МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Паралельні та розподілені обчислення
ЛАБОРАТОРНА РОБОТА №9
«Інтерфейс паралельного програмування MPI»

Виконала:
студентка групи ПМі-31
Дудчак Валентина Юріївна

Львів 2024

Тема: Інтерфейс паралельного програмування MPI

Мета: Побудувати один з методів лабораторних 2-7 в інтерфейсі паралельного програмування: MPI.

Хід роботи:

Я побудувала метод множення матриць, використовуючи мову C++.

Послідовний алгоритм

Послідовний алгоритм реалізовано у методі sequential_matrix_multiply.

У ньому я просто множу матриці, не використовуючи додаткових методів:

```
void sequential_matrix_multiply(int a[N][N], int b[N][N], int c[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            c[i][j] = 0;
            for (int k = 0; k < N; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```

Окремо я створила функцію print_matrix, щоб перевірити, чи методи обчислюють результати правильно:

```
Матриця A:
8 74 31 45 24
41 93 88 28 41
4 10 61 100 17
98 13 11 80 80
68 94 86 29 95
```

```
Матриця B:
50 59 73 79 10
66 43 4 30 13
70 58 34 79 36
27 68 34 50 22
73 37 46 92 58
```

```
Результат множення C:
10421 9400 4568 9759 4540
18097 14943 9195 18153 7781
9071 11633 6588 11999 5552
14528 15379 13980 20361 7945
23342 18529 13620 25176 11146
```

```
Результат множення C (паралельний метод)
10421 9400 4568 9759 4540
18097 14943 9195 18153 7781
9071 11633 6588 11999 5552
14528 15379 13980 20361 7945
23342 18529 13620 25176 11146
```

Паралельний алгоритм

Паралельний алгоритм в інтерфейсі MPI реалізовано у методі mpi_matrix_multiply.

Для розпаралелення я використовую MPI_Scatter (розподіляю рядки матриці A для обчислення між потоками), MPI_Bcast (транслює матрицю B до всіх потоків) та MPI_Gather (збирає результати зі всіх потоків в одну результуючу матрицю):

```
void mpi_matrix_multiply(int* a, int* b, int* c, int n, int rank, int size) {
    int rows_per_process = n / size;
    int* sub_a = new int[rows_per_process * n];
    int* sub_c = new int[rows_per_process * n];

    MPI_Scatter(a, rows_per_process * n, MPI_INT, sub_a, rows_per_process * n, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Bcast(b, n * n, MPI_INT, 0, MPI_COMM_WORLD);

    for (int i = 0; i < rows_per_process; i++) {
        for (int j = 0; j < n; j++) {
            sub_c[i * n + j] = 0;
            for (int k = 0; k < n; k++) {
                sub_c[i * n + j] += sub_a[i * n + k] * b[k * n + j];
            }
        }
    }

    MPI_Gather(sub_c, rows_per_process * n, MPI_INT, c, rows_per_process * n, MPI_INT, 0, MPI_COMM_WORLD);

    delete[] sub_a;
    delete[] sub_c;
}
```

На скріншоті вище видно, що цей метод також обчислює результати правильно.

Аналіз результатів

Як і у попередніх лабораторних роботах, розпаралелення помітно ефективніше зі зростанням розміру даних:

```
Розмірність матриць: 100x100
Послідовний метод: 0.00298825 секунд
Паралельний метод (MPI): 0.00251471 секунд
```

```
Розмірність матриць: 200x200
Послідовний метод: 0.0295111 секунд
Паралельний метод (MPI): 0.00874771 секунд
```

```
Розмірність матриць: 400x400
Послідовний метод: 0.161509 секунд
Паралельний метод (MPI): 0.0436125 секунд
```

```
Розмірність матриць: 800x800
Послідовний метод: 1.28281 секунд
Паралельний метод (MPI): 0.345979 секунд
```

Висновок:

У даній лабораторній роботі я порівняла швидкодію паралельного методу множення матриць з використанням MPI та послідовного. Паралельний метод значно прискорює обчислення, розподіляючи роботу між кількома процесами. Зі збільшенням розміру матриць та кількості процесів ефективність паралельного методу зростає, хоча для невеликих матриць накладні витрати можуть знижувати перевагу MPI.