Wydział Fizyki i Informatyki Stosowanej

Systemy równoległe i rozproszone

Drzewa wszystkich najkrótszych ścieżek - algorytm Dijkstry
Aplikacja PGAS

Arkadiusz Kasprzak
Aleksandra Poręba

6 czerwca 2020

Spis treści

1 Wstęp

Niniejszy dokument stanowi dokumentację drugiego projektu wykonanego w ramach przedmiotu *Systemy równoległe i rozproszone*. Jego tematem było stworzenie, z wykorzystaniem technologii *Unified Parallel C*, aplikacji implementującej algorytm Dijkstry.

1.1 Algorytm Dijkstry

Zaimplementowany w ramach projektu algorytm Dijkstry jest algorytmem poszukiwania najkrótszych ścieżek z wybranego wierzchołka do wszystkich pozostałych w grafie (skierowanym lub nieskierowanym) o nieujemnych wagach krawędzi. Algorytm ten oparty jest na metodzie zachłannej: w każdym kroku wybierany jest wierzchołek, do którego dotarcie wiąże się z najmniejszym kosztem.

```
V − zbior wierzcholkow
E − zbior krawedzi z wagami
v − wierzcholek startowy

Dijkstra(V, E, v)
    Q := ∅;
    p := −1;
    d := ∞
    d(v) := 0
    dopoki Q != V wykonaj
        wybierz spoza zbioru Q wierzcholek u o najmniejszym koszcie d(u)
        Q := Q ∪ {u}
        dla kazdego sasiada w wierzcholka u spoza zbioru Q wykonaj
            jesli d(w) > d(u) + E(w, u) to
                d (w) :=  d(u) + E(w, u)
                p(w) := u
```

Listing 1: Pseudokod klasycznego algorytmu Dijkstry.

W czasie wykonywania klasyczny algorytm Dijkstry operuje na dwóch tablicach. Jedna z nich, oznaczana zwykle literą `d`, przechowuje koszty dotarcia z wierzchołka źródłowego do każdego z wierzchołków w badanym grafie. Druga, oznaczana litera `p`, przechowuje informacje o poprzedniku każdego z wierzchołków - ta informacja pozwala, po zakończeniu działania algorytmu, odtworzyć ścieżkę z wierzchołka źródłowego do każdego wierzchołka w grafie. Ponadto w algorytmie przechowuje się zwykle zbiór (oznaczany np. literą `Q`) wierzchołków, które zostały już przetworzone. Może on być zaimplementowany np. za pomocą tablicy wartości logicznych, co pozwala na łatwe współdzielenie go między wieloma procesami.

1.2 Zastosowane technologie

Przygotowana aplikacja napisana została w języku programowania C, z wykorzystaniem technologii UPC (*Unified Parallel C*). Projekt korzysta w znacznym stopniu z dostarczanej przez *UPC* funkcjonalności rozproszonej pamięci współdzielonej, w tym z tzw. alokacji katalogowej rozproszonych tablic. Wykorzystane zostały ponadto: pętla `upc_forall` oraz operacje kolektywne (nagłówek `upc_collective.h`). Struktura plików wejściowych i wyjściowych pozostała taka sama jak w przypadku aplikacji MPI. System budowania oparty został o narzędzie Make oraz prosty skrypt powłoki, dzięki czemu uruchomienie programu w środowisku pracowni wydziału nie powinno sprawiać trudności.

2 Struktura projektu

Poniżej została opisana budowa projektu. Dokumentacja samego kodu przygotowana została za pomocą narzędzia `Doxygen`. Instrukcja jej wygenerowania została przedstawiona w punkcie 4.1.

Program został podzielony na poszczególne funkcjonalności:

- działanie algorytmu, czyli jego inicjalizacja, szukanie najmniejszej odległości, aktualizacja ścieżek, czy zapis wyników do pliku,
- dodatkowe narzędzia związane z działaniem programu, a nie z algorytmem, takie jak odczyt danych, alokacja pamięci.

Zostały również przygotowane pomocnicze struktury do przetwarzania danych, takie jak:

- `VertexData` reprezentująca poszczególny wierzchołek, zawiera jego indeks w macierzy sąsiedztwa oraz koszt dotarcia do niego,
- `ColumnData` przechowująca dane o poszczególnej kolumnie, jej indeksie w macierzy sąsiedztwa oraz wskaźniku na początek kolumny,
- `ColumnsToProcess` zawierająca listę kolumn dla danego procesu, ich ilość oraz rozmiar.

