AKADEMIA GÓRNICZO-HUTNICZA

WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ

Projekt 2

# Mnożenie macierzy - algorytm Cannon'a

Systemy równoległe i rozproszone

*Skład zespołu:*
*Gabriela Międlar*
*Bartłomiej Mucha*

# 1. Wstęp teoretyczny

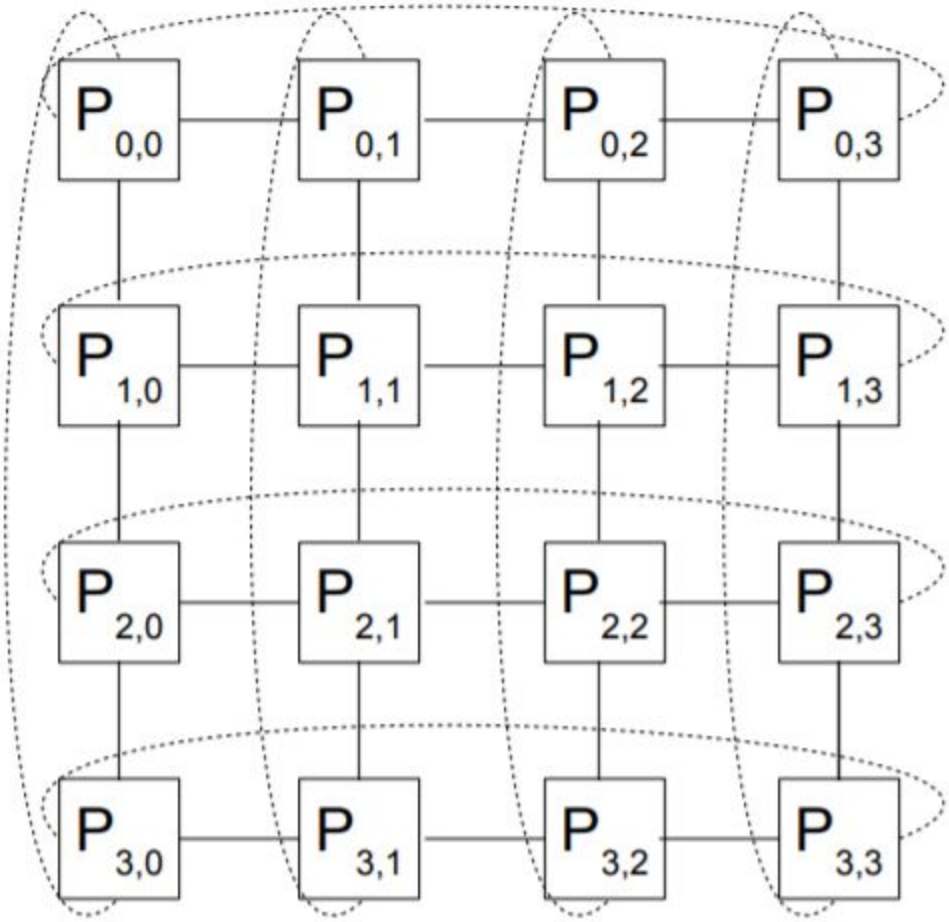
## 1.1. Założenia

Algorytm Cannona wykonuje mnożenie macierzy, wykorzystując możliwości systemów informatycznych zdolnych do realizacji wielowątkowych aplikacji. Takimi systemami mogą być systemy klastrowe, komputery z wielordzeniowymi, bądź wielowątkowymi procesorami centralnymi, jak również komputery z zainstalowanym układem karty graficznej. W kontekście tego algorytmu każdy wątek, rdzeń czy węzeł będzie nazywany procesorem. Założeniem algorytmu jest podział obliczeń, tak by każdy procesor miał do przetworzenia jak najmniejszego rozmiaru podmacierz, a wszystkie obliczenia działy się równolegle.

