МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Параллельные алгоритмы»

Тема: Виртуальные топологии

Студент гр. 3381 _____ Козлов Г.Е.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2025

Цель работы

Цель работы: Получение практических навыков работы с декартовыми топологиями в MPI: создание двумерной решетки процессов, настройка периодических граничных условий и реализация параллельного алгоритма циклического сдвига данных по строкам матрицы.

Задание Вариант 10

Число процессов K является четным: $K = 2N$, $N > 1$; в каждом процессе дано вещественное число A. Определить для всех процессов декартову топологию в виде матрицы размера $2 \times N$ (порядок нумерации процессов оставить прежним) и для каждой строки матрицы осуществить циклический сдвиг исходных данных с шагом 1 (число A из каждого процесса, кроме последнего в строке, пересылается в следующий процесс этой же строки, а из последнего процесса — в главный процесс этой строки). Для определения рангов посылающих и принимающих процессов использовать функцию MPI_Cart_shift, пересылку выполнять с помощью функции MPI_Sendrecv. Во всех процессах вывести полученные данные.

Выполнение работы.

Краткое описние алгоритма программы:

- Инициализация: Запуск MPI, определение общего числа процессов (K) и ранга текущего процесса. Вычисление параметра N=K/2.
- Создание топологии: Формирование виртуальной декартовой решетки процессов размера 2×N с помощью MPI_Cart_create. При этом для измерения строк устанавливается периодичность (замкнутость), а порядок нумерации процессов остается естественным.
- Поиск соседей: Определение рангов процессов для отправки (dest) и приема (source) данных с помощью функции MPI_Cart_shift. Сдвиг выполняется по 1-му измерению (вдоль строк) с шагом +1.

- Обмен данными: Выполнение одновременной отправки своего значения A соседу справа и приема нового значения от соседа слева с использованием функции MPI_Sendrecv. Благодаря настройкам топологии, сдвиг происходит циклически внутри каждой строки.
- Завершение: Вывод полученных значений каждым процессом и освобождение ресурсов MPI.