3 Działanie projektu

W poniższej części dokumentacji zostało omówione działanie programu implementującego algorytm Dijkstry z wykorzystaniem technologii *UPC*. Składa się on z następujących części:

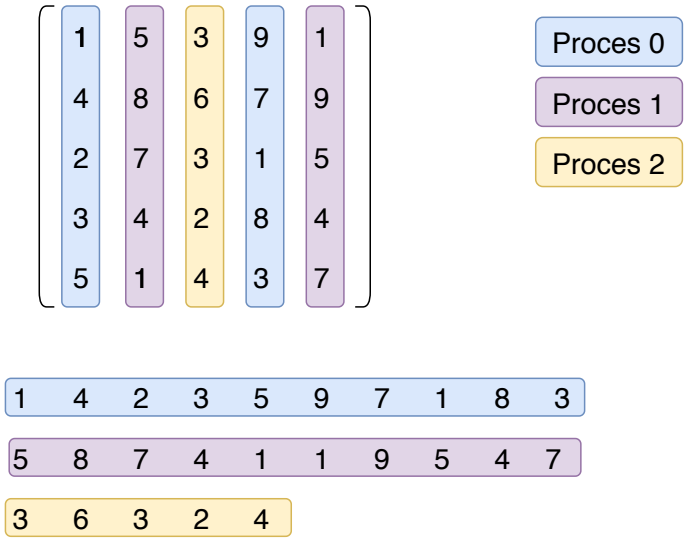
- wczytanie danych wejściowych - początkowego wierzchołka oraz macierzy sąsiedztwa i jej rozmiaru,
- zapis kolumn do odpowiednich procesów,
- inicjalizacja współdzielonych danych,
- obliczanie odległości i ścieżek,
- zapis wyników do pliku oraz zakończenie działania.

3.1 Inicjalizacja

Działanie programu rozpoczyna się od wczytania danych wejściowych przez proces 0, takich jak wierzchołek, do którego będziemy szukać ścieżek oraz rozmiar macierzy sąsiedztwa przetwarzanego grafu. W przypadku braku tych informacji program kończy działanie.

Jeśli dane są poprawne, następuje alokacja pamięci z wykorzystaniem funkcji *UPC* - `upc_global_alloc`. Najpierw proces 0 przygotowuje współdzielone dane, takie jak tablica z odległościami, z poprzednikami wierzchołków oraz z informacją, czy wierzchołek był już analizowany. Każdy proces będzie je wypełniał tylko dla swoich wierzchołków.

Następnie, dla każdego procesu następuje alokacja pamięci za pomocą `upc_alloc` na odpowiednią liczbę kolumn. Dzięki tej funkcjonalności pamięć zostanie zaalokowana tylko dla danego procesu, w jego sąsiedztwie (nie jest współdzielona). Wskaźniki te zapisywane do współdzielonego katalogu - każdy proces ma tam swoje miejsce. Jeśli jest więcej kolumn niż procesów, procesy w kolejności od 0 otrzymują dodatkowe kolumny, które zapisywane są w pamięci ciągłej. Przykładowe rozdysponowanie zostało przedstawione poniżej.



Rysunek 1: Przydział kolumn do poszczególnych procesów oraz ich organizacja w pamięci. Przedstawiony przykład pokazuje sytuację, gdy macierz ma wymiar 5, a procesów jest 3 - do pierwszych dwóch została przydzielona dodatkowa kolumna.

Gdy pamięć została zaalokowana proces 0 wypełnia odpowiednie kolumny wartościami z macierzy sąsiedztwa. Wskaźniki do poszczególnych kolumn zostają zapisane do pomocniczej struktury `ColumnsToProcess`, podobnie jak ich ilość, oraz rozmiar pojedynczej kolumny. Ułatwi to dostęp do danych w czasie działania algorytmu.

Na koniec procesu inicjalizacji wypisana zostaje informacja ile kolumn otrzymał dany proces, oraz ich indeksy. Przykładowa informacja znajduje się poniżej.

```
Proces 0 posiada 4 kolumn o indeksach: 0, 3,  6,  9,
Proces 1 posiada 3 kolumn o indeksach: 1, 4,  7,
Proces 2 posiada 3 kolumn o indeksach: 2, 5,  8,
```

