Minimalne drzewo rozpinające - algorytm Prim'a przy użyciu UPC++.

Daria Kokot, Bartosz Konopka

Spis treści

1 Wprowadzenie

1.1 Wstęp teoretyczny

Programowanie równoległe jest to sposób pisania programów, w których wiele zadań wykonywanych jest jednocześnie. Zamiast jednego ciągu instrukcji (jak w tradycyjnym programowaniu sekwencyjnym), istnieje wiele procesów lub wątków pracujących równolegle.

UPC++, czyli Unified Parallel C++, to biblioteka programowania równoległego oparta na języku C++, działająca w modelu PGAS (Partitioned Global Address Space). Umożliwia wielu procesom współdzielenie i bezpośredni dostęp do rozproszonej pamięci w środowiskach klastrowych i superkomputerowych. UPC++ łączy wysoką wydajność z wygodą programowania, oferując niskopoziomowe mechanizmy komunikacji, takie jak zdalne odczyty i zapisy pamięci, oraz asynchroniczne wywołania funkcji zdalnych (remote procedure calls – RPC).

W celu zrozumienia Algorytmu Prima, niezbędne jest zrozumienie niektórych zagadnień związanych z teorią grafów:

- **Graf nieskierowany** - graf, w którym krawędzie nie mają określonego kierunku, tzn. połączenie między dwoma wierzchołkami działa w obie strony.
- **Graf spójny** - graf, w którym istnieje ścieżka łącząca dowolną parę wierzchołków.
- **Graf acykliczny** - graf, w którym nie występują żadne cykle, czyli ścieżki zaczynające i kończące się w tym samym wierzchołku bez powtarzania krawędzi.
- **Graf ważony** - graf, w którym każda z krawędzi posiada przypisaną wagę.
- **Drzewo** - graf nieskierowany, który jest jednocześnie spójny i acykliczny. Oznacza to, że można przejść z dowolnego wierzchołka do innego tylko jednym, unikalnym sposobem, bez możliwości poruszania się „w kółko".
- **Drzewo rozpinające** - drzewo zawierające wszystkie wierzchołki danego grafu G, przy czym jego krawędzie stanowią podzbiór krawędzi grafu
- **Minimalne drzewo rozpinające** - drzewo rozpinające o najmniejszej możliwej sumie wag krawędzi.

Algorytm Prima to algorytm zachłanny służący do wyznaczania minimalnego drzewa rozpinającego. Działa na grafie nieskierowanym i spójnym, czyli takim, w którym każda para wierzchołków jest połączona pewną ścieżką, a krawędzie nie mają ustalonego kierunku. Algorytm znajduje podzbiór krawędzi $E\ prim \subseteq E$, taki że graf pozostaje spójny, a łączny koszt (waga) wszystkich wybranych krawędzi jest najmniejszy z możliwych.

1.2 Sekwencyjne działanie algorytmu Prima

Wejście: graf nieskierowany, spójny i ważony (każda krawędź ma przypisaną wagę).
Wyjście: minimalne drzewo rozpinające (MST – Minimum Spanning Tree).

Kroki algorytmu:

