МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий
Кафедра параллельных вычислений

Основы параллельного программирования

Отчет

О выполнении работы № 5

Работу выполнил:
Е. И. Биточкин
Группа: 22209
Преподаватель:
А. А. Артюхов

Новосибирск
2024

Содержание

1. Цель

- Освоить разработку многопоточных программ с использованием POSIX Threads API.
- Познакомиться с задачей динамического распределения работы между процессорами.

2. Задание

- Разработка политики коммуникации между процессами.
- Реализация алгоритма.
- Измерение эффективности.

3. Описание работы

3.1. Описание алгоритма

При использовании балансировки в каждом процессе создается 3 потока:

Поток вычислитель не делает MPI запросов, а занимается только полезной работой.

Поток-сервер ждет запросов от других процессов. При получении запроса он сравнивает число задач и в случае удвлетворения прошения посылает массив с задачами. В случае неудвлетворения зарпоса также посылает массив, но пустой, и с тэгом, сигнализирующем о том, что запрос не удвлетворен. Также сохраняется информация о том, что у просившего нет задач, а значит нет смысла обращаться к нему за добавкой.

Поток-клиент начинает работать, когда число задач у процесса доходит до определенной границы. Клиент по очереди запрашивает у всех процессов задачи (Рисунок 3.2) и не останавливается, пока не получит задачу (или не опросит всех). Данные о запросе на соседний процесс также как и в потоке сервере сохраняются.