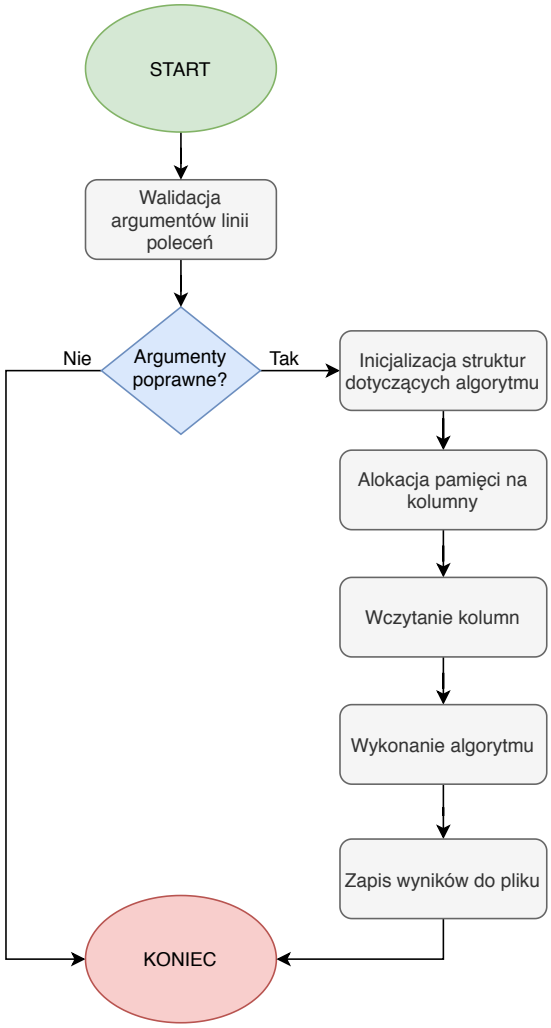
3.2 Implementacja i zakończenie algorytmu

Każdy proces przetwarza swoje przydzielone kolumny. Na samym początku znajdowany jest wierzchołek, do którego odległość jest najmniejsza wśród wszystkich przydzielonych kolumn. Do przeglądania kolejnych elementów została użyta pętla `upc_forall`, która umożliwia iterowanie po elementach znajdujących się w sąsiedztwie procesu.

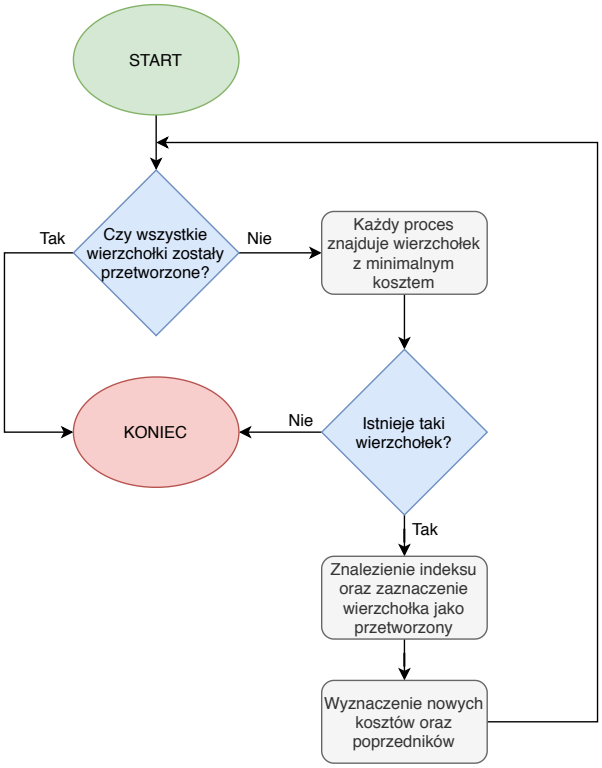
Najmniejsze odległości zapisywane są w współdzielonej tablicy `localMin`. Gdy wszystkie procesy zakończą tą czynność, za pomocą funkcjonalności `upc_all_reduce` znajdowane jest minimum i zapisywane do współdzielonego `globalMin`. Teraz, przeszukując tablicę `localMin` możemy wyznaczyć indeks wierzchołka z najmniejszą odległością. Na koniec głównej pętli każdy proces aktualizuje tablicę kosztów oraz poprzedników.

Czynności te są wykonywane, aż nie uda się odnaleźć wierzchołka z minimalną odległością lub wszystkie wierzchołki zostaną przetworzone. Sprawdzane jest to za pomocą `upc_all_reduce`, które sumuje wartości w tablicy z przetworzonymi wierzchołkami.

Na sam koniec znalezione odległości oraz ścieżki zostają zapisane do pliku `resultsUPC.txt`.



