Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
имени академика С.П. Королева»

ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ

Отчёт по лабораторной работе №3

РЕАЛИЗАЦИЯ ПЕРЕМНОЖЕНИЯ МАТРИЦ ПО ТЕХНОЛОГИИ MPI

Томашайтис Павел

Группа 6313-100503D

Самара 2024

1 Цель работы

1) Реализовать программу перемножения двух матриц с использованием технологии MPI.

2) Упростить программу для облегчения её запуска на суперкомпьютере «Сергей Королёв».

3) Запустить программу на суперкомпьютере «Сергей Королёв», провести эксперименты для различного числа используемых ядер при работе программы.

4) Измерить статистические характеристики для времени перемножения двух матриц.

2 Реализация программы перемножения двух матриц с использованием технологии MPI.

Программа перемножения двух матриц на языке C++, разработанная в предыдущих лабораторных работах, была существенно изменена при подготовке к данной работе. Матрицы теперь представляются в виде одной строки, а не в виде массива строк, что позволяет существенно ускорить время выполнения программы. Сама программа была разделена на 3 файла – matrix.cpp, matrix.hpp и lab_3.cpp. Файлы matrix.hpp и matrix.cpp соответствуют обновлённому классу матрица, lab_3.cpp – точке входа в программу. При запуске программа перемножает случайно сгенерированные матрицы различных размеров (от 100 до 1000) по технологии MPI, при этом основной процесс выводит в консоль время, затраченное на перемножение. Размер матрицы изменяется с шагом 100; для определённого размера выполняется перемножение 100 различных пар матриц – необходимо для подсчёта различных статистических показателей. Таким образом, всего в программе перемножается 1000 матриц при запуске.

Метод перемножения 2 матриц по технологии MPI представлен на рисунке 1.

```
double SquareMatrix::mpi_dot(SquareMatrix& rhs, int rank, int count) {
    if (rhs._size != _size)
        throw std::invalid_argument("Error! Matrix size was not the same.");
    double begin = MPI_Wtime();
    SquareMatrix result = SquareMatrix(_size);
    int* A = _data, *B = rhs._data, *C = result._data;
    MPI_Status status;
    int tmp = 0, ind = 0;
    int part_size = _size / count;
    int part = _size * part_size;
    int* bufA = new int[part];
    int* bufB = new int[part];
    int* bufC = new int[part];
    if (rank == 0) {
        for (int i = 0; i < part_size; i++) {
            for (int j = 0; j < part_size; j++) {
                tmp = B[i * _size + j];
                B[i * _size + j] = B[j * _size + i];
                B[j * _size + i] = tmp;
            }
        }
    }
    MPI_Scatter(A, part, MPI_INT, bufA, part, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Scatter(B, part, MPI_INT, bufB, part, MPI_INT, 0, MPI_COMM_WORLD);
    for (int i = 0; i < part_size; i++) {
        for (int j = 0; j < part_size; j++) {
            tmp = 0;
            for (int k = 0; k < part_size; k++)
                tmp += bufA[i * _size + k] * bufB[k * _size + j];
            bufC[i * _size + j + part_size * rank] = tmp;
        }
    }
    int next_proc, prev_proc;
    for (int p = 1; p < count; p++) {
        next_proc = rank + 1;
        if (rank == count - 1)
            next_proc = 0;
        prev_proc = rank - 1;
        if (rank == 0)
            prev_proc = count - 1;
        MPI_Sendrecv_replace(bufB, part, MPI_INT, next_proc, 0, prev_proc, 0,
MPI_COMM_WORLD, &status);
        for (int i = 0; i < part_size; i++) {
            for (int j = 0; j < part_size; j++) {
                tmp = 0;
                for (int k = 0; k < part_size; k++)
                    tmp += bufA[i * _size + k] * bufB[k * _size + j];
                if (rank - p >= 0)
                    ind = rank - p;
                else
                    ind = (count - p + rank);
                bufC[i * _size + j + ind * part_size] = tmp;
            }
        }
    }
    MPI_Gather(bufC, part, MPI_INT, C, part, MPI_INT, 0, MPI_COMM_WORLD);
    delete[] bufA;
    delete[] bufB;
    delete[] bufC;
    double end = MPI_Wtime();
    *this = result;
    return end - begin;
}
```

Рисунок 1 – Метод перемножения 2 матриц по технологии MPI.

