Министерство образования и науки Российской Федерации

Новосибирский национальный исследовательский государственный университет

Основы параллельного программирования

Отчет по лабораторной работе № 1

«Параллельная реализация решения системы линейных алгебраических уравнений с помощью MPI»

РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ ИТЕРАЦИОННЫМИ МЕТОДАМИ

Студент: Усольцев Антон, гр. 21209

Преподаватель: Мичуров Михаил Антонович

Новосибирск, 2023 г.

1. **Цель работы**

Разработать и исследовать параллельные программы решения СЛАУ методом простой итерации с применением одной из библиотек, реализующих стандарты MPI.

1. **Задачи**
2. Написать программу на языке C или C++, которая реализует итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b в соответствии с выбранным вариантом. Здесь A – матрица размером N×N, x и b – векторы длины N. Тип элементов – double.
3. Программу распараллелить с помощью MPI с разрезанием матрицы A по строкам на близкие по размеру, возможно не одинаковые, части. Соседние строки матрицы должны располагаться в одном или в соседних MPI-процессах. Реализовать вариант программы: векторы x и b дублируются в каждом MPI-процессе,
4. Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: 1,2, 4, 8, 16. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и ε подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд.
5. Выполнить профилирование двух вариантов программы с помощью MPE при использовании 16-и ядер.
6. **Краткое описание подходов к организации решения прикладной задачи параллельными взаимодействующими процессами**

Реализован метод простой итерации и применен к следующим начальным значениям: элементы главной диагонали матрицы A равны 2.0, остальные равны 1.0. Все элементы вектора b равны N+1. В этом случае решением системы будет вектор, элементы которого равны 1.0. Начальные значения элементов вектора x берутся равными 0.

Распараллеливание произведено с помощью разрезания матрицы по строкам на примерно одинаковые куски, а также разрезания вектора b. Каждый процесс считает сумму квадратов своей части вектора b и полученные значения редуцируются в нулевом процессе. Затем каждый процесс умножает свой кусок матрицы на вектор х, вычитает из полученного значения свою часть вектора b и считает сумму квадратов полученного вектора. Эти значения, а также вектор Axn-b собираются в нулевом процессе, который проверяет соответствие параметру ε и делает несколько операций с полным вектором Axn-b. Затем итоговый вектор xn – τ(Axn-b) рассылается по всем процессам и алгоритм повторяется.

Репозиторий с кодом программы:

