МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт

З комп’ютерного практикуму № 6 з дисципліни
«Технології паралельних обчислень»

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

Виконав(ла) *ІП-15 Мєшков Андрій* _____

(шифр, прізвище, ім'я, по батькові)

Перевірив *Дифучина О. Ю.* _____

(шифр, прізвище, ім'я, по батькові)

Київ 2024

ЗАВДАННЯ

1.Ознайомитись з методами блокуючого та неблокуючого обміну повідомленнями типу point-to-point (див. лекцію та документацію стандарту MPI).

2. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів блокуючого обміну повідомленнями (лістинг 1). 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

Лістинг коду:

BlockedMult.py

```python
from mpi4py import MPI
import numpy as np
import sys

def main():
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    size = comm.Get_size()
    n = 2000
    master = 0

    if size < 2:
        if rank == master:
            print("Cannot mult")
        MPI.COMM_WORLD.Abort(0)
        sys.exit(0)

    workers = size - 1
    rows_for_worker = n // workers
    extra_rows = n % workers

    if rank == master:
        A = np.ones((n, n), dtype=int)
        B = np.ones((n, n), dtype=int)
        C = np.zeros((n, n), dtype=int)
        start_time = MPI.Wtime()

        start_row_a = 0
        for dest in range(1, workers + 1):
            rows = rows_for_worker + 1 if dest <= extra_rows else rows_for_worker
            comm.Send([A[start_row_a:start_row_a + rows, :], MPI.INT], dest=dest, tag=0)
            comm.Send([B, MPI.INT], dest=dest, tag=0)
            start_row_a += rows

        start_row = 0
        for source in range(1, workers + 1):
            rows = rows_for_worker + 1 if source <= extra_rows else rows_for_worker
            comm.Recv([C[start_row:start_row + rows, :], MPI.INT], source=source, tag=0)
            start_row += rows

        end_time = MPI.Wtime()
        elapsed_time_ms = end_time - start_time
        print(f"Time elapsed: {elapsed_time_ms:.2f} s - blocked matrix multiplication
{n}x{n}")

        # for i in range(n):
```

```
        #     for j in range(n):
        #         print(C[i, j], end=" ")
        #     print()

    else:
        rows = rows_for_worker + 1 if rank <= extra_rows else rows_for_worker
        a_rows = np.empty((rows, n), dtype=int)
        b = np.empty((n, n), dtype=int)
        comm.Recv([a_rows, MPI.INT], source=master, tag=0)
        comm.Recv([b, MPI.INT], source=master, tag=0)
        result = np.zeros((rows, n), dtype=int)

        for row in range(rows):
            for column in range(n):
                for k in range(n):
                    result[row, column] += a_rows[row, k] * b[k, column]

        comm.Send([result, MPI.INT], dest=master, tag=0)

if __name__ == "__main__":
    main()
```

NotBlockedMult.py

```
from mpi4py import MPI
import numpy as np
import sys

def main():
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    size = comm.Get_size()
    n = 2500
    master = 0

    if size < 2:
        if rank == master:
            print("Cannot mult")
        MPI.COMM_WORLD.Abort(0)
        sys.exit(0)

    workers = size - 1
    rows_for_worker = n // workers
    extra_rows = n % workers

    if rank == master:
        A = np.ones((n, n), dtype=int)
        B = np.ones((n, n), dtype=int)
        C = np.zeros((n, n), dtype=int)
        start_time = MPI.Wtime()

        requests = []
        start_row_a = 0
```

```
        for dest in range(1, workers + 1):
            rows = rows_for_worker + 1 if dest <= extra_rows else rows_for_worker
            req_a = comm.Isend([A[start_row_a:start_row_a + rows, :], MPI.INT], dest=dest,
tag=0)
            req_b = comm.Isend([B, MPI.INT], dest=dest, tag=0)
            requests.append(req_a)
            requests.append(req_b)
            start_row_a += rows

        MPI.Request.Waitall(requests)

        requests = []
        start_row = 0
        for source in range(1, workers + 1):
            rows = rows_for_worker + 1 if source <= extra_rows else rows_for_worker
            req_c = comm.Irecv([C[start_row:start_row + rows, :], MPI.INT], source=source,
tag=0)
            requests.append(req_c)
            start_row += rows

        MPI.Request.Waitall(requests)
        end_time = MPI.Wtime()
        elapsed_time_ms = end_time - start_time
        print(f"Time elapsed: {elapsed_time_ms:.2f} s - not-blocked matrix multiplication
{n}x{n}")

        # Uncomment below lines to print matrix C (Not recommended for large n)
        # for i in range(n):
        #     for j in range(n):
        #         print(C[i, j], end=" ")
        #     print()

    else:
        rows = rows_for_worker + 1 if rank <= extra_rows else rows_for_worker
        a_rows = np.empty((rows, n), dtype=int)
        b = np.empty((n, n), dtype=int)
        requests = []
        req_a_rows = comm.Irecv([a_rows, MPI.INT], source=master, tag=0)
        req_b = comm.Irecv([b, MPI.INT], source=master, tag=0)
        requests.append(req_a_rows)
        requests.append(req_b)

        MPI.Request.Waitall(requests)

        result = np.zeros((rows, n), dtype=int)
        for row in range(rows):
            for column in range(n):
                for k in range(n):
                    result[row, column] += a_rows[row, k] * b[k, column]

        comm.Isend([result, MPI.INT], dest=master, tag=0)

if __name__ == "__main__":
```

```
main()
```

Результат:

Рисунок 1 – Результат запуску програми

В таблиці зображено порівняння алгоритмів.

Розмір	Звичайний, с	8			
		Блокований, с	Прискорення	Не блокований, с	Прискорення
1000	567	193	2,9378	205	2,7659
1500	2167	739	2,9322	765	2,8291
2000	4698	1605	2,9271	1583	2,9679
2500	10864	3699	2,9369	3644	2,9815
3000	17792	5964	2,9831	5948	2,9911
Розмір	Звичайний, с	9			
		Блокований, с	Прискорення	Не блокований, с	Прискорення
1000	567	179	3,1676	185	3,0648
1500	2167	690	3,1405	683	3,1728
2000	4698	1466	3,2046	1392	3,375
2500	10864	3362	3,2333	3291	3,3011
3000	17792	5325	3,3412	5301	3,3563
Розмір	Звичайний, с	10			
		Блокований, с	Прискорення	Не блокований, с	Прискорення
1000	567	182	3,1154	181	3,1326
1500	2167	661	3,2783	648	3,3441
2000	4698	1329	3,535	1256	3,7404
2500	10864	3019	3,5985	2892	3,7566
3000	17792	4991	3,5648	4647	3,8287

ВИСНОВКИ

В результаті роботи над комп’ютерним практикумом було розроблено програму, що реалізує паралельне множення матриць з використанням MPI методів.