Wymagania algorytmu wobec danych wejściowych są następujące:

- Macierze wejściowe muszą być kwadratowe.
- Ilość procesorów musi być kwadratem liczby naturalnej
- Ilość elementów macierzy wejściowych musi być podzielna przez ilość procesorów.
- W przypadku macierzy nie spełniających powyższych warunków rozmiar macierzy można uregulować wypełniając potrzebne pola zerami lub zredukować ilość procesorów.
- początkowo macierz wynikowa musi być macierzą zer.

## 1.2. Działanie algorytmu



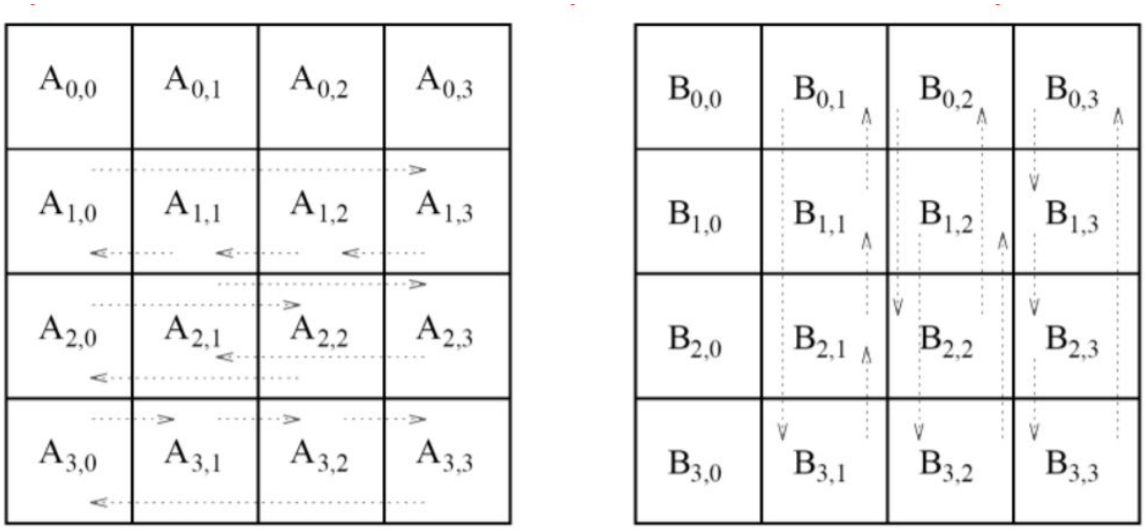

Rysunek 1. Schemat podziału macierzy wejściowych i wynikowej na procesory

Celem algorytmu jest rozwiązanie działania:

$$C = A * B, \quad (1)$$

gdzie C to macierz wynikowa, a A i B to macierze wejściowe. Wszystkie te macierze muszą zostać podzielone na podmacierze na kształt tej z Rysunku 1.

Zatem poszczególny procesor $P_{i,j}$ otrzyma na stałe podmacierz $C_{i,j}$ oraz w trakcie wykonywania będzie otrzymywać podmacierze $A_{i,j}$ i $B_{i,j}$.




Rysunek 2. Wstępna modyfikacja macierzy wejściowych.

Przed pierwszą iteracją należy zmienić kolejność podmacierzy zgodnie z Rysunkiem 2. Działanie to ma na celu zapobiec przetwarzania tych samych podmacierzy przez różne procesory

$$A_{i,j} := A_{i,(j+i)\%k} \ , \qquad (2)$$

$$B_{i,j} := B_{(i+j)\%k,j} \ , \qquad (3)$$

gdzie k jest pierwiastkiem liczby procesorów.