Основной идеей при перемножении двух матриц по технологии MPI являлось разделение матрицы на отдельные участки и из перемножение

отдельными процессами. Время измерялось встроенной в библиотеку mpi.h функцией MPI_Wtime().

Для облегчения запуска разработанной программы на суперкомпьютере Самарского Университета «Сергей Королёв», она была несколько упрощена – 3 файла были объединены в один (main.cpp), от части используемых библиотек пришлось избавиться.

3 Запуск программы перемножения двух матриц по технологии MPI на суперкомпьютере «Сергей Королёв»

Для получения доступа к удалённому рабочему столу Самарского Университета использовалась программа VMware Horizon Client. Далее для получения доступа к суперкомпьютеру Самарского Университета «Сергей Королёв» были установлены программы PuTTY (необходима для входа на кластер по протоколу SSH версии 2) и WinSCP (необходима для передачи данных на кластер по протоколу SFTP).

Компиляция программы на кластере выполнялась командой, представленной на рисунке 2.

```
mpicxx main.cpp -o lab3
```

Рисунок 2 – команда для компиляции программы с MPI.

Для того, чтобы поставить выполнение данной программы в очередь задач на суперкомпьютере, был написан специальный PBS-скрипт start-MPI.pbs, представленный на рисунке 3.

```
#!/bin/bash
#SBATCH --job-name=lab3
#SBATCH --time=0:10:00
#SBATCH --ntasks-per-node=16
#SBATCH --partition batch

module load intel/mpi4
mpirun -r ssh ./lab3
```

Рисунок 3 – PBS-скрипт для постановки задачи в очередь задач на суперкомпьютере.

Постановка задачи выполнения программы в очередь задач на суперкомпьютере выполняется командой, представленной на рисунке 4.

```
sbatch StartMPI.pbs
```

Рисунок 4 – Команда для постановки задачи в очередь задач на суперкомпьютере.

Проверка выполнения задачи, поставленной в очередь задач на суперкомпьютере, выполняется командой, представленной на рисунке 5.

```
squeue -u <login>
```

Рисунок 5 – Команда для проверки выполнения задачи, поставленной в очередь задач на суперкомпьютере.

После выполнения программы на суперкомпьютере её вывод в стандартный поток вывода будет сохранён в файл `slurm-<номер>.out`.

Всего было проведено 9 экспериментов с различным количеством доступных для работы программы ядер – 1, 2, 4, 6, 8, 10, 12, 14, 16. При проведении эксперимента с одним ядром количество перемножаемых матриц было уменьшено с 1000 до 100 (перемножалось по 10 различных матриц для каждого размера).

Пример работы с суперкомпьютером через программу PuTTY представлен на рисунке 6.

```
[2021-00255@login2 ~]$ mpicxx main.cpp -o lab3
[2021-00255@login2 ~]$ sbatch startMPI.pbs
Submitted batch job 91063
[2021-00255@login2 ~]$ squeue -u 2021-00255
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             91063     batch     lab3 2021-002  R       0:03      1 n281
[2021-00255@login2 ~]$ squeue -u 2021-00255
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
[2021-00255@login2 ~]$
```

Рисунок 6 – Пример работы с суперкомпьютером через программу PuTTY.

Пример работы с суперкомпьютером через программу WinSCP представлен на рисунке 7.

