys(s)	0,050	0,053	0,070	0,063	0,083	0,043	0,143	0,093
speed-up	1,000	1,908	2,920	3,687	4,298	4,944	5,149	6,486


Wykres wydajności dla pliku wiki-vote-sort.txt
10
9
8
7
6
5
4
3
2
1
0
1
2
3
4
5
6
7
8
real(s)
user(s)
speed-up