Министерство образования и науки Российской Федерации

Новосибирский национальный исследовательский государственный университет

Основы параллельного программирования

Отчет по лабораторной работе № 1

Студент: Нелтанов Б. В.

Преподаватель: Мичуров М. А.

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. **Краткое описание подходов к организации решения прикладной задачи параллельными взаимодействующими процессами**

Реализован 1 подход к организации параллельной программы при умножении матрицы на вектор: векторы x и b дублируются в каждом MPI-процессе.

Ссылка на git репозиторий с программой и файлами, предназначенными для работы с вычислительным кластером:

<https://github.com/neltanov/opp>

*Команда для компиляции:*

mpicc -std=gnu99 -O2 -Wpedantic -Wall -Werror -o main main.c -lm

*Команда для запуска (на вычислительном кластере)*

mpirun -machinefile $PBS\_NODEFILE -np $MPI\_NP .main

1. **Исследование производительности программ**

Ссылка на таблицу с измерениями:

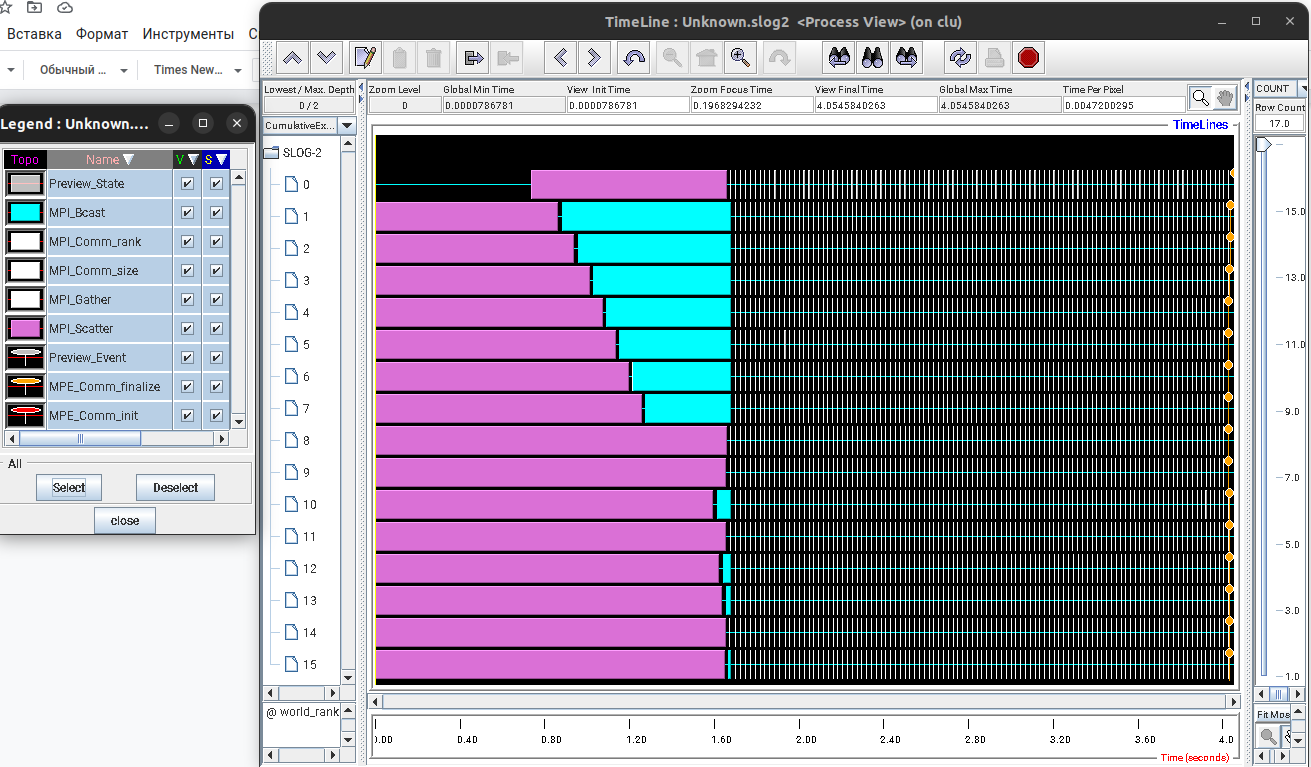
[Time counting opp\_lab1](https://docs.google.com/spreadsheets/d/1iNJtUcL7KdO9zKl3sx3Tx2xJDZfU6ZJ_XxjV0FTTpqA/edit#gid=193252643)

Как видим, распараллелив программу между процессами, мы получаем значительное ускорение времени работы программы. В основном, время работы программы чуть выше, чем предполагаемое ввиду осуществления коммуникации между процессами. На 16 ядрах нам пришлось задействовать 2 узла, т. е. 2 компьютера. Из-за этого очень много времени ушло на коммуникацию между узлами, поэтому эффективность работы программы на 16 ядрах заметно ухудшилась.

1. **Профилирование**

Команда для компиляции программы с профилированием:

mpecc -mpilog -std=gnu99 -O2 -Wpedantic -Wall -Werror -o mpe\_main main.c -lm  
В результате создается файл с расширением .clog2, который запускаем с помощью команды jumpshot:



По временной линии видим, что почти половина времени уходит на то, чтобы инициализировать матрицу в корневом процессе (под номером 0) и разделить между остальными процессами, а для этого мы пользуемся функцией MPI\_Scatter, из-за которого образуется простой в ожидании данных от корневого процесса. То есть львиная доля времени уходит на коммуникацию между процессами. При увеличении количества процессов количество коммуникаций увеличивается, следовательно и времени на это уходит гораздо больше. Ввиду этого ухудшается эффективность распараллеливания программы.

1. **Заключение**

* Программа распараллеливания решения системы линейных уравнений реализована.
* Характеристики производительности удовлетворительные, так как я добился значительного ускорения программы.
* С увеличением количества ядер время выполнения программы уменьшается, ускорение увеличивается, а эффективность распараллеливания понемногу падает ввиду увеличения времени коммуникации между процессами.
* На профиле: последовательность взаимодействий полностью соответствует алгоритму, процент времени простоя в ожидании данных около 40%. Профиль позволяет увидеть, что во время простоя можно было бы занять более полезными вещами.
* Если брать размеры матрицы, не кратные числу процессов, то время работы программы немного увеличивается, ввиду того, что число коммуникаций увеличивается (MPI\_Send и MPI\_Recv при пересылке оставшейся части массива).