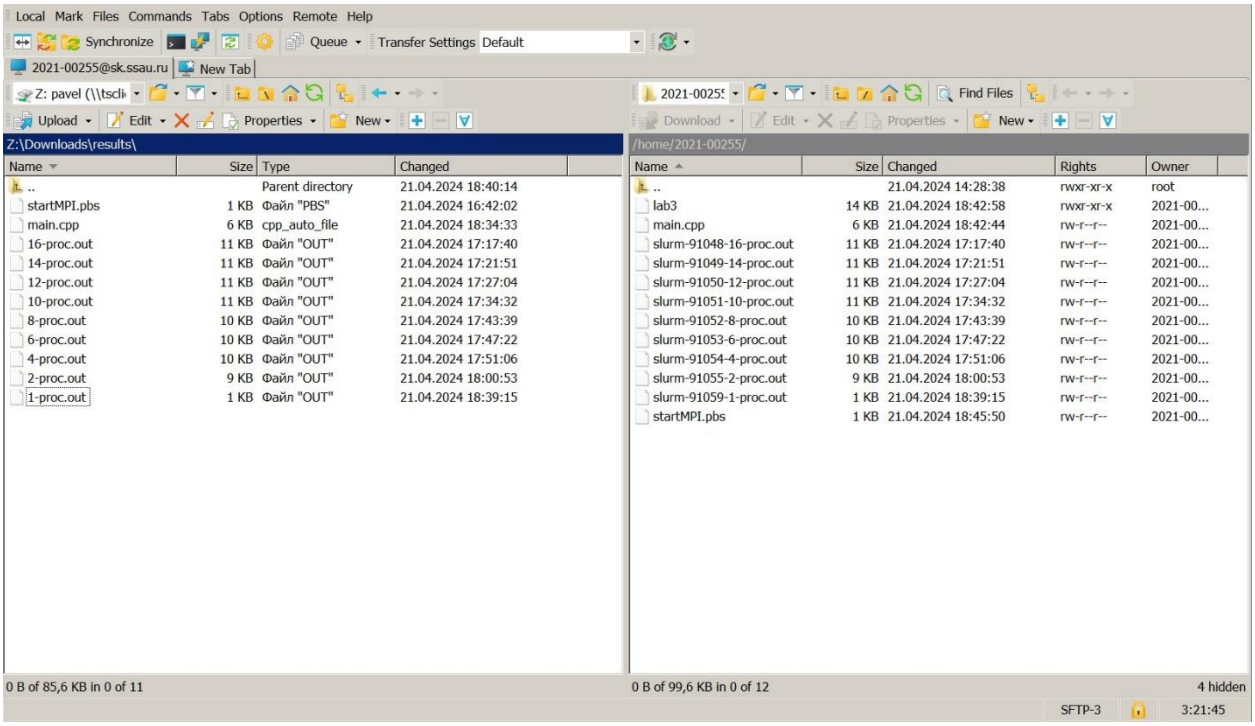

Рисунок 7 – Пример работы с суперкомпьютером через программу WinSCP.

По результатам работы с суперкомпьютером были получены файлы, содержащие время перемножения матриц различных размеров для различного числа используемых ядер. Далее эти файлы будут подвергнуты статистической обработке.

4 Программа для измерения статистических характеристик, связанных со временем перемножения двух матриц, на языке Python

Программа, написанная на языке Python и представленная в файле statistics.py, позволяет провести статистический анализ по выборке из временных интервалов, полученных при перемножении матриц. Для данной выборки в программе вычисляется среднее, медиана, дисперсия, среднеквадратическое отклонение, коэффициент эксцесса, коэффициент асимметрии, доверительный интервал (надёжность = 0,95). Результат работы программы для матриц размером 1000 на 1000 и различного числа используемых ядер представлен в файле statistics.txt.

```
Number of process:  16
Matrices 1000x1000:
Mean: 0.020853531
Median: 0.020793
Dispersion: 7.644745733900006e-08
STD: 0.00027649133320775183
Skewness: 4.924856488089981
Kurtosis: 24.263058300311943
Confidence interval for GAMMA = 0.95: (0.020798392736959073, 0.02090866926304093)
```

Рисунок 8 – Пример работы программы измерения статистических характеристик для матриц 1000 на 1000 и 16 используемых ядер.