1. Формальное описание алгоритма

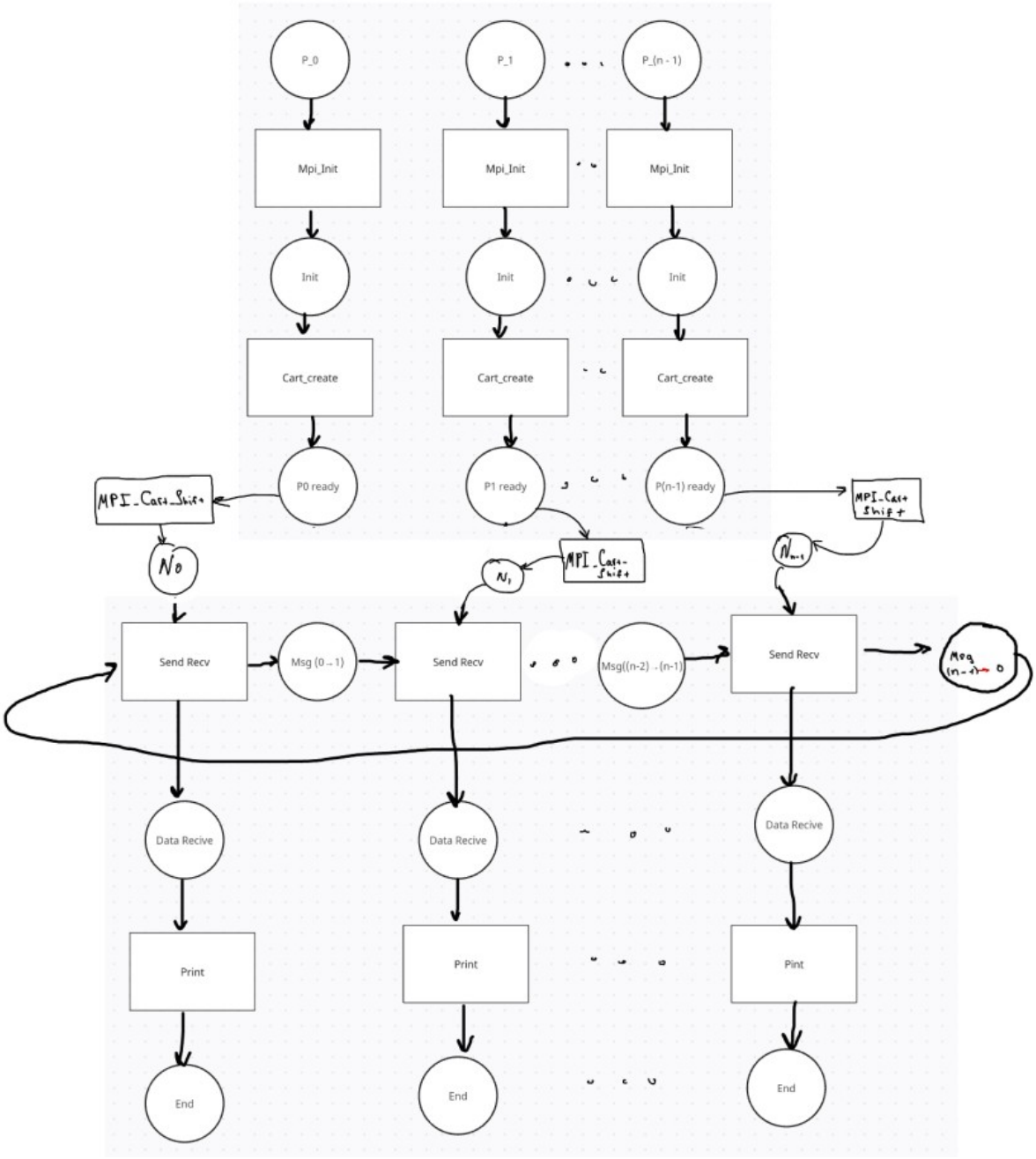

Рисунок 1

Данная схема для строки 0, для строки 1 схема та же, но процессы начинаются не 0, а с N и последний процесс будет 2N - 1. N_i помечаю состояние, когда сосед стал известен

1. Листинг программы:

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[]) {
    int rank, size;
    MPI_Comm comm_cart; // Новый коммуникатор для декартовой топологии
    int dims[2];        // Размеры решетки (2xN)
    int periods[2];     // Периодичность
    int reorder = 0;    // Не переупорядочивать ранги
    int my_coords[2];   // Координаты процесса в решетке
    int rank_source, rank_dest; // Ранги для отправки и получения

    double A_send; // Исходное число A
    double A_recv; // Полученное число A

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Проверка условия K = 2N, N > 1
    if (size % 2 != 0 || size <= 2) {
        if (rank == 0) {
            fprintf(stderr, "Ошибка: Число процессов K=%d должно быть
четным и > 2 (K=2N, N>1).\n", size);
        }
        MPI_Finalize();
        return 1;
    }

    int N = size / 2;

    // Инициализируем данные
    A_send = (double)rank;

    // Определение декартовой топологии 2xN
    dims[0] = 2;
    dims[1] = N;

    // Строки являются циклическими, столбцы нет
    periods[0] = 0;
    periods[1] = 1;

    // новая декартова топология
```

```
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder,
&comm_cart);

    // Получаем свои координаты
    MPI_Cart_coords(comm_cart, rank, 2, my_coords);

    // Определение рангов для циклического сдвига
    MPI_Cart_shift(comm_cart, 1, 1, &rank_source, &rank_dest);

    // Выполнение пересылки с помощью MPI_Sendrecv
    MPI_Sendrecv(
        &A_send, 1, MPI_DOUBLE, rank_dest, 0,
        &A_recv, 1, MPI_DOUBLE, rank_source, 0,
        comm_cart, MPI_STATUS_IGNORE
    );

    // Вывод полученных данных (с упорядочиванием)
    MPI_Barrier(comm_cart);

    int i;
    for (i = 0; i < size; i++) {
        if (rank == i) {
            printf("Ранг %d (Коорд: [%d, %d]): Исходное A = %.1f,
Полученное A = %.1f (от Ранга %d)\n",
                   rank, my_coords[0], my_coords[1], A_send, A_recv,
rank_source);
        }

