PORR – Projekt
Etap 1.

Michał Cybulski, Piotr Grzegorski

30 listopada 2015r.

1 Zadanie

Zrównoleglone wyznaczanie najkrótszej ścieżki w grafie metodą aukcyjną oraz porównanie z algorytmem Dijkstry.

Zdecydowano o realizacji zadania przy pomocy technologii OpenMP.

2 Implementacja

Program został napisany w języku C przy wykorzystaniu interfejsu programowania aplikacji *OpenMP* (ang. *Open Multi-Processing*), który wspiera tworzenie aplikacji na systemy wieloprocesorowe z pamięcią współdzieloną.

Na cały program składają się:

- Generator grafów
- Algorytm Dijkstry
- Algorytm Aukcyjny

Implementacje generatora grafów oraz algorytmu Dijkstry opierają się na kodzie znalezionym w internecie, natomiast algorytm aukcyjny został zaimplementowany na potrzeby projektu od zera.

Współbieżna implementacja algorytmu aukcyjnego wyznaczenia najkrótszej ścieżki w grafie oparta jest o artykuł [2]. W standardowej wersji algorytm ten ma za zadanie znaleźć tylko jedną, najkrótszą ścieżkę pomiędzy dwoma wybranymi wierzchołkami grafu przestrzegając prostych zasad podczas poruszania się po nim. Taką wersję algorytmu trudno jest jednak zrównoleglić. Algorytm aukcyjny bardzo dobrze zrównolegla się, gdy zamiast skupienia się na szukaniu ścieżki między parą wierzchołków, rozpatrujemy przypadek szukania najkrótszych ścieżek prowadzących do jednego wierzchołka-celu wychodząc z wielu innych wierzchołków.

2.1 Argumenty wywołania

Aplikacja wymaga trzech liczbowych parametrów:

1. liczba wierzchołków grafu
2. liczba krawędzi grafu
3. maksymalna waga krawędzi

Dodatkowy, opcjonalny parametr to poziom logowania. Na potrzeby testów można podać wartość 10, która zaskutkuje wypisaniem tylko jednej linii podsumowania zawierającej same liczby.

2.2 Przebieg działania programu

Generuj graf ⇒ Wywołaj algorytm Dijkstry ⇒ Jeśli graf jest niespójny odrzucamy wynik, w przeciwnym wypadku wywołaj algorytm aukcyjny. ⇒ Koniec.

2.3 Zrównoleglenie algorytmu Dijkstry

Zrównoleglenie algorytmu Dijkstry polega na przydzieleniu każdemu z wątków puli wierzchołków. W każdej iteracji etap wybrania aktualnie najkrótszej ścieżki opiera się na sprawdzeniu każdej możliwej – i w tym momencie odpowiedni wątek sprawdza tylko podlegającą mu pulę wierzchołków. Potem informacje są synchronizowane i rozpoczyna się kolejna iteracja.

2.4 Sekwencyjny algorytm aukcyjny

Sekwencyjna wersja algorytmu opisana w książce [1] polega na iteracyjnym budowaniu listy wierzchołków, które składają się na szukaną najkrótszą ścieżkę. Poza listą wierzchołków utrzymywany jest też wektor cen/kosztów wierzchołków.

W każdej iteracji, w zależności od wyniku porównania kosztu ostatniego wierzchołka na liście z minimalną sumą kosztu wierzchołka sąsiedniego oraz wagi krawędzi do niego prowadzącej, do listy albo dodawany jest kolejny wierzchołek-sąsiad, albo z listy usuwany jest ostatni wierzchołek i uaktualniany jset wektor cen/kosztów. Wykonując kolejne iteracje algorytm zbliża się do szukanego wierzchołka i przerywany jest w momencie, gdy go znajdzie. W przypadku gdy nie istnieje żadna ścieżka prowadząca z wierzchołka początkowego do wierzchołka końcowego, algorytm będzie zapętlony w nieskończoność, a koszty wierzchołków będą dążyły do nieskończoności.