Кроме того, программа в конце своей работы формирует графики зависимости средней величины времени, необходимой для перемножения матриц, от их размера и от количества используемых ядер при перемножении.

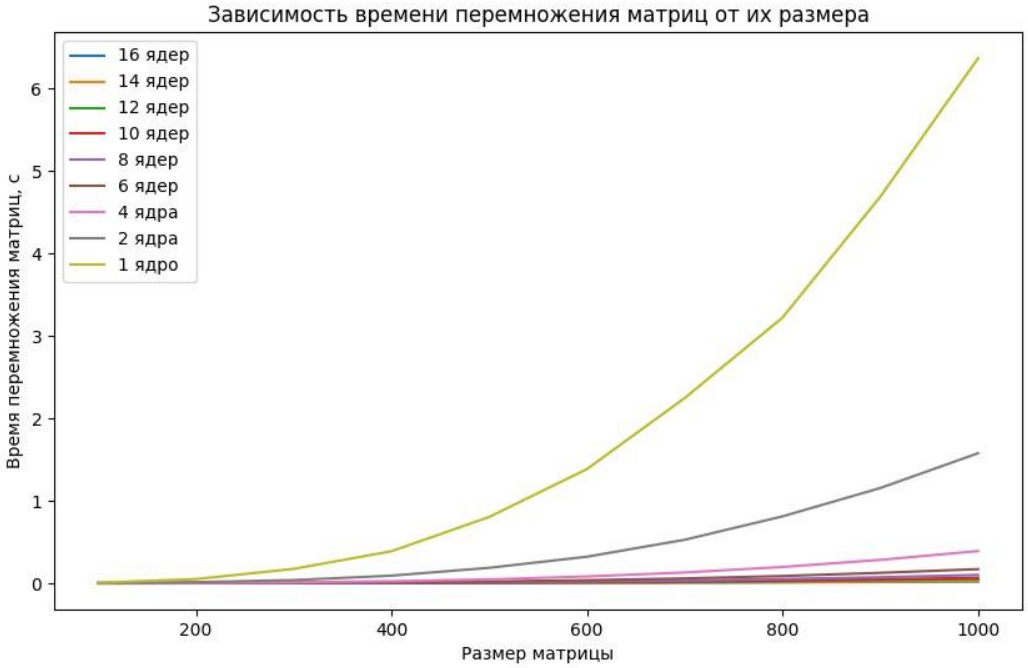

Рисунок 9 – График зависимости времени перемножения матриц от их размера, линейная шкала.