<https://github.com/AntonyUsoltsev/OPP_lab1_MPI/tree/master/matrix_mpi_task>

1. **Исследование производительности программ**

Начальные данные программы N = 16500, τ = 0.00001, ε = 0.00001

Измерения каждого типа проводились три раза и выбиралось наименьшее время

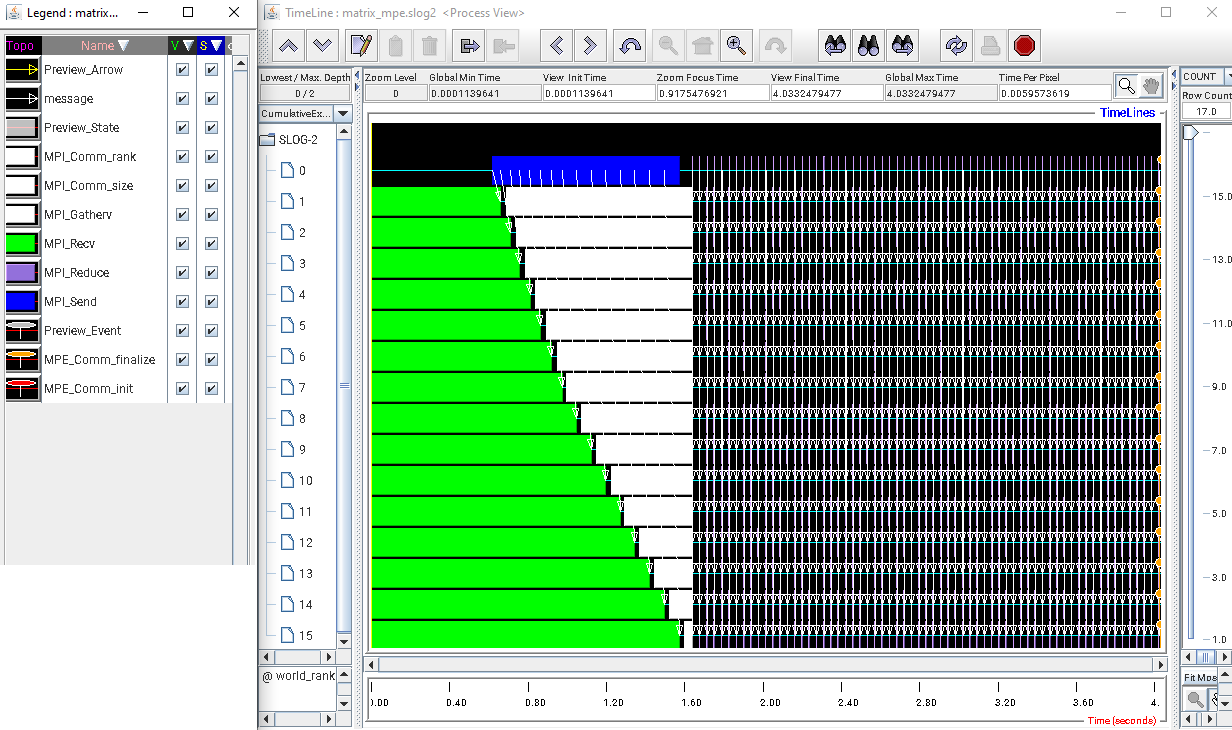
Команда компиляции

|  |  |
| --- | --- |
| **Количество процессов** | **Время, c** |
| Последовательная программа | 24 |
| 1 | 30,5526 |
| 2 | 18,7039 |
| 4 | 8,6201 |
| 8 | 6,1325 |
| 16 | 4,0230 |

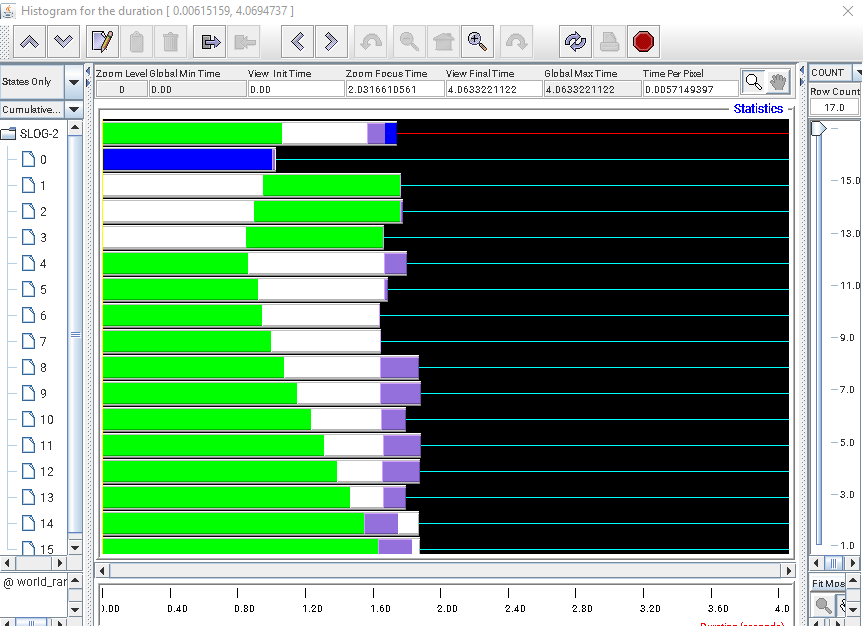
Из графика видно, что при увеличении числа процессов время исполнения программы уменьшается.

Рассчитаем ускорение и эффективность параллельной программы относительно параллельной с использованием одного исполнительного устройства:

1. **Профилирование**



В начале программы нулевой процесс сначала создает матрицу А, а затем рассылает всем остальным процессам куски этой матрицы, из-за чего они долго находятся в состоянии Recv и из-за этого также происходит долгая блокировка при первом вызове Gatherv (нулевой процес не готов принять данные от остальных процессов через Gatherv, потому что еще занят Send-ом матрицы).



Суммарно почти половину времени работы процесса он простаивает в ожидании Send/Recv/Gather

1. **Заключение**

* Реализован вариант программы с разрезанием матрицы и вектора b
* В результате измерений получили, что с увеличением числа процессов время исполнения программы уменьшается
* При исполнении программы на 4 процессах заметен скачок во времени исполнения и, соответственно, скачки на графиках ускорения и эффективности
* Последовательность взаимодействий соответствует алгоритму (описано выше). Процент времени простоя в ожидании данных около 50% (для 16 процессов), что является довольно большим показателем. Профиль дает следующие идеи для оптимизации алгоритма: 1) оптимизация распределения кусков матрицы по процессам или полный отказ от рассылки кусков, а прямое создание нужного куска в самом процессе (но для этого необходимо изначально знать общую конфигурацию матрицы);

2) использование большего числа встроенных MPI методов (например для той же рассылки)

* Прочие результаты: произведено подключение WSL и кластера внутрь Clion, что оказалось очень удобным

Новый код с использованием Scatterv

|  |  |
| --- | --- |
| **Количество процессов** | **Время, c** |
| 1 | 25,1069 |
| 2 | 18,9331 |
| 3 | 11,8556 |
| 4 | 11,7746 |
| 5 | 10,7525 |
| 8 | 6,4287 |
| 16 | 5,4514 |