НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Лабораторна робота №6

із дисципліни «Технології паралельних та розподілених обчислень»

на тему «Розробка паралельного алгоритму множення матриць з використанням MPI-методів обміну повідомленнями «один-до-одного» та дослідження його ефективності»

Підготував:

Студент ФІОТ, гр. ІТ-83,

Троян О.С,

Перевірив:

пос. Дифучина О. Ю.

Київ 2020

Мета роботи: розробити паралельний алгоритм множення матриць з використанням MPI-методів обміну повідомленнями «один-до-одного» та дослідження його ефективності.

Завдання:

1. Ознайомитись з методами блокуючого та неблокуючого обміну повідомленнями типу point-to-point.
2. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів блокуючого обміну повідомленнями

   30 балів.
3. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів неблокуючого обміну повідомленнями.

   30 балів.
4. Дослідити ефективність розподіленого обчислення алгоритму множення матриць при збільшенні розміру матриць та при збільшенні кількості вузлів, на яких здійснюється запуск програми. Порівняйте ефективність алгоритму при використанні блокуючих та неблокуючих методів обміну повідомленнями.

   40 балів.

Хід роботи

1. Множення матриць з блокуючим обміном повідомленнями.

1.1. Запускаємо середовище, у випадку коли логічних процесорів менше двох ми чекаємо коли з'являться інші.

```
var args1:string[] = args;
Environment.Run(ref args, action:comm:Intracommunicator =>
{
    while (true)
    {
        if (comm.Size >= 2)
            break;
        Console.WriteLine("Need at least two MPI tasks. Tasks - " + comm.Size);
        Thread.Sleep(millisecondsTimeout:1000);
    }

    if (comm.Rank == Master)
    {
        Console.WriteLine("Blocking");
        Console.WriteLine("Matrices rank - " + args1[0]);
        Console.WriteLine("Processes - " + comm.Size);
```

1.2. Якщо ранг процесса Master ми створюємо матриці

```
var rank = Convert.ToInt32(args1[0]);
var first:int[][] = CreateMatrix(rank);
var second:int[][] = CreateMatrix(rank);
var result:int[][] = CreateEmptyMatrix(rank);

var watch = new Stopwatch();
watch.Start();

var workersCount:int = comm.Size - 1;
var rowsPerWorker:int = first.Length / workersCount;

if (first.Length % workersCount != 0)
    throw new ArgumentException(message:"Workers count is not valid for matrix size");

for (var i = 0; i < workersCount; i++)
{
    var offset:int = rowsPerWorker * i;
    var request = new MatrixRowsDto
    {
        FirstRows = first[offset..(offset + rowsPerWorker)],
        Second = second,
        Offset = offset
```

Обов'язковою умовою є кратність кількості процесі і рангу матриці для рівномірної передачі даних.

1.3. Надсилаємо процесам блокуючі повідомлення.

```
for (var i = 0; i < workersCount; i++)
{
    var offset :int = rowsPerWorker * i;
    var request = new MatrixRowsDto
    {
        FirstRows = first[offset..(offset + rowsPerWorker)],
        Second = second,
        Offset = offset
    };
    comm.Send(request, dest: i + 1, tag: FromMaster);
}
```

1.4. Отримуємо результат множення, та додаємо до результуючої матриці.

```
for (var i = 0; i < workersCount; i++)
{
    var response = comm.Receive<MultiplyResultDto>( source: i + 1, tag: FromWorker);
    var offset :int = response.Offset;

    for (var j = 0; j < response.Result.Length; j++)
        result[offset + j] = response.Result[j];
}
```

1.5. Виводимо результат та час який був витрачений на виконання множення.

```
watch.Stop();
Console.WriteLine("Time for single: " + watch.ElapsedMilliseconds);
Console.WriteLine("Speedup: " + Inline2000MultiplicationTime / watch.ElapsedMilliseconds);
```

1.6. Якщо ж ранг процесу не Master, він виконує множення двох матриць

```
else
{
    var data = comm.Receive<MatrixRowsDto>( source: Master, tag: FromMaster);
    var result :int[][] = Multiply(data.FirstRows, data.Second);
    comm.Send( value: new MultiplyResultDto {Result = result, Offset = data.Offset}, dest: Master, tag: FromWorker);
}
```

1.7. Виконаємо множення для 5 процесів матриць 100 на 100

Результат:

```
C:\Users\Hapan9\Documents\GitHub\Parallel\Lab6>mpiexec -n 5 Lab6\bin\Debug\net5.0\Lab6.exe 100
Blocking
Matrices rank - 100
Processes - 5
Time for single: 130
Speedup: 752,9076923076923
```

2. Множення матриць з блокуючим обміном повідомленнями.

Всі кроки до відправки повідомлень аналогічні до пункту 1.