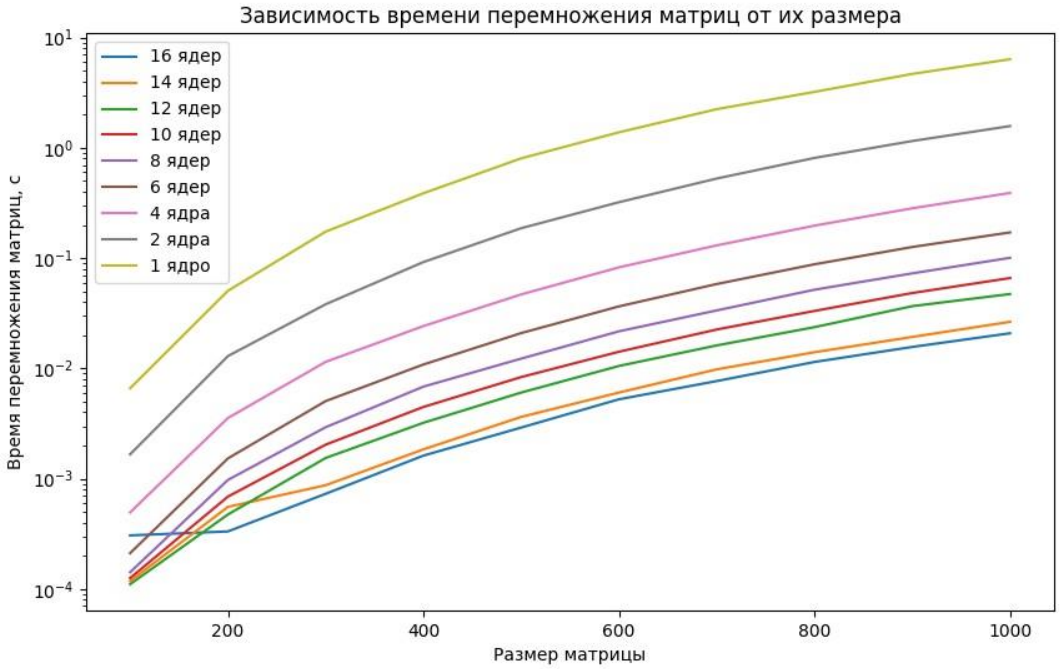

Рисунок 10 – График зависимости времени перемножения матриц от их размера, логарифмическая шкала.

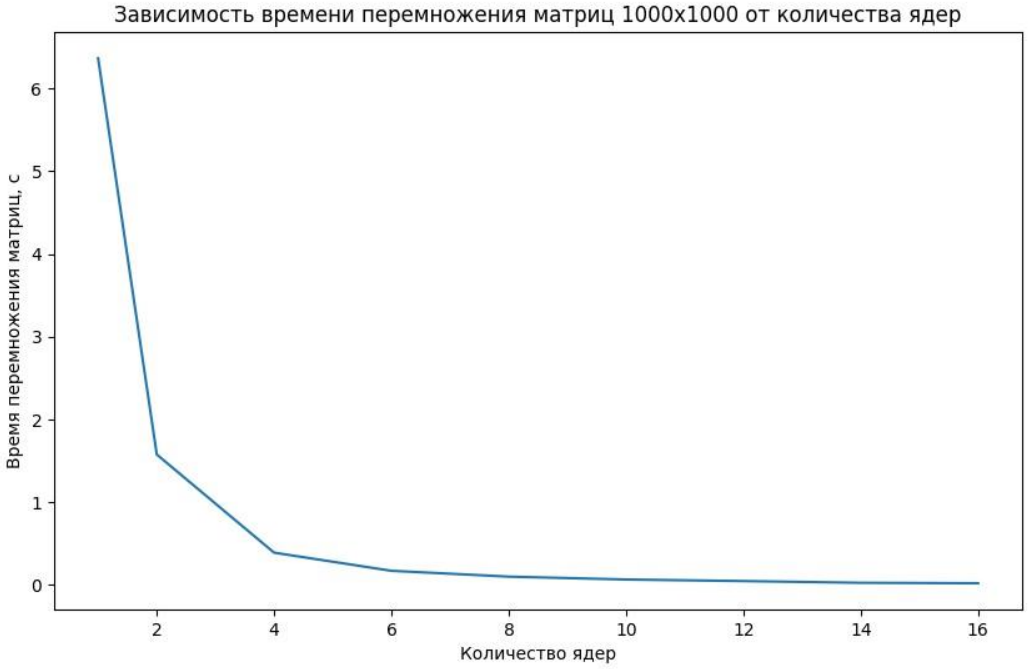

Рисунок 11 – График зависимости времени перемножения матриц размером 1000 на 1000 в зависимости от количества ядер, линейная шкала.