        MPI_Barrier(comm_cart);
    }

    MPI_Comm_free(&comm_cart);

    MPI_Finalize();
    return 0;
}
}
```

Примеры запуска

```
george@george-ThinkBook:~/pa/lab5$ mpirun -np 4 --oversubscribe ./main
Ранг 0 (Коорд: [0, 0]): Исходное A = 0.0, Полученное A = 1.0 (от Ранга 1)
Ранг 1 (Коорд: [0, 1]): Исходное A = 1.0, Полученное A = 0.0 (от Ранга 0)
Ранг 2 (Коорд: [1, 0]): Исходное A = 2.0, Полученное A = 3.0 (от Ранга 3)
Ранг 3 (Коорд: [1, 1]): Исходное A = 3.0, Полученное A = 2.0 (от Ранга 2)
```

Рисунок 2

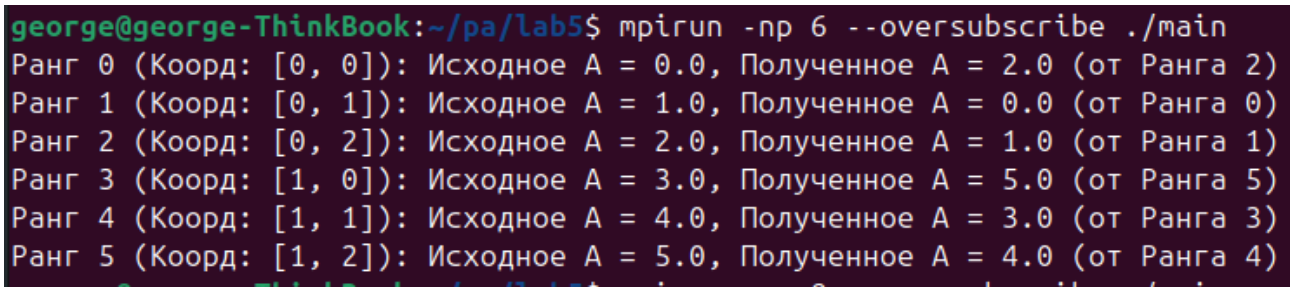

Рисунок 3

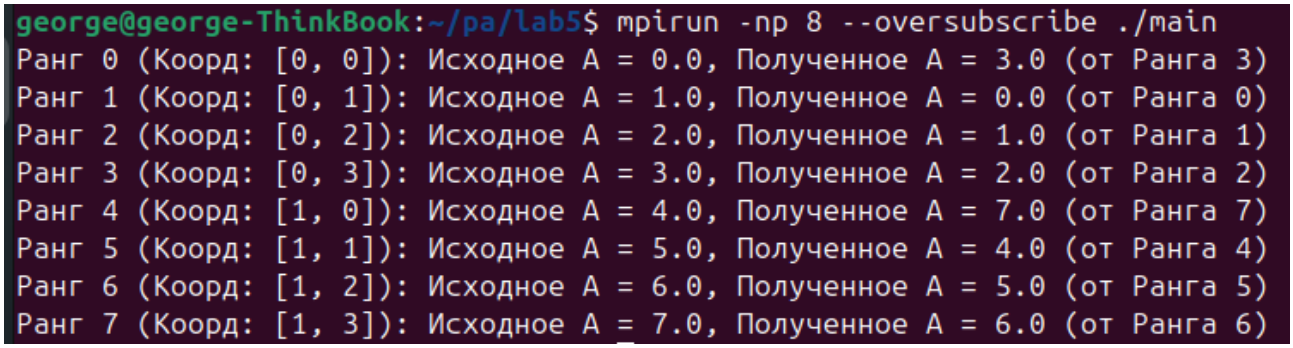

Рисунок 4

2. Распечатка таблицы результатов и времени работы программы

Таблица 1 входные данные: 100 000 элементов

Число процессов (p)	**Время (сек)**	**Ускорение Sp=T4/Tp**	**Эффективность Ep (%)**
4	0.000470	1.00	100.00%
6	0.012807	0.04	2.45%
8	0.013613	0.03	1.73%