W danej iteracji procesor $P_{i,j}$ przemnaża otrzymane podmacierze i rezultat dodaje do podmacierzy wynikowej.

$$C_{i,j} \mathrel{+}= A_{i,j} * B_{i,j} \qquad (4)$$

Następnie przekazuje podmacierz $A_{i,j}$ procesowi na lewo, a podmacierz $B_{i,j}$ procesorowi w górę. W praktyce oznacza to przebudowanie macierzy wejściowych lub manipulacji w poszczególnych procesorach indeksami macierzy wejściowych, tak aby:

$$A_{i,j} := A_{i,(j+1)\%k} \ , \qquad (5)$$

$$B_{i,j} := B_{(i+1)\%k,j} \ , \qquad (6)$$

Po inicjalizacji danych wejściowych w postaci wykonania działań (2) i (3) można je przekazać do przetwarzania przez procesory. Zatem algorytm dla pojedynczego procesora w liście kroków wygląda następująco:

1. Podmień podmacierze wejściowe zgodnie ze wzorami (5) i (6).
2. Wykonaj działanie (4).
3. Powtórz kroki 1 i 2 k-razy.

Rezultatem będzie wynik działania (1).

## 2. Opis programu

Program *cannon.c* implementuje opisany powyżej algorytm z wykorzystaniem środowiska PGAS UPC. Umożliwia ono prowadzenie obliczeń w sposób równoległy. Aby móc korzystać z UPC na początku programu umieszczono dyrektywę:

***#include "upc.h"***

Następnie muszą zostać zaalokowane obiekty globalne, które będą współdzielone pomiędzy wszystkimi procesorami. Służy do tego słowo kluczowe ***shared***. Obiekty te będą reprezentować macierze wykorzystane do obliczeń.

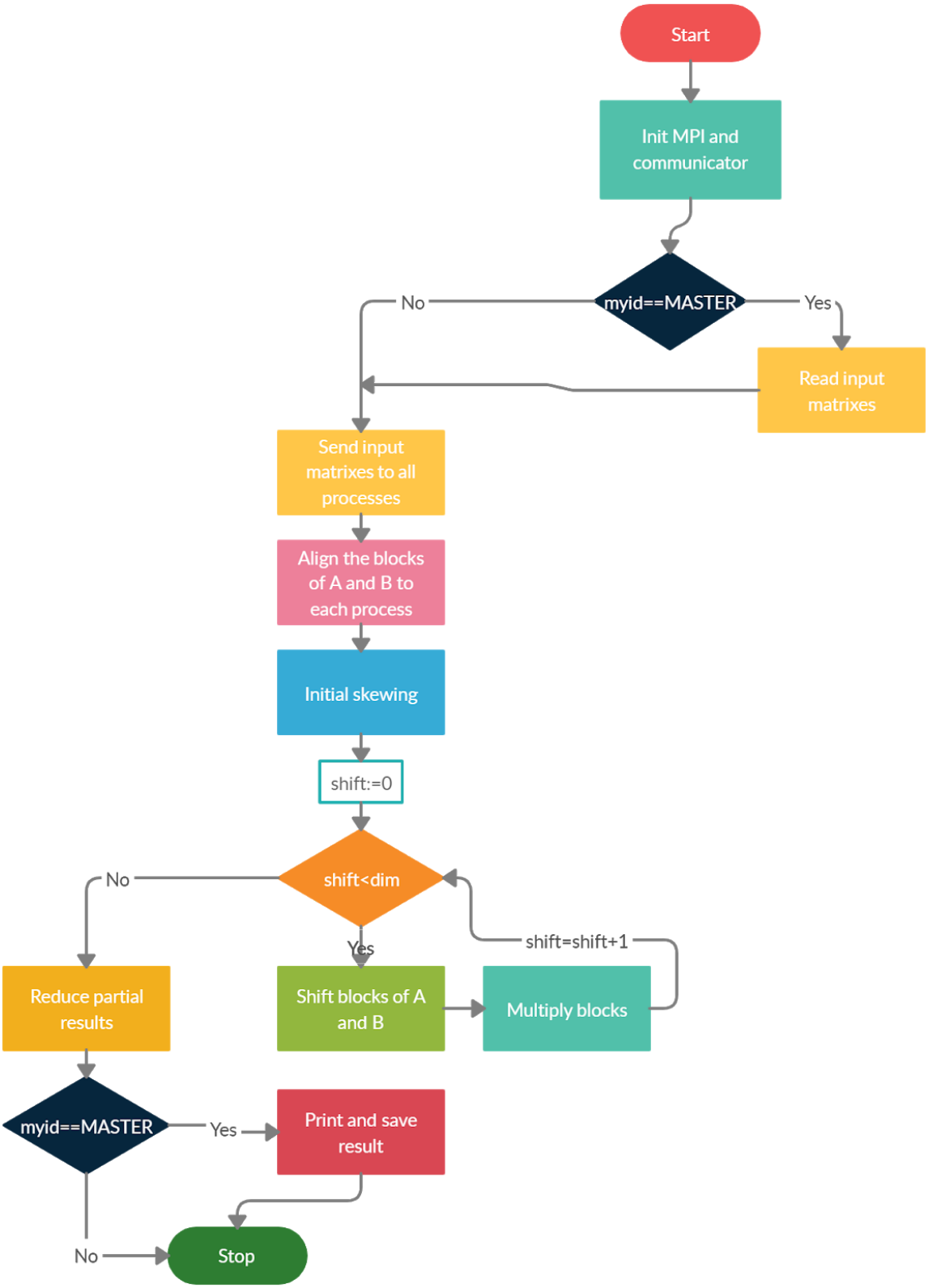
```
shared  int A[THREADS];
shared  int B[THREADS];
shared  int C[THREADS];
```