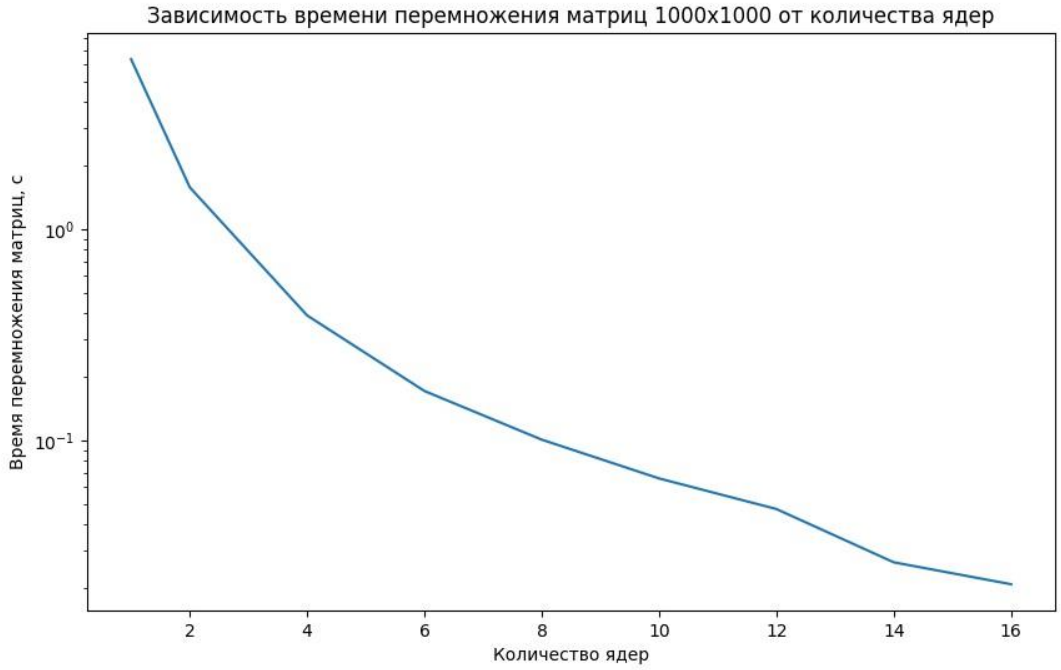

Рисунок 12 – График зависимости времени перемножения матриц размером 1000 на 1000 в зависимости от количества ядер, логарифмическая шкала.

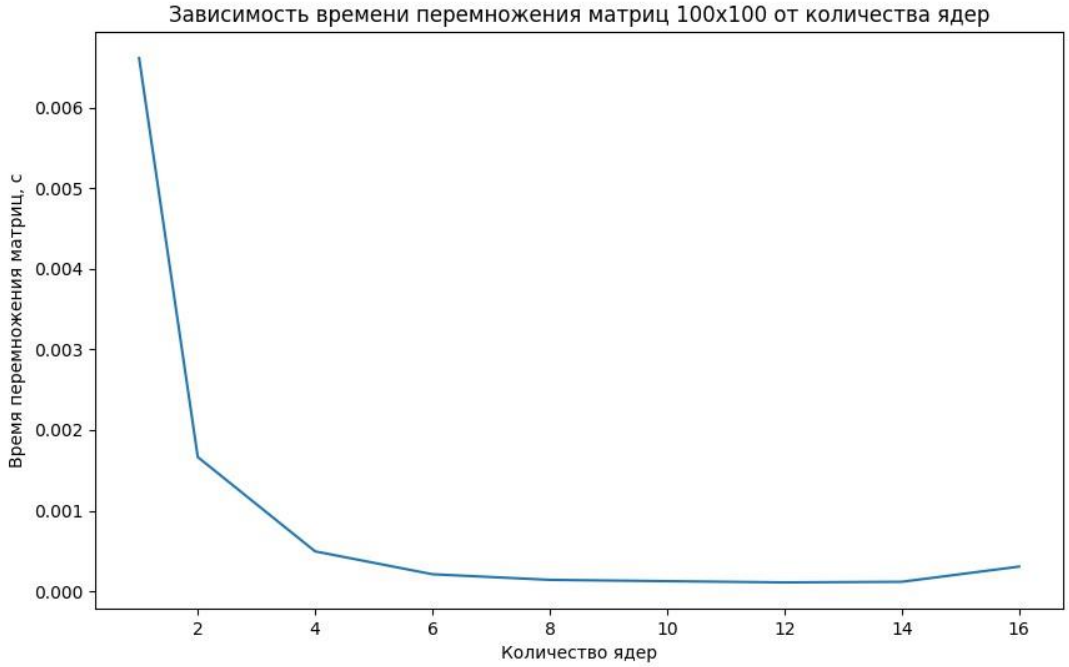

Рисунок 13 – График зависимости времени перемножения матриц размером 100 на 100 в зависимости от количества ядер, линейная шкала.

