Министерство образования и науки Российской Федерации Федеральное агентство по образованию Федеральное государственное бюджетное образовательное учреждение высшего образования «Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Лабораторная работа №4

по курсу «Параллельное программирование»

Выполнил студент группы ИВТ-31\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Птахова А.М/

Проверил доцент кафедры ЭВМ \_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Долженкова М.Л./

Киров 2023

1. Цель

Знакомство со стандартом MPI, получение навыков реализации многопоточных SPMD-приложений с применением библиотеки MPICH.

2. Задание

- Выполнить разбиение исследованного в ходе первой лабораторной работы алгоритма на независимо выполняемые фрагменты;

- Реализовать многопоточную версию алгоритм с помощью языка С++ и библиотеки MPI, используя при этом необходимые примитивы синхронизации;

- Показать корректность полученной реализации путем осуществления тестирования на построенном в ходе первой лабораторной работы наборе тестов;

- Провести доказательную оценку эффективности многопоточной реализации алгоритма.

3. Алгоритм

3.1. Описание алгоритма

Алгоритм LU – разложения матрицы с использованием MPI следующий:

1. Исходная матрица транслируется всем потокам

2. Выделяется нулевой поток – мастер, который распределяет строки матрицы между потоками

3. Каждый поток принимает свою строку и выполняет преобразования для нее по алгоритму Краута

4. Каждый поток пакует вычисленные значения в 2 буфера (для L и для U матриц)

5. Основной поток принимает данные из всех потоков и записывает их в исходную матрицу

6. п. 2-5 повторяется несколько раз (рассчитывается, как отношение размерности матрицы к числу выделенных потоков)

3.2. Выбор топологии

Для решения поставленной задачи была выбрана топология «звезда», так как:

1) подзадачи не имеют последовательного выполнения

2) подзадачи необходимо распределять между потоками и лучше всего это делать централизовано

3.3. Листинг программы

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size\_of\_claster);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &process\_Rank);

MPI\_Status status;

double totalTime=0;

int count\_loop = ceil( n / size\_of\_claster) + 1 ;

/\*cout << "loop: " << count\_loop << '\n';\*/

double start = MPI\_Wtime();

for (int z=0; z<250; z++ )

{

if (process\_Rank == 0)

{

for (int i = 1; i < size\_of\_claster; i++)

{

MPI\_Send(&(mas[i + offset][0]), n, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

}

for (int i = 1; i < size\_of\_claster; i++)

{

MPI\_Recv(buf, n, MPI\_INT, i, 0, MPI\_COMM\_WORLD, &status);

for (int j = 0; j < n; j++)

{

mas[i][j] = buf[j];

}

}

for (int i = 1; i < size\_of\_claster; i++)

{

MPI\_Recv(buf2, n, MPI\_INT, i, 0, MPI\_COMM\_WORLD, &status);

for (int j = 0; j < n; j++)

{

if (buf2[j] != 0)

{

mas[j][i] = buf2[j];

}

}

}

double elapsedTime = MPI\_Wtime() - start;

totalTime += elapsedTime;

std::cout << "total Time: " << totalTime << '\n';

}

else

{

MPI\_Recv(buf, n , MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

for (int k = process\_Rank; k < n; k++)

{

int sum = 0;

for (int j = 1; j < process\_Rank; j++)

{

sum += (mas[k][j] \* mas[j][k]);

}

buf[k] = buf[k] - sum;

}

for (int k = process\_Rank; k < n; k++)

{

if (process\_Rank < k)

{

int sum = 0;

for (int j = 1; j < process\_Rank; j++)

{

sum += (mas[k][j] \* mas[j][k]);

}

buf2[k] = (mas[k][process\_Rank] - sum) / mas[process\_Rank][process\_Rank];

}

}

MPI\_Send(buf, n, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Send(buf2, n, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

}

offset += 2;

}

MPI\_Finalize();

4. Таблица

Сравнение времени выполнения программы представлено в таблице 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Разм. | Послед. | Потоки | OpenMP | MPI |
| 1 | 10 | time: 0.019 ms | time: 9.851 ms | time: 7.197 ms | time: 5.87 ms |
| 2 | 50 | time: 2.04 ms | time: 18.373 ms | time: 8.262 ms | time: 11.654 ms |
| 3 | 100 | time: 15.798 ms | time: 37.944 ms | time: 10.88 ms | time: 76.523 ms |
| 4 | 500 | time: 1890.83 ms | time: 577.73 ms | time: 277.44 ms | Ошибка |
| 5 | 1000 | time: 15242 ms | time: 3399.6 ms | time: 2262.8 ms | Ошибка |
| 6 | 2000 | time: 148965 ms | time: 31956.3 ms | time: 19494.4 ms | Ошибка |
| 7 | 5000 | time: 1.93568e+06 ms | time: 412307 ms | time: 281279 ms | Ошибка |
| 8 | 7500 | time: 6.65623e+06 ms | time: 1.55738e+06 ms | time: 1.16805e+06 ms | Ошибка |
| 9 | 13000 | time: 2.31827e+07 ms | time: 4.9538e+06 ms | time: 4.27352e+06 ms | Ошибка |
| 10 | 15000 | time: 1.414706e+08 ms | time: 3.15265e+07 ms | time: 2.81277e+07 ms | Ошибка |

Таблица 1 – Сравнительные результаты

5. Доказательство эффективности

Для вычисления новых значений L и U матриц необходимо вычислить сумму произведений элементов, находящихся на пересечении строки и столбца. То есть для вычисления суммы для элемента, находящегося на позиции (4, 4), нужно вычислить произведение элементов (1, 4) и (4, 1), (2, 4) и (4, 2), (3, 4) и (4, 3), а потом вычислить их сумму. Вычисление суммы представлено на рисунке 1.

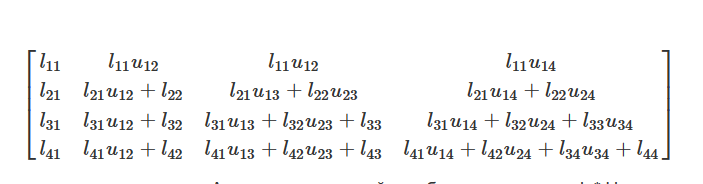


Рисунок 1 – Вычислении суммы

Для возможности вычисления суммы необходимо выделять матрицу каждому потоку. И в этот момент возникают проблемы, так как копирование матрицы для каждого потока приводит к большим ресурсным затратам, что приводит к выходу программы из строя на больших объемах данных. Эту гипотезу можно проверить при помощи изменения числа работающих потоков: при кол-ве работающих потоков равных 5 программа перестает работать на размерности 200; при кол-ве работающих потоков равных 3 – на размерности 500.

Кроме того, алгоритм LU – разложения требует постоянной синхронизации для записи вычисленных значений в матрицу. Для выполнения синхронизации необходимо использовать передачу сообщения. Это сказывается на времени выполнения программы, что можно заметить на полученных результатах времени выполнения программы.

6. Выводы

В ходе выполнения лабораторной работы были получены навыки реализации многопоточных приложений при помощи использования библиотеки MPI. В результате выполнения сформировалось представление о том, что использование данной библиотеки целесообразно лишь в том случае, когда задача представляет собой несвязанные по данным подзадачи. В противном случае прироста в скорости выполнения программы наблюдаться не будет, будет лишь ухудшение за счет затраты на время передачи и приема сообщения. Кроме того, был сделан вывод о невозможности использования MPI для нераспределенных задач на большом объеме данных.