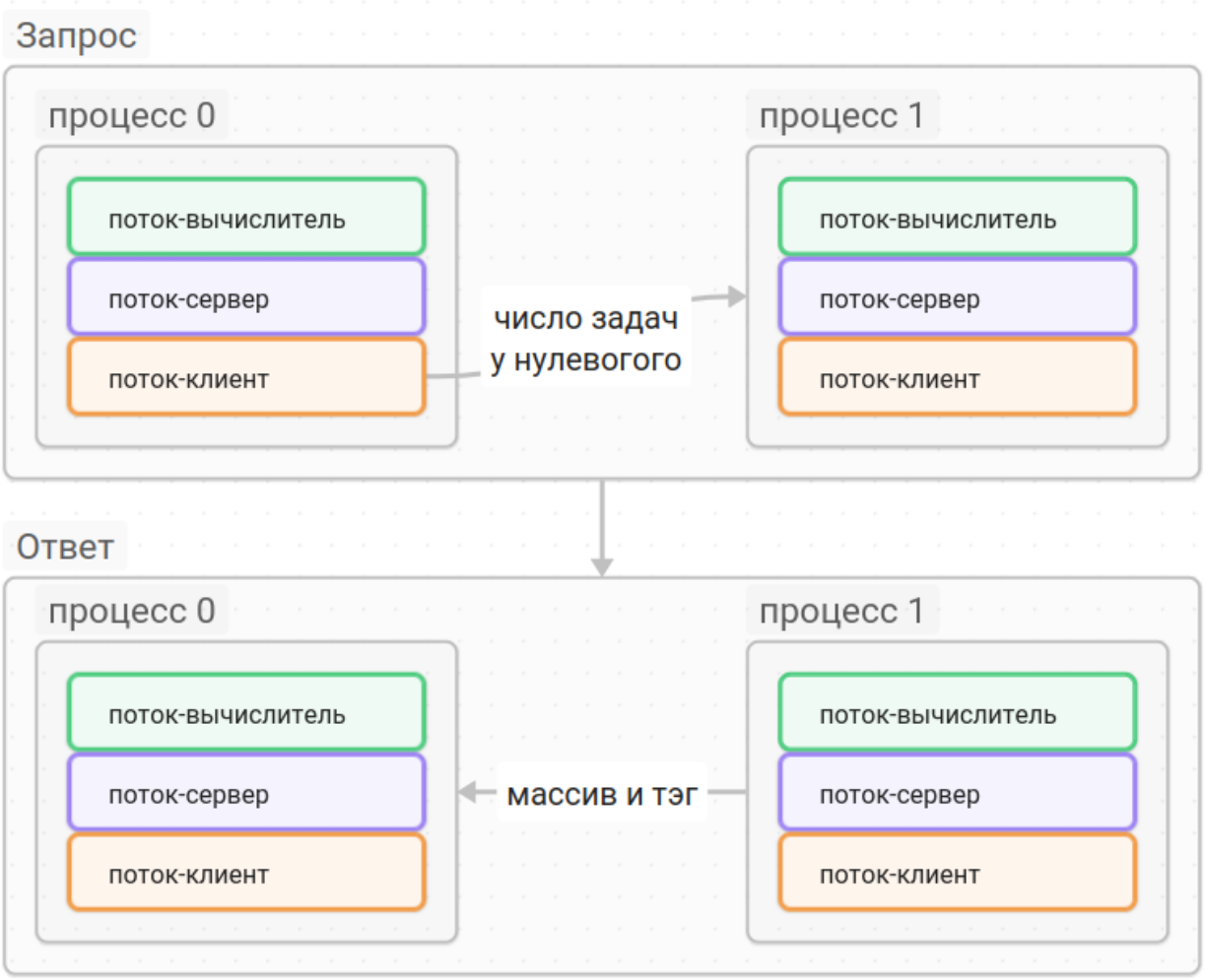


Рисунок 3.1. Описание алгоритма

3.2. Профилирование

Из статистики вызовов (5) видно, что вызовы MPI функций влияют на время работы минимально.

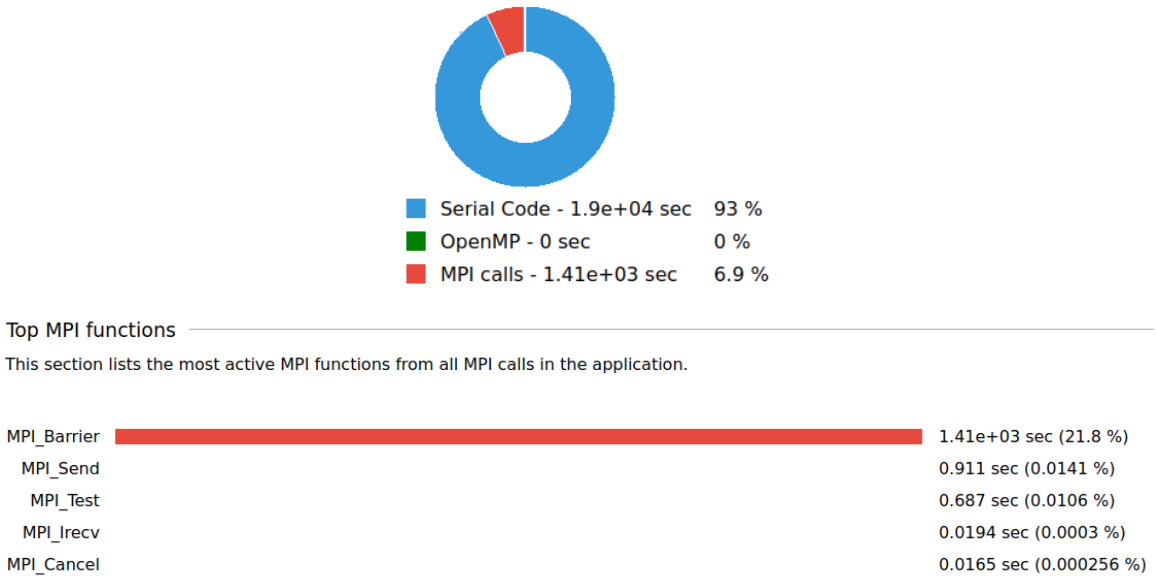


Рисунок 3.2. Статистика MPI вызовов

Всего запросов на балансировку (MPI_Send) было не много, и они не сильно нагружали систему.

Число MPI_Irecv, MPI_Test обусловленно их постоянным повторением.

5772 из 5788 вызовов было отменено по таймауту (6).