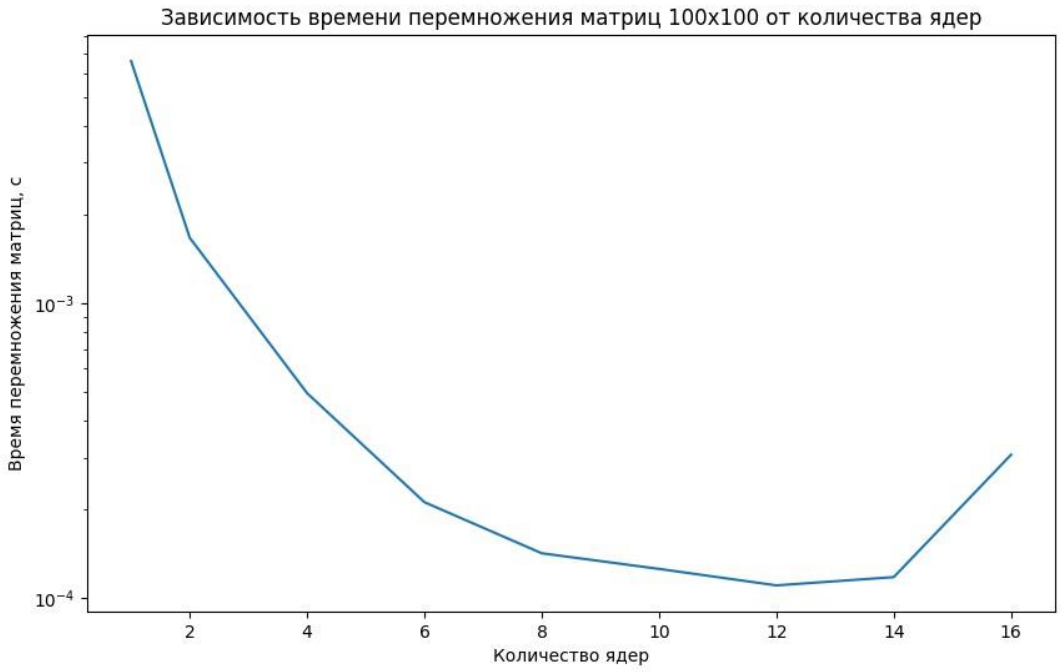

Рисунок 14 – График зависимости времени перемножения матриц размером 100 на 100 в зависимости от количества ядер, логарифмическая шкала.

Как можно увидеть по рисункам 9 и 10, зависимость времени перемножения пары матриц от их размера является не экспоненциальной, а кубической. При этом, чем больше используется ядер при перемножении, тем быстрее оно выполняется.

5 Выводы

В данной лабораторной работе было реализовано перемножение матриц с использованием технологии MPI и проведено несколько экспериментов по перемножению матриц различного размера при различном числе доступных для программы ядер. Эксперименты проводились с использованием суперкомпьютера Самарского Университета «Сергей Королёв».

По результатам проведённой работы можно утверждать, что параллельное перемножение двух матриц тем эффективнее, чем больше вычислительных узлов (ядер) доступно программе. Так, в среднем

перемножение пары матриц размером 1000 на 1000 занимает 0,021 секунд при 16 доступных ядрах и 6,337 секунд при 1 доступном ядре. То есть, программа на 16 вычислительных узлах работает примерно в 300 раз быстрее последовательного исполнения. Следовательно, можно сделать вывод о том, что использование большого числа вычислительных узлов позволяет существенно ускорить выполнение высокопроизводительных программ и открывает широкие возможности для программиста.