Proces MASTER (o identyfikatorze równym 0) wykorzystując funkcję ***loadInputFromFile*** czyta macierze wejściowe. W pętli ***upc_forall*** każda iteracja reprezentuje osobny wątek i to w jej ciele są obliczenia na wcześniej zaalokowanych obiektach współdzielonych.

Procesy organizowane są w siatkę. Zgodnie z algorytmem do każdego z nich przypisane są odpowiednie bloki macierzy wejściowych. Następnie

wykonywane jest wstępne przesunięcie bloków. Każdy z procesów przemnaża przypisane mu bloki macierzy A i B i wykonuje kolejne przesunięcie. Wymiar macierzy mówi o tym ile razy ta operacja jest wykonywana. Uzyskane wyniki są sumowane i zapisywane w tablicy pomocniczej pod indeksem odpowiadającym identyfikatorowi procesu.

Na koniec główny proces wypisuje macierz wynikową na standardowe wyjście oraz przy pomocy funkcji ***saveOutputToFile*** zapisuje ją do pliku *Output.txt*.


Start
Init MPI and communicator
myid==MASTER
No
Yes
Read input matrixes
Send input matrixes to all processes
Align the blocks of A and B to each process
Initial skewing
shift:=0
shift<dim
No
Yes
shift=shift+1
Reduce partial results
Shift blocks of A and B
Multiply blocks
myid==MASTER
Yes
Print and save result
No
Stop

## 3. Działanie programu

Do programu został dołączony plik *makefile* oraz pliki wejściowe *mat1.txt* i *mat2.txt*. Umieszczając je w tym samym katalogu co program możemy wywołać komendy:

- **make** - załadowuje zmienne środowiskowe do bieżącej powłoki Bash oraz kompiluje program z odpowiednimi flagami do postaci wykonywalnej,
- **make run** - uruchamia program domyślnie na 9 węzłach. Ilość węzłów można wyspecyfikować dodając argument **n=<ilość_węzłów>**,
- **make clean** - przywraca zawartość katalogu do stanu wejściowego, usuwając pliki wynikowe.

Aby program działał prawidłowo pliki wejściowe muszą zawierać macierze o takiej ilości elementów jak ilość węzłów, na których będzie uruchamiany program. Jeśli ilość elementów jest większa, z pliku wejściowego zostanie pobranych tylko n pierwszych wartości (gdzie n=<ilość_węzłów>).