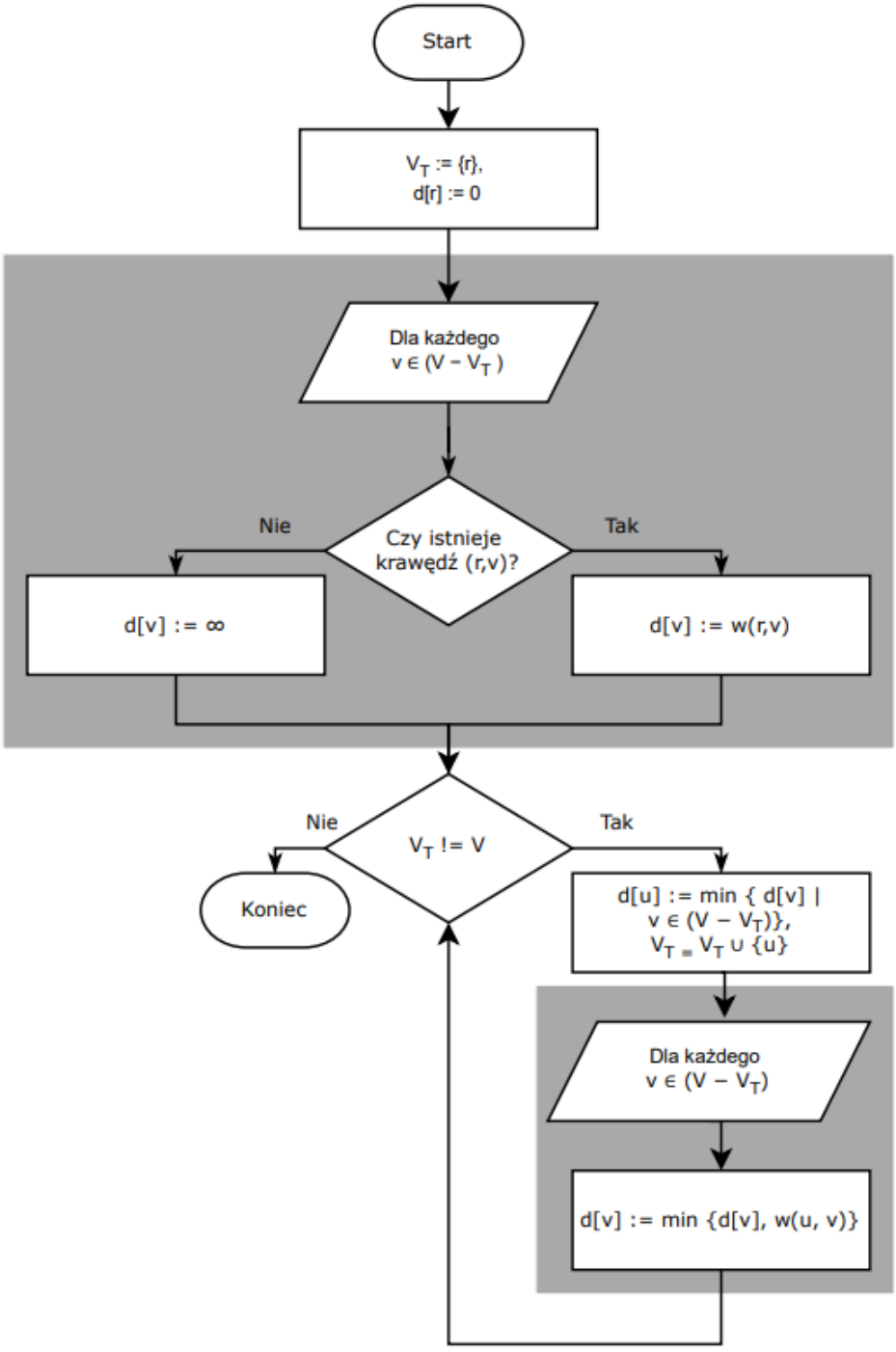
1. Wybierz dowolny wierzchołek początkowy i dodaj go do zbioru MST.
2. Znajdź wszystkie krawędzie wychodzące z MST (czyli takie, które łączą już odwiedzony wierzchołek z nieodwiedzonym).
3. Wybierz krawędź o najmniejszej wadze spośród dostępnych (której jeden koniec jest w MST, a drugi jeszcze nie).
4. Dodaj tę krawędź i jej końcowy wierzchołek do MST.
5. Powtarzaj kroki 2–4, aż wszystkie wierzchołki znajdą się w MST (czyli drzewo będzie zawierać n - *1* krawędzi dla n wierzchołków).

```
1.   procedure PRIM_MST(V, E, w, r)
2.   begin
3.       V_T := {r};
4.       d[r] := 0;
5.       for all v ∈ (V − V_T) do
6.           if edge (r, v) exists set d[v] := w(r, v);
7.           else set d[v] := ∞;
8.       while V_T ≠ V do
9.       begin
10.          find a vertex u such that d[u] := min{d[v]|v ∈ (V − V_T)};
11.          V_T := V_T ∪ {u};
12.          for all v ∈ (V − V_T) do
13.              d[v] := min{d[v], w(u, v)};
14.      endwhile
15.  end PRIM_MST
```

Rysunek 1: Algorytm przedstawiający działanie algorytmu Prima, Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, Introduction to Parallel Computing, Addison-Wesley / Pearson Education 2003 [1]

Wyjaśnienie symobli w algorytmie:

- V – zbiór wszystkich wierzchołków grafu.
- E – zbiór wszystkich krawędzi grafu.
- $w(u, v)$ – waga (koszt) krawędzi między wierzchołkami u i v.
- r – wierzchołek początkowy, od którego zaczyna się budowanie drzewa rozpinającego.
- VT – zbiór wierzchołków już włączonych do drzewa minimalnego (MST).
- $d[v]$ – minimalny koszt krawędzi łączącej wierzchołek v z dowolnym wierzchołkiem należącym do VT (czyli do drzewa).
- u – wierzchołek spoza VT, który ma najmniejszy koszt przyłączenia do VT (czyli minimalne $d[u]$).