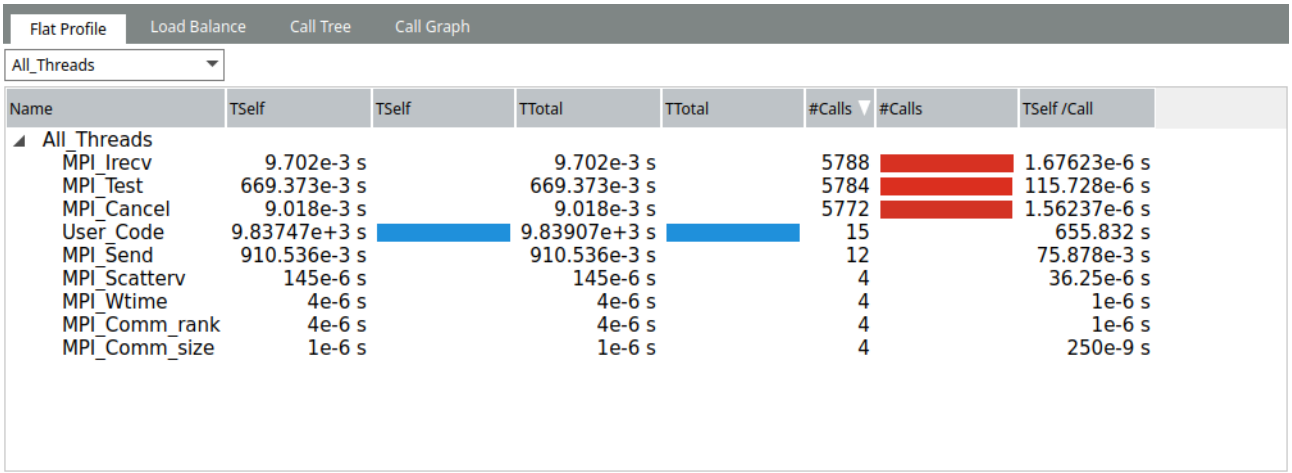


Рисунок 3.3. Статистика MPI вызовов по колличеству

Можно посмотреть статистику вызовов в одном процессе:

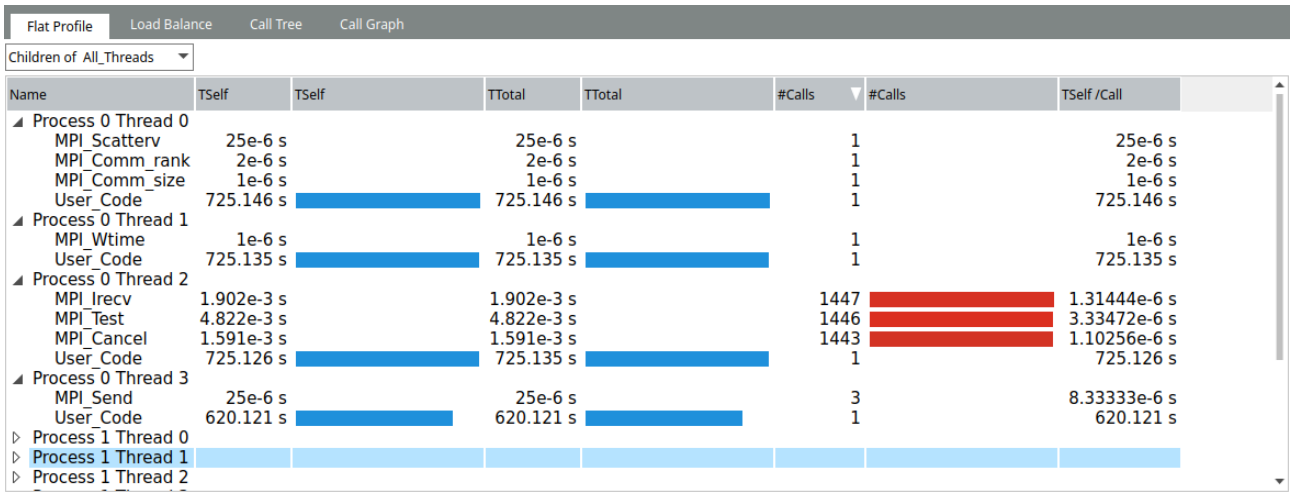


Рисунок 3.4. MPI вызовы для одного процесса (без подсчета значений)