2.1. Відправка повідомлень

```
var requests = new RequestList();

for (var i = 0; i < workersCount; i++)
{
    var offset:int = rowsPerWorker * i;
    var value = new MatrixRowsDto
    {
        FirstRows = first[offset..(offset + rowsPerWorker)],
        Second = second,
        Offset = offset
    };
    var request = comm.ImmediateSend(value, dest:i + 1, tag:FromMaster);
    requests.Add(request);
}
```

Відправляємо неблокуючі повідомлення.

2.2. Після відправки всіх повідомлень отримуємо відповіді

```
requests.WaitAll();

var receiveRequests = new RequestList();

for (var i = 0; i < workersCount; i++)
{
    var receiveRequest = comm.ImmediateReceive<MultiplyResultDto>(source:i + 1, tag:FromWorker, action:response:Multip
    {
        var offset:int = response.Offset;

        for (var j = 0; j < response.Result.Length; j++) result[offset + j] = response.Result[j];
    });

    receiveRequests.Add(receiveRequest);
}

receiveRequests.WaitAll();
```

Важливо дочекатись поки операції відправки і отримання будуть закінчені.

2.3. Якщо ранг не Master виконуємо множення матриць

```
else
{
    var receiveRequest = comm.ImmediateReceive<MatrixRowsDto>( source: Master,  tag: FromMaster);
    receiveRequest.Wait();
    var data = (MatrixRowsDto) receiveRequest.GetValue();
    var result = new MultiplyResultDto
        {Result = Multiply(data.FirstRows, data.Second), Offset = data.Offset};
    comm.ImmediateSend(result,  dest: Master,  tag: FromWorker);
}
```

2.4. Як і в минулому прикладі розрахуємо добуток матриць 100 на 100 при 5 процессах.

Результат:

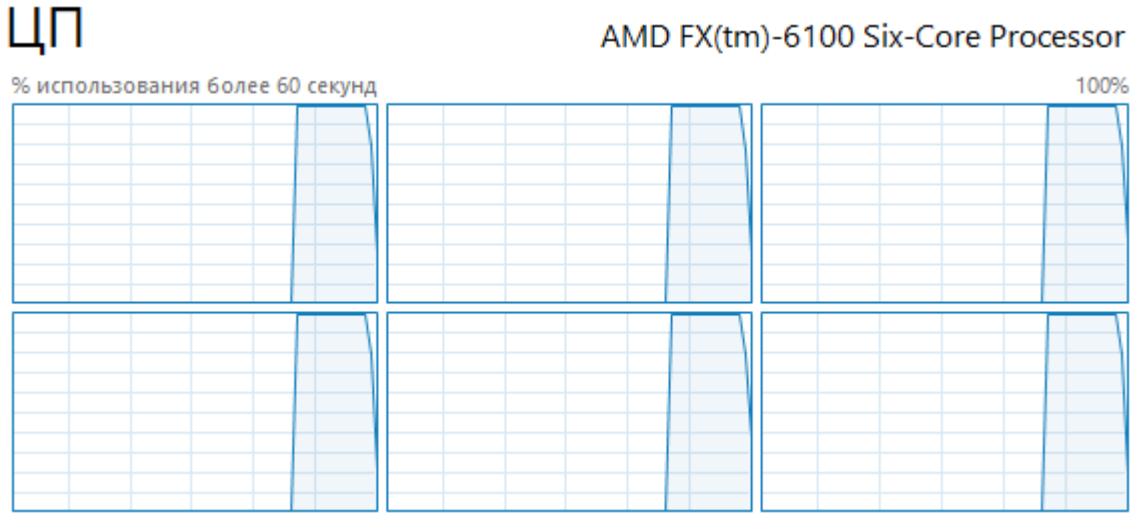
```
Immediate. Let's go.
Matrices rank - 100
Processes - 5
Time for parallel: 126
Speedup: 776,8095238095239
```

3. Виконаємо обчислення з різним рангом матриць та кількістю процесів:

Ранг матриці	100	100	500	500	1000	1000
Процеси	10	2	10	5	20	2
Час блокуючого	231	104	965	616	30302*	13814
Час не блокуючого	202	123	681	640	6322*	13753

*Тому як у мене процесор має 6 ядер (фізичних та логічних), запуск 21 процесу є доволі трудоємкою задачею, на виміри

можуть впливати будь які фонові процеси уже не кажучи про активні, тому виміри можуть бути помилковими.



Чим більше повідомлень надсилається тим швидше працює неблокована відправка повідомлень, але для цього потрібні машини з великою обчислювальною потужністю.

Висновок: Під час виконання лабораторної роботи ми Розробили паралельний алгоритм множення матриць з використанням MPIметодів обміну повідомленнями «один-до-одного» та дослідили його ефективність