Rysunek 2: Schemat blokowy algorytmu Prima

1.3 Równoległe działanie algorytmu Prima

W klasycznej wersji algorytmu Prima, w każdej iteracji wyszukiwana jest najmniejsza krawędź wychodząca z już zbudowanego drzewa rozpinającego (MST), a następnie dodawany jest nowy wierzchołek. Algorytm ten można zrównoleglić poprzez podział macierzy sąsiedztwa grafu pomiędzy procesy w środowisku UPC++. Każdy proces pracuje na własnej części danych, które umieszczone są w rozproszonej przestrzeni adresowej.

W każdej iteracji:

- Każdy proces przeszukuje lokalnie swoje przydzielone krawędzie w celu znalezienia lokalnego minimum.
- Wykonywana jest operacja `upcxx::reduce_all` w celu znalezienia globalnego minimum.
- Globalnie wybrany wierzchołek jest rozgłaszany do wszystkich procesów z użyciem mechanizmu RPC.
- Aktualizowane są struktury danych `mst[v]` i `mst_w[v]` przechowujące poprzedników i wagi krawędzi w MST.

Algorithm 1 Równoległy algorytm Prima z UPC++ (pseudokod)

```
Initialize UPC++
Get current process rank rank and total number of processes size
Distribute graph adjacency matrix among processes
Initialize MST data structures: mst_prev, mst_weight, in_mst
for step = 0 to num_vertices − 1 do
    local_min ← find minimal edge in local graph part, excluding MST vertices
    [global_min, owner] ← reduce_all to find global minimum and owner process
    Broadcast global_min from owner to all processes
    Update MST data structures with global_min
end for
Finalize UPC++
```

UPC++ pozwala na wydajną komunikację bez konieczności kopiowania danych, co czyni go bardzo użytecznym w przypadku obliczeń równoległych na dużych grafach.

2 Budowa projektu

2.1 Technologia

Technologia **UPC++** (Unified Parallel C++) to biblioteka programistyczna oparta na języku C++, która umożliwia tworzenie wydajnych programów równoległych działających w modelu PGAS (Partitioned Global Address Space). Jej zastosowanie w rozwiązaniu **algorytmu Prima** – służącego do znajdowania minimalnego drzewa rozpinającego (MST) w grafie – pozwala przyspieszyć obliczenia na dużych grafach dzięki równoległości i współdzielonej przestrzeni adresowej.

2.2 Inicjalizacja UPC++, odczytanie danych z pliku

Program rozpoczyna się od inicjalizacji biblioteki UPC++ oraz wczytania danych wejściowych z pliku tekstowego, zawierającego rozmiar macierzy sąsiedztwa oraz jej elementy.

Listing 1: Inicjalizacja UPC++ i wczytanie danych

```
upcxx::init();
int rank = upcxx::rank_me();
int ranks = upcxx::rank_n();

std::ifstream infile("Input/Matrix.txt");
if (!infile.is_open()) {
    if (rank == 0) {
        std::cerr << "Failed to open input file." << std::endl;
    }
    upcxx::finalize();
    return 1;
}

int mSize;
infile >> mSize;
std::vector<int> matrix(mSize * mSize, 0);
```

W przypadku braku danych wejściowych generowana jest losowa macierz symetryczna, reprezentująca graf nieskierowany.

2.3 Podział macierzy sąsiedztwa na węzły, inicjalizacja macierzy sąsiedztwa

Algorytm Prima służy do znajdowania minimalnego drzewa rozpinającego (MST) w grafie ważonym. W tej implementacji:

- Każdy proces otrzymuje fragment wierzchołków (kolumn) do analizy.
- Macierz sąsiedztwa przesyłana jest do pamięci globalnej przy użyciu wskaźników globalnych UPC++.
- Pętla główna wykonuje iteracje dla każdej z $mSize - 1$ krawędzi MST.

Listing 2: Przygotowanie danych i pętla główna MST

```
int vertices_per_rank = (mSize + ranks - 1) / ranks;
int start = rank * vertices_per_rank;
int end = std::min(mSize, start + vertices_per_rank);

std::vector<bool> selected(mSize, false);
selected[0] = true;

upcxx::global_ptr<int> g_matrix = upcxx::new_array<int>(mSize *
   mSize);
upcxx::rput(matrix.data(), g_matrix, mSize * mSize).wait();
upcxx::barrier();
```

2.4 Wybór minimalnej krawędzi przez wszystkie procesy

Każdy proces lokalnie przeszukuje swoje fragmenty w celu znalezienia najlżejszej krawędzi łączącej wybrany już wierzchołek z nieodwiedzonym. Następnie odbywa się globalna redukcja do wyboru najlepszego z lokalnych wyników.

Listing 3: Wyszukiwanie i redukcja krawędzi

```
int local_min_weight = INF;
int local_u = -1, local_v = -1;

for (int i = 0; i < mSize; ++i) {
    if (selected[i]) {
        for (int j = start; j < end; ++j) {
            if (!selected[j]) {
                int weight = upcxx::rget(g_matrix + i * mSize + j
                   ).wait();
                if (weight != 0 && weight < local_min_weight) {
                    local_min_weight = weight;
                    local_u = i;
                    local_v = j;
                }
            }
        }
    }
}

std::pair<int, int> local_data{local_min_weight, rank};
std::pair<int, int> global_data;
upcxx::reduce_all(&local_data, &global_data, 1,
    [](const std::pair<int,int> &a, const std::pair<int,int> &b)
       {
        return (a.first < b.first) ? a : b;
    }, upcxx::world()).wait();
```

2.5 Aktualizacja drzewa MST i synchronizacja

Po wyborze krawędzi przez najlepszy proces, dane są rozgłaszane do pozostałych, a nowy wierzchołek dodawany jest do zbioru wybranych. Operacja powtarza się aż do utworzenia pełnego drzewa MST.

Listing 4: Aktualizacja MST i synchronizacja

```
int chosen_u = upcxx::broadcast(local_u, global_data.second).wait
    ();
int chosen_v = upcxx::broadcast(local_v, global_data.second).wait
    ();
int chosen_weight = upcxx::broadcast(local_min_weight,
    global_data.second).wait();

selected[chosen_v] = true;
total_weight += chosen_weight;
mst_edges.emplace_back(chosen_u, chosen_v, chosen_weight);
++edge_count;

upcxx::barrier();
```

2.6 Zapis wyników do pliku i ich wyświetlenie

Poniższy fragment kodu odpowiada za zapis wyników obliczeń algorytmu Prima (minimalne drzewo rozpinające, MST) do pliku oraz wyświetlenie wyników na ekranie. Opis wykonania tych operacji jest następujący:

1. **Sprawdzenie ranku procesu:**
   - Tylko proces o `rank == 0` (zwykle proces nadrzędny lub główny) wykonuje zapis wyników.
   - Pozostałe procesy (o innych rangach) nie biorą udziału w zapisie do pliku ani wyświetlaniu wyników.
2. **Otwieranie pliku do zapisu:**
   - Otwierany jest plik `Output/Output.txt` w trybie `app`, co oznacza, że wyniki będą dopisywane na końcu pliku bez nadpisywania poprzednich wyników.
   - Jeśli plik nie może zostać otwarty (np. z powodu braku katalogu lub uprawnień), nic nie jest zapisywane i może zostać zgłoszony błąd.
3. **Zapis do pliku:**
   - Jeżeli plik został pomyślnie otwarty, zapisywane są następujące dane:
     - `total_weight` — całkowita waga minimalnego drzewa rozpinającego.
     - Lista krawędzi tworzących MST — każda krawędź zawiera numer wierzchołków oraz wagę.
     - Dodatkowe informacje: liczba procesorów (`ranks`), liczba wierzchołków (`mSize`), czas wykonania algorytmu (`exec_time`).
   - Plik jest następnie zamykany.

4. **Wyświetlanie wyników na ekranie:**

   - Proces o `rank == 0` może również opcjonalnie wypisać wyniki na standardowe wyjście, co bywa pomocne przy uruchomieniach testowych.
   - Wypisywane dane mogą zawierać:
     - Krawędzie MST z ich wagami.
     - Całkowitą wagę drzewa.
     - Czas wykonania obliczeń.

Listing 5: Zapis wyników do pliku przez proces 0

```
if (rank == 0) {
    std::ofstream outfile("Output/Output.txt", std::ios::app);
    outfile << "-------------------------------------------------\
       n";
    outfile << "Created Matrix for " << ranks << " ranks:\n";
    for (int i = 0; i < mSize; ++i) {
        for (int j = 0; j < mSize; ++j) {
            outfile << std::setw(4) << matrix[i * mSize + j];
        }
        outfile << "\n";
    }
    outfile << "\nThe min weight is " << total_weight << "\n";
    for (const auto& [u, v, w] : mst_edges) {
        outfile << "Edge " << v << " " << u << " - weight " << w
           << "\n";
    }
    outfile << "Processors: " << ranks << "\n";
    outfile << "Vertices: " << mSize << "\n";
    outfile << "Execution Time: " << std::fixed
            << std::setprecision(8) << exec_time.count() << " s\n
               ";
    outfile << "Total Weight: " << total_weight << "\n";
    outfile.close();
}
```

3 Instrukcja obsługi projektu

Projekt przygotowany jest w ten sposób aby działał prawidłowo w sali 204 w *stud-204*. Znajdując się w katalogu głównym projektu można skompilować i uruchomić program wykorzystując plik *Makefile*, lub skorzystać z gotowych skryptów *run.sh* i *clean.sh*.

W przypadku uruchomiania projektu ręcznie za pomocą *Makefile* neleży:

1. Wykonać poleceni3: *source optnfsconfigsource_upcxx_2023.9.sh* w celu zainicjalizowania UPC++.

2. Wykonać polecenie: */opt/nfs/config/station204_name_list.sh 1 16 > nodes* zapisujące do pliku *nodes* informacje o dostępnych węzłach.

3. Wykonać polecenie: *make* kompilujące program do plików wykonywalnch.

4. Wpisać w plik *InputMatrix.txt* wyłącznie jedną liczbę całkowitą w pierwszej linijce, definiującą rozmiar macierzy sąsiedztwa.

5. Wykonać polecenie: *make run ARG=ilosc_wezlow* odpowiedzialne za uruchomienie programu. Do argumentu ARG należy wpisać ilość procesów na której program będzie się wykonywał.

6. Wynik programu widoczny będzie w pliku *Output/Output.txt*

7. Wykonać polecenie: *make clean* w celu wyczyszczenia projektu z plików wykonywalnych.

W przypadku wykorzystania skryptów textitrun.sh i *clean.sh* należy:

1. Wpisać w plik *InputMatrix.txt* wyłącznie jedną liczbę całkowitą w pierwszej linijce, definiującą rozmiar macierzy sąsiedztwa.

2. Wykonać polecenie *./run.sh*, które zainicjalizuje UPC++, zapisze do pliku *nodes* odpowiednie informacje o dostępnych węzłach, skompiluje program oraz wywoła go dla 4 i 8 węzłów.

3. Wynik programu widoczny będzie w pliku *Output/Output.txt*

4. Wykonać polecenie: *./clean.sh* w celu wyczyszczenia projektu z plików wykonywalnych i pliku *Output.txt*.

Poza tym w pliku *InputMatrix.txt* poza wpisaniem rozmiaru macierzy sąsiedztwa, można ją ręcznie zdefiniować w następnych linijkach, przykładowo, w następujący sposób:

0	5	0	9	0	0	3	0
5	0	9	0	8	6	0	7
0	9	0	9	4	0	5	3
9	0	9	0	0	0	8	0
0	8	4	0	0	2	1	0
0	6	0	0	2	0	6	0
3	0	5	8	1	6	0	9
0	7	3	0	0	0	9	0

Macierz powinna być zdefiniowana przez użytkownika w sposób prawidłowy.

4 Podsumowanie

W ramach projektu przeprowadzono implementację zrównoleglonego algorytmu Prima z wykorzystaniem biblioteki UPC++. Celem było przyspieszenie wyznaczania minimalnego drzewa rozpinającego (MST) w grafach o dużej liczbie wierzchołków i krawędzi.

Przeprowadzone testy wykazały, że zrównoleglenie algorytmu przyniosło zauważalny wzrost wydajności, szczególnie dla większych grafów. Dla małych instancji różnice czasowe były nieznaczne, co jest zrozumiałe ze względu na narzut komunikacyjny między procesami. Jednak w przypadku grafów o rozmiarze rzędu kilku tysięcy wierzchołków, czas wykonania zmniejszał się wraz ze wzrostem liczby procesów, aż do momentu, gdy

dalsze zwiększanie liczby procesów przestawało być opłacalne (ze względu na rosnący koszt synchronizacji i komunikacji).

Efektywność równoległej wersji algorytmu była zależna m.in. od struktury grafu oraz strategii podziału danych. Lepsze rezultaty uzyskano w przypadku grafów gęstych, gdzie koszt obliczeń dominował nad kosztem komunikacji.

Zastosowanie **UPC++** do implementacji **algorytmu Prima** pozwala na:

- wydajną dekompozycję danych i obliczeń,
- efektywną komunikację między procesami bez kosztownego kopiowania,
- osiągnięcie dobrej skalowalności przy pracy na dużych, rozproszonych grafach.

Literatura

[1] https://www.cs.purdue.edu/homes/ayg/book/Slides/chap10_slides.pdf