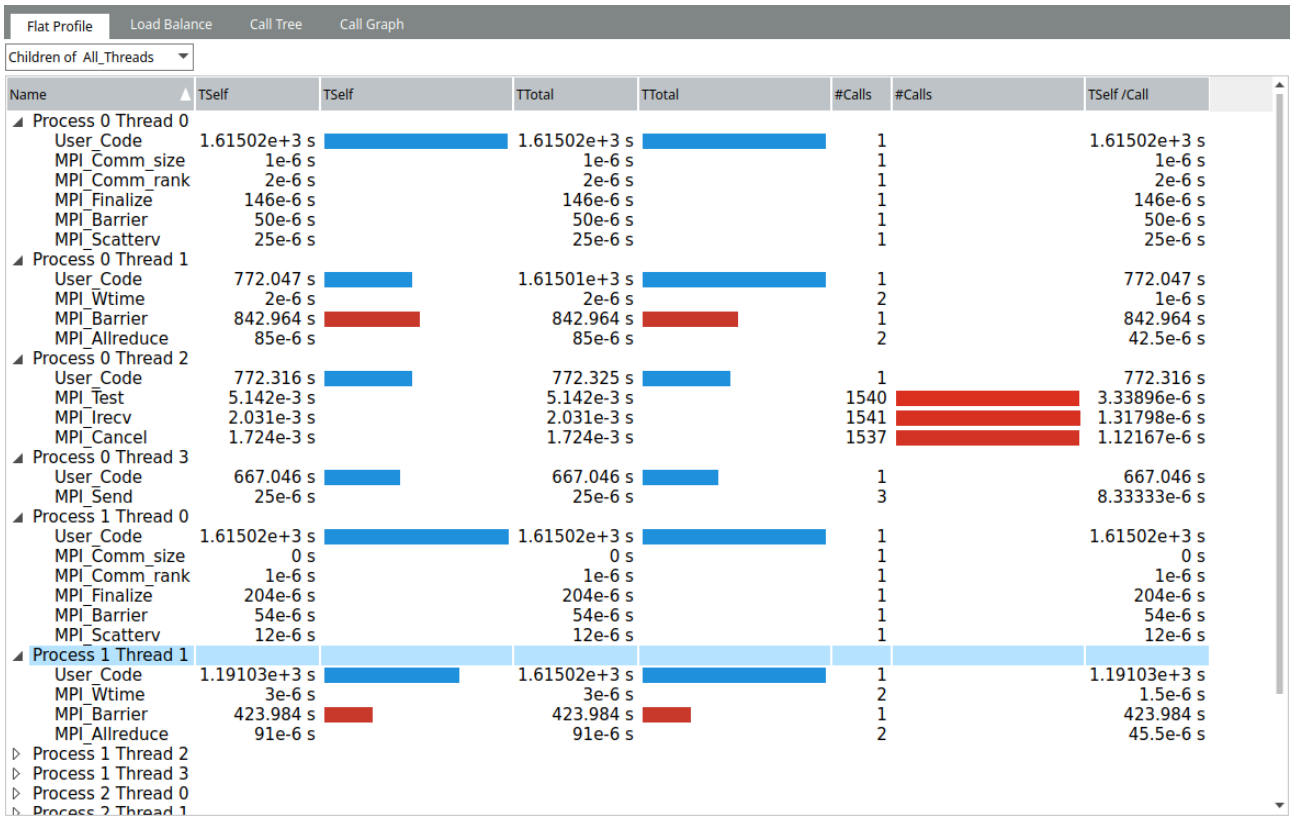


Рисунок 3.5. Полная картина MPI вызовов для одного процесса

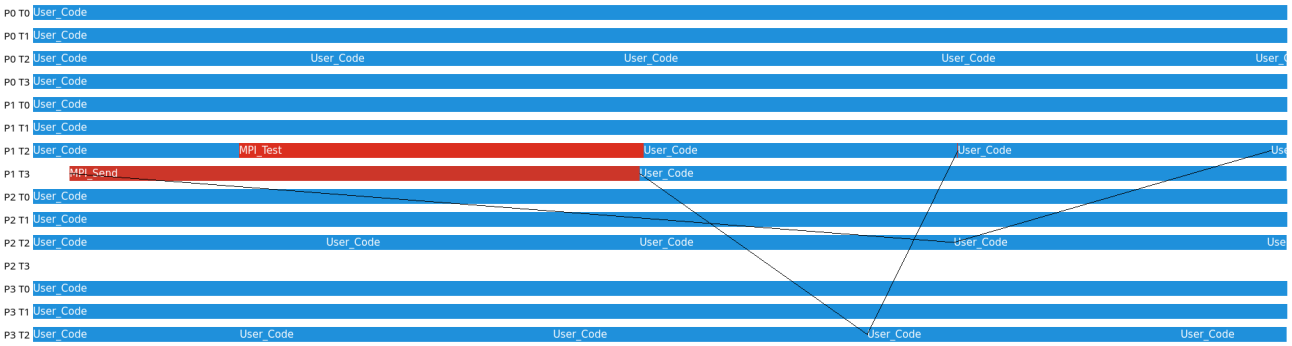


Рисунок 3.6. Пример коммуникации

На рисунке 7 видно, как процесс P1 из потока T3 отправляет запросы на P2 и P3, и получает ответы в T2.

3.3. Замеры

3.3.1. Зависимость времени выполнения от числа узлов

Для тестирования было взято 20 задач, 4 процесса. По очереди проводилось тестирование на 1, 2 и 4 узлах кластера.

no balance		
count nodes	max time, sec	Disbalance, %
1	2200,02559	85,22
2	2200	85,2
4	2199,999464	85,22

balanced		
count nodes	max time, sec	Disbalance, %
1	1615,124794	52,19
2	1615,040122	52,19
4	1615,084335	52,19

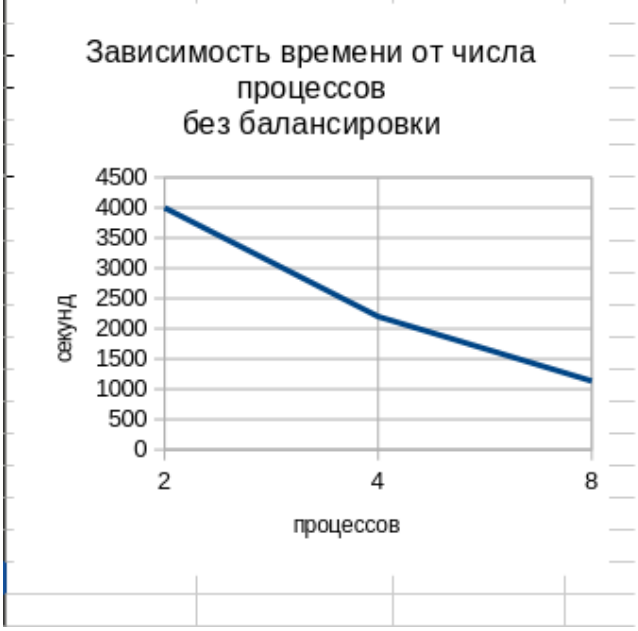
Рисунок 3.7. Зависимость от числа узлов

Как видно из 8, время не зависит от числа узлов.

3.3.2. Зависимость времени выполнения от числа процессов

no balance		
count process	max time, sec	disbalance, %
2	3998	73,11
4	2199,999464	85,22
8	1133,99845	91,97

balanced		
count process	max time, sec	disbalance, %
2	3576,060481	58,77
4	1615,025193	52,19
8	1017,1	82,4



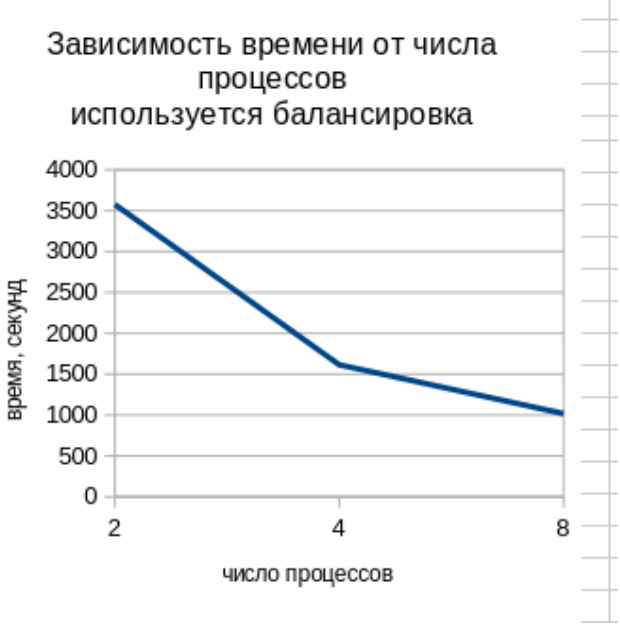


Рисунок 3.8. 100 задач, от 1 до 100, равномерно разделены между процессами

Как видно из графиков (8) эффективность балансировки падает. Очевидно, что чем больше процессов, тем меньше задач на процесс, а значит у алгоритма