Rysunek 2: Schemat blokowy ilustrujący przebieg programu.



Rysunek 3: Schemat blokowy ilustrujący przebieg algorytmu.

4 Obsługa programu

Niniejsza część dokumentacji przedstawia informacje związane z kompilacją i uruchomieniem projektu.

4.1 Obsługa projektu na pracowni WFiIS

W celu ułatwienia kompilacji i uruchomienia projektu na pracowni Wydziału Fizyki i Informatyki Stosowanej przygotowane zostały dedykowane pliki `Makefile` oraz skrypty do obsługi programu. Znajdują się one w katalogu `build_uni`. Aby dokonać koniecznej konfiguracji środowiska oraz, następnie, kompilacji programu należy wykonać następujące polecenia:

```
cd build_uni
source setup.sh
make
```

Plik `setup.sh` pozwala na skonfigurowanie zmiennych środowiskowych oraz produkuje plik `nodes`. Następnie, aby uruchomić program należy użyć polecenia:

```
make run VERTEX=V FILE=F PROC_COUNT=P HOSTS=H
```

gdzie `V` oznacza wierzchołek startowy (domyślnie: 0), `F` to ścieżka do pliku z danymi (domyślnie: `../data/graph.dat`), `P` to liczba procesów, które zostaną użyte do wykonania algorytmu (domyślnie: 1) a `H` to ścieżka do pliku z węzłami (domyślnie: `nodes`). Inne opcje udostępnione przez plik `Makefile` to:

- `make build` - wykonanie procesu kompilacji - tożsame z poleceniem `make` bez argumentów
- `make run` - uruchomienie programu z domyślnymi opcjami
- `make clean` - przywraca projekt do stanu początkowego
- `make docs` - tworzy dokumentację za pomocą narzędzia Doxygen
- `make install` - kopiuje plik wykonywalny do aktualnego katalogu

Wynik działania programu zapisywany jest do pliku `resultsUPC.txt`.

4.2 Dane wejściowe

Program jako jedną z danych wejściowych przyjmuje ścieżkę do pliku z grafem zapisanym w postaci macierzy sąsiedztwa. W pierwszej linii pliku powinna znajdować się liczba wierzchołków kodowanego grafu. Kolejne linie powinny odpowiadać wierszom macierzy sąsiedztwa. Wagi powinny być zapisane w formie dodatnich liczb rzeczywistych.

Listing 2: Przykładowy plik wejściowy.

```
7
0.0000 5.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 7.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000
0.0000 4.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 2.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 10.0000
0.0000 0.0000 0.0000 0.0000 0.0000 2.0000 0.0000
```

4.3 Format pliku wynikowego

Wyniki wygenerowane przez program zapisywane są do pliku tekstowego o nazwie `resultsUPC.txt`. Wyprodukowany plik ma następujący format:

Listing 3: Przykładowy plik z wynikami

```
============ RESULTS ============
Distance from vertex 0 to 0: 0.00
Distance from vertex 0 to 1: 5.00
Distance from vertex 0 to 2: 12.00
Distance from vertex 0 to 3: 13.00
Distance from vertex 0 to 4: inf
Distance from vertex 0 to 5: inf
Distance from vertex 0 to 6: inf
============= PATHS =============
0,
0, 1,
0, 1, 2,
0, 1, 2, 3,
Vertex 4 unreachable from source vertex.
Vertex 5 unreachable from source vertex.
Vertex 6 unreachable from source vertex.
```

W pierwszej części pliku znajdują się wyliczone koszty dotarcia z wierzchołka źródłowego do pozostałych wierzchołków w grafie. Jeśli dotarcie nie było możliwe, zapisywana jest wartość `inf`. W drugiej części pliku znajdują się wyliczone ścieżki - ponownie, jeśli wierzchołek był nieosiągalny, jest to odnotowywane.

Literatura

[1] dr inż. Piotr Gronek - wykład z przedmiotu *Systemy równoległe i rozproszone* prowadzony na Wydziale Fizyki i Informatyki Stosowanej AGH

[2] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar *Introduction to Parallel Computing, Second Edition.* Addison-Wesley, 2003.

[3] Tarek El-Ghazawi, William Carlson, Thomas Sterling, Katherine Yelick *UPC: Distributed Shared Memory Programming*

[4] Dokumentacja projektu *Berkeley UPC* - `https://upc.lbl.gov/docs/` (dostęp: 01.06.2020)

[5] UPC Consortium - *UPC Required Library Specifications Version 1.3*