2.5 Zrównoleglenie algorytmu aukcyjnego

Gdy rozpatrzymy problem szukania ścieżek wychodzących z wielu wierzchołków, a kończących się w jednym wybranym wierzchołku, algorytm aukcyjny bardzo dobrze się zrównolega. Jedno z rozwiązań problemu zasugerowane w artykule [2] przewiduje wspólną synchroniczną pracę wielu wątków/procesów nad jednym współdzielonym wektorem kosztów.

Każdy wątek odpowiada za szukanie ścieżki rozpoczynającej się w wybranym wierzchołku porównując odpowiadający sobie koszt z sumami kosztów i wag krawędzi prowadzących do wierzchołków sąsiednich, a po zakończeniu iteracji, wyniki poszczególnych wątków są przetwarzane w celu budowania wspólnego wektora kosztów. Autor artykułu zauważa, że jeżeli jakiś wierzchołek jest w danej iteracji ostatnim wierzchołkiem kilku list wierzchołków pochodzących z różnych „punktów startowych", to wynik iteracji w każdym z tych wątków będzie taki sam - czyli takie samo wydłużenie, bądź skrócenie listy wierzchołków. Nie wystąpią więc problemy z wyścigami pomiędzy wątkami.

Jedyna sytuacja, w której może powstać konflikt, to sytuacja gdy pewien wierzchołek jest ostatnim wierzchołkiem listy jednego wątku, a dopiero ma stać się ostatnim wierzchołkiem innego. W przypadku, gdy okaże się, że wynikiem iteracji pierwszego wątku ma być skrócenie listy wierzchołków i zwiększenie kosztu rozpatrywanego wierzchołka, to operacja wydłużenia listy drugiego wątku powinna zostać wstrzymana do czasu kolejnej iteracji.

3 Testy

Testy obu algorytmów wykonywano dla tych samych grafów – zarówno generowanych losowo, jak i ze stałym ziarnem losowości. Wywołania sekwencyjne pozwoliło zrealizować ustawienie zmiennej środowiskowej `OMP_NUM_THREADS` na wartość 1. Testy były wykonywane w pętli po 1000 uruchomień (wyjątek stanowią testy dla grafów o 10000 wierzchołków – te uruchamiane były po 500 razy).

Wynik testu stanowiła mediana czasu wykonania – pozwoliło to ograniczyć wpływ sytuacji wyjątkowych, np. dodatkowego chwilowego obciążenia systemu, na uzyskane rezultaty.

W toku testowania wykazano, że wpływ na wydajność programu miały niejawne synchronizacje na funkcji logującej, zdecydowano więc o wykomentowaniu jej wywołań na potrzeby testowania. Poniższe wyniki uzyskano po poprawce.

3.1 Wyniki czasowe

Mierzony czas wywołania to czas procesora. Czasy należy więc przedzielić przez liczbę wątków, by otrzymać czasy rzeczywiste. Tabelka z pomiarami rzeczywistymi znajduje się niżej.

	Jeden wątek		Cztery wątki	
L. wierzch.	Dijkstry	aukcyjny	Dijkstry	aukcyjny
100	0,0001	0,0023	0,0011	0,0058
1000	0,0072	0,3744	0,0101	0,3851
10000	0,8392	28,0580	0,8705	27,9950

Rysunek 1: Tabelka median czasów procesora

	Jeden wątek		Cztery wątki	
L. wierzch.	Dijkstry	aukcyjny	Dijkstry	aukcyjny
100	0,0001	0,0023	0,0003	0,0015
1000	0,0072	0,3744	0,0025	0,0963
10000	0,8392	28,0580	0,2176	6,9988

Rysunek 2: Tabelka median czasów rzeczywistych

4 Wnioski

Zgodnie z oczekiwaniami wydajność po zrównolegleniu obliczeń wzrosła dla nietrywialnych przypadków. Przy małych grafach narzut na synchronizację kilku wątków (widniejące w logach profilera czasy oczekiwania na barierach OpenMP – `gomp_team_barrier_wait_end` oraz `gomp_barrier_wait_end`) dominowały nad czasem obliczeń.

Literatura

[1] Dimitri P. Bertsekas *Network Optimization: Continuous and Discrete Models* Athena Scientific 1998.

[2] Dimitri P. Bertsekas, *An Auction Algorithm For Shortest Paths.* SIAM Journal on Optimization, Vol. 1, No. 4, pp. 425-447 November 1991.