16	0.032456	0.01	0.36%
32	0.052766	0.01	0.11%

Таблица 2 входные данные: 1 000 000 элементов

Число процессов (p)	**Время (сек)**	**Ускорение Sp=T4/Tp**	**Эффективность Ep (%)**
4	0.013205	1.00	100.00%
6	0.031557	0.42	27.90%
8	0.070012	0.19	9.43%
16	0.179565	0.07	1.84%
32	0.404525	0.03	0.41%

Таблица 3 входные данные: 10 000 000 элементов

Число процессов (p)	**Время (сек)**	**Ускорение Sp=T4/Tp**	**Эффективность Ep (%)**
4	0.046099	1.00	100.00%
6	0.145022	0.32	21.19%
8	0.213191	0.22	10.81%
16	0.418569	0.11	2.75%
32	43.253504	0.001	0.01%

3. Графики времени выполнения программы в зависимости от объема исходных данных и числа запущенных процессов.

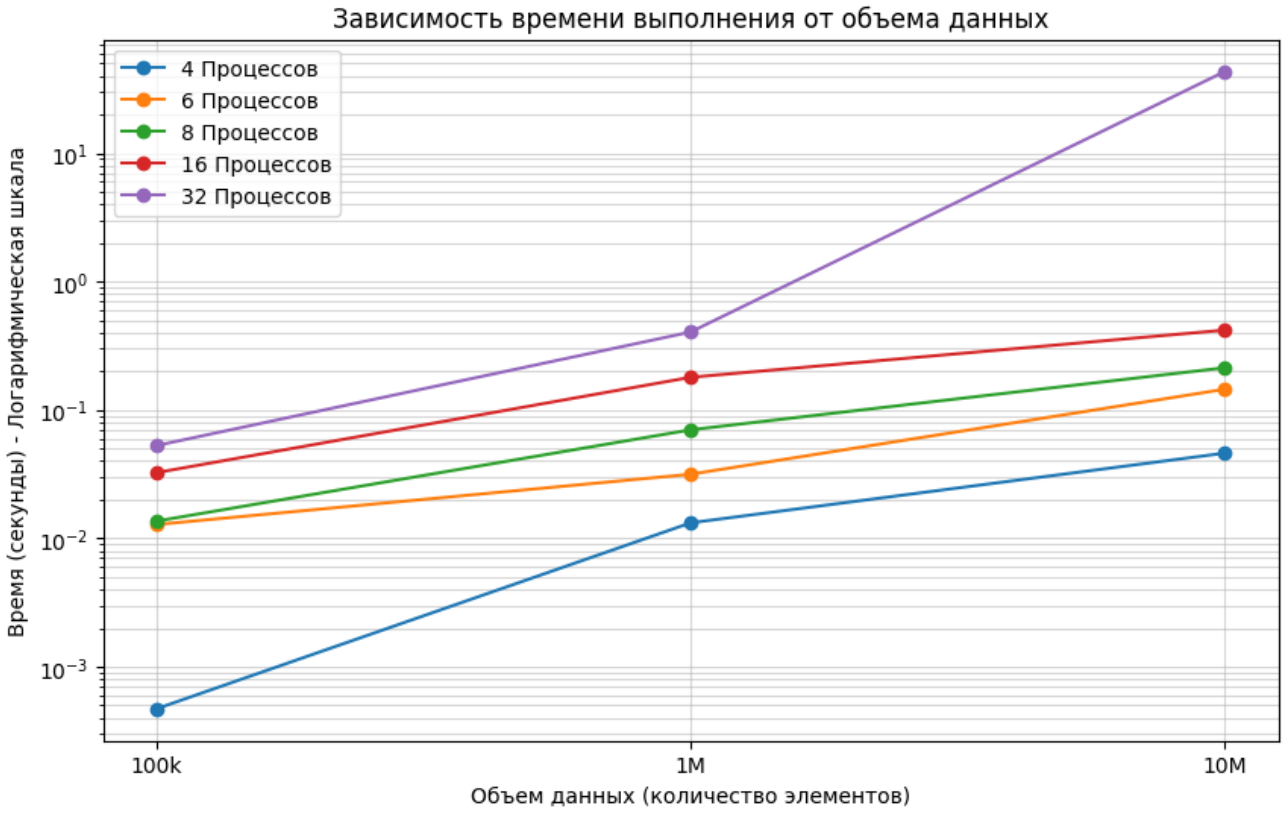

Рисунок 5

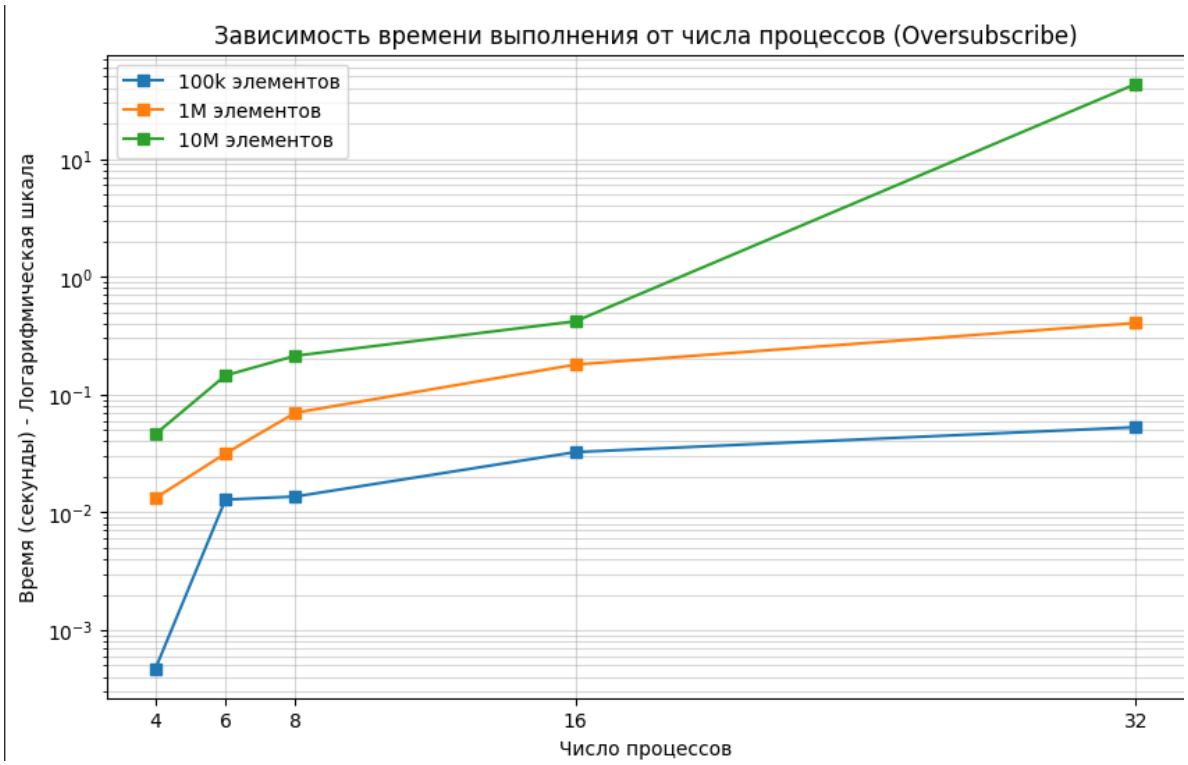

Рисунок 6

4. Графики ускорения на основании результатов, полученных выше.

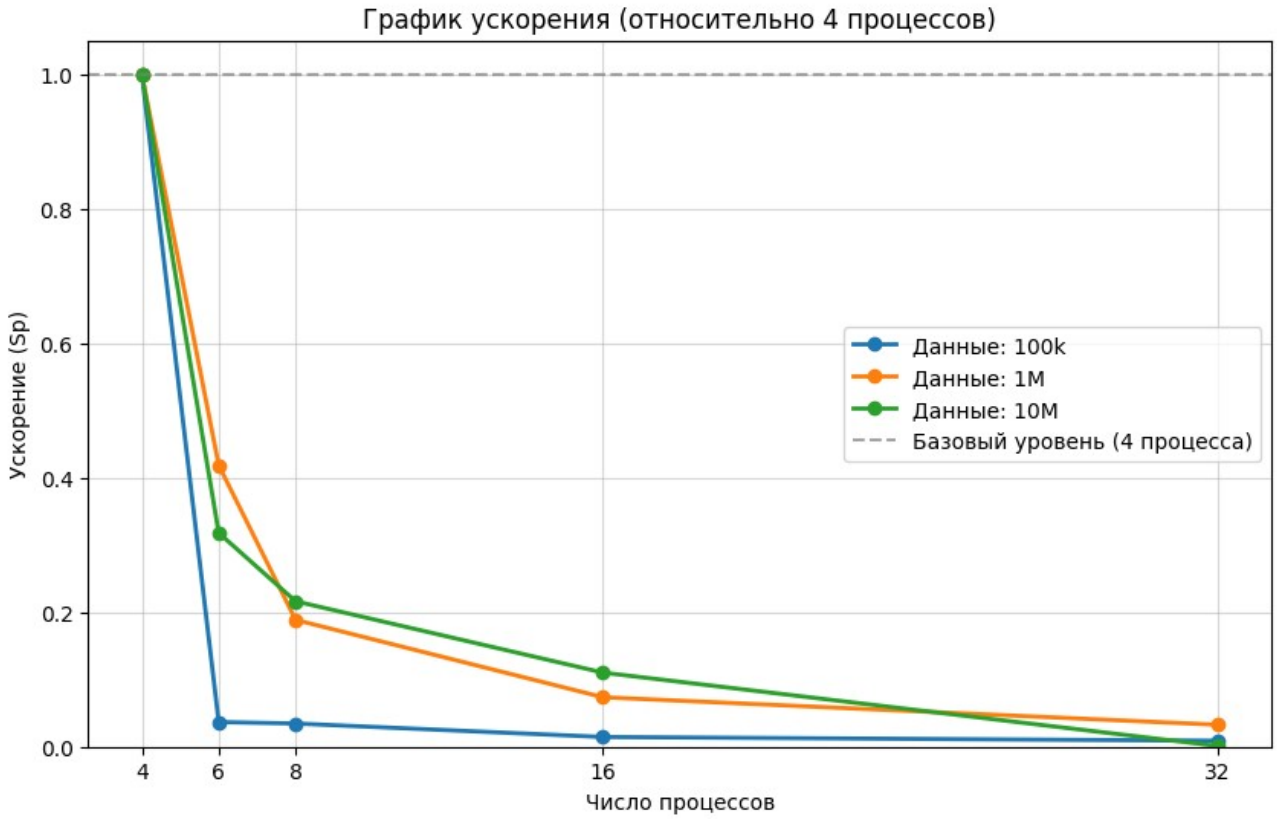

Рисунок 7

Вывод

Анализ полученных результатов демонстрирует, что в условиях запуска задачи с флагом переподписки (oversubscribe) на оборудовании с ограниченным числом физических ядер увеличение количества процессов MPI приводит не к ускорению, а к деградации производительности. Общий тренд графиков показывает замедление вычислений при переходе от 4 к 32 процессам, что объясняется конкуренцией потоков за процессорное время и высокими накладными расходами операционной системы на переключение контекста, которые начинают преобладать над полезной вычислительной работой.

Характерные изломы и резкие скачки на графиках обусловлены двумя различными факторами в зависимости от объема данных. Для малых объемов данных (100 тысяч элементов) наблюдается наиболее резкое падение эффективности сразу после базовой точки в 4 процесса, так как время, затрачиваемое на инициализацию коммуникаций и латентность сети, значительно превышает время самой пересылки данных. Напротив, критический излом графика времени выполнения для 32 процессов на объеме в 10 миллионов элементов, где время подскакивает до аномальных 43 секунд, имеет иную природу и вызван исчерпанием физической оперативной памяти. При запуске 32 процессов, каждый из которых выделяет значительный буфер памяти под массивы типа double, суммарное потребление ресурсов превышает доступный объем RAM, вынуждая операционную систему задействовать механизм своппинга (подкачки) на жесткий диск, что замедляет работу на порядки по сравнению с операциями в оперативной памяти. Таким образом, оптимальной конфигурацией для данной задачи является использование числа процессов, равного числу физических ядер, так как дальнейшее распараллеливание в данном режиме неэффективно из-за архитектурных ограничений.