AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ

Aplikacja rozproszona UPC

Mnożenie macierzy - algorytm Cannon'a

Krystian ŚLEDŹ

Mirosław KOŁODZIEJ

Kraków, 05.05.2023

Spis treści

Wstęp

Mnożenie macierzy jest podstawową operacją w wielu dziedzinach matematyki oraz informatyki. Proces mnożenia macierzy jest czasochłonny i może stanowić wąskie gardło w przypadku dużych rozmiarów macierzy.

Za pomocą rozproszonego modelu programowania możliwe jest znaczne przyspieszenie tych obliczeń przez dystrybucję zadań do wielu węzłów w klastrze obliczeniowym.

W projekcie zastosowano **programowanie rozproszone** do przyspieszenia procesu mnożenia macierzy. Wykorzystując globalnie adresowany model pamięci **UPC**, algorytm dystrybuuje elementy macierzy wejściowych pomiędzy dostępne węzły, które następnie wykonują operacje mnożenia w sposób równoległy. Wyniki są następnie zbierane i zapisywane, co daje ostateczną macierz wynikową.

Algorytm Cannon'a

Algorytm Cannon'a umożliwia równoległe mnożenie macierzy o rozmiarze $n \times n$, gdzie n jest potęgą dwójki. Algorytm ten składa się z kilku kroków:

Krok 1: Podział macierzy wejściowych na bloki Macierze wejściowe A oraz B zostają podzielone na bloki o rozmiarze $\frac{n}{p} \times \frac{n}{p}$, gdzie p oznacza liczbę węzłów przetwarzających.

Krok 2: Inicjalizacja węzłów przetwarzających Każdy węzeł przetwarzający posiada własny blok pamięci, do którego można wpisać elementy macierzy wejściowych. Następnie bloki $A_{i,j}$ oraz $B_{i,j}$ zostają przesłane do odpowiednich węzłów przetwarzających.

Krok 3: Przesuwanie bloków Bloki $A_{i,j}$ i $B_{i,j}$ zostają przesunięte o jedną pozycję w lewo wzdłuż osi X oraz o jedną pozycję w górę wzdłuż osi Y.

Krok 4: Mnożenie bloków Każdy węzeł przetwarzający wykonuje operację mnożenia na swoim bloku. Operacja ta polega na przemnożeniu bloku z jego sąsiadem po lewej stronie wzdłuż osi X oraz z sąsiadem powyżej wzdłuż osi Y, a następnie zsumowaniu wyników.

Krok 5: Przesuwanie wyników Wyniki mnożenia zostają przesunięte o jedną pozycję w prawo wzdłuż osi X oraz o jedną pozycję w dół wzdłuż osi Y.

Krok 6: Powtarzanie kroków 3-5 Kroki 3-5 są powtarzane $p - 1$ razy, aby wyniki mnożenia pojawiły się we właściwych blokach.

Krok 7: Zbieranie wyników Wyniki mnożenia znajdują się w blokach $C_{i,j}$, które należy zebrać w jedną macierz wynikową C.

Kod

3.1 Struktura kodu

3.1.1 Główne struktury danych

Program korzysta z trzech zmiennych typu `shared int[]`:

- `aMatrix`: Pierwsza macierz.
- `bMatrix`: Druga macierz.
- `resultMatrix`: Macierz wynikowa.

Rozmiar każdej z tych macierzy to `THREADS`, który jest liczony jako ilość wątków uruchomionych przez UPC.

3.1.2 Główne funkcje

Program składa się z czterech głównych funkcji:

- `printMatrix()`: Wypisuje daną macierz na standardowe wyjście. Każdy wiersz jest oddzielony nowym wierszem, a każdy element w wierszu jest oddzielony tabulacją.
- `readMatrix()`: Wczytuje macierz z pliku. Każdy element macierzy jest odczytywany jako pojedyncza liczba całkowita.
- `saveMatrix()`: Zapisuje daną macierz do pliku. Każdy wiersz jest oddzielony nowym wierszem, a każdy element w wierszu jest oddzielony tabulacją.
- `main()`: Inicjalizuje macierze, uruchamia mnożenie macierzy i zapisuje wynik do pliku wyjściowego.

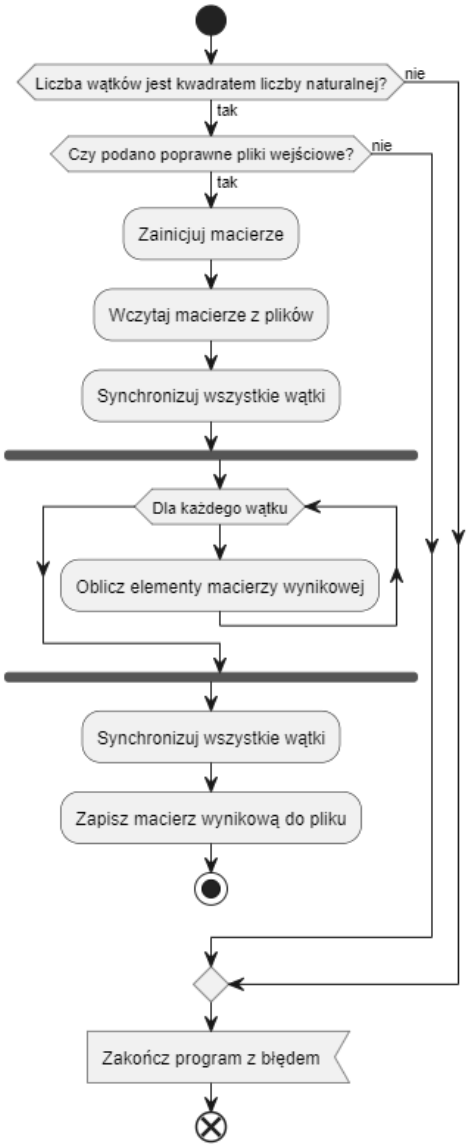
3.2 Implementacja

Najważniejszą częścią programu jest algorytm mnożenia macierzy, który jest zaimplementowany w funkcji `main()` w bloku `upc_forall()`, który tworzy wątki UPC, aby wykonać równoległe obliczenia. Każdy wątek oblicza jeden element macierzy

wynikowej. W celu obliczenia danego elementu, wątek mnoży odpowiedni wiersz z `aMatrix` przez odpowiednią kolumnę z `bMatrix` i dodaje wyniki tych mnożeń.

Wszystkie wątki są synchronizowane przed i po mnożeniu macierzy za pomocą `upc_barrier`, aby upewnić się, że wszystkie wątki zakończyły inicjalizację i obliczenia.

3.3 Schemat blokowy



Rysunek 3.1: Schemat blokowy

3.4 Uruchomienie

Do przygotowania i uruchomienia rozwiązania służy plik makefile, w którym zdefiniowane są następujące zmienne:

- `SHELL`
- `DIM`: wymiar macierzy,
- `NODES`: liczba wątków (obliczana jako kwadrat DIM),
- `F1`: plik z macierzą A,
- `F2`: plik z macierzą B,
- `FR`: plik wynikowy,
- `CLEAN`: lista plików do usunięcia.

oraz komendy:

- `default`: kompiluje i uruchamia program (build run)
- `v`: jw. z parametrem verbose (build run_verbose),
- `build`: kompiluje program,
- `nodes`: tworzy plik z węzłami,
- `run`: uruchamia skompilowany program,
- `run_verbose`: jw. z parametrem verbose,
- `clean`: usuwa utworzone pliki.

3.5 Ograniczenia i założenia

1. Program zakłada, że liczba wątków jest kwadratem liczby naturalnej.
2. Wymiary każdej z macierzy do mnożenia muszą być równe liczbie wątków.
3. Elementy macierzy są liczbami całkowitymi.

Podsumowanie

Zastosowanie algorytmu Cannon'a w kontekście języka programowania **UPC** stanowi skuteczną strategię rozproszonego mnożenia macierzy, umożliwiając wykorzystanie wielu procesorów do równoczesnego wykonywania operacji. Taka metoda nie tylko poprawia efektywność obliczeń, ale również znacznie przyspiesza proces mnożenia macierzy, co jest kluczowe w przypadku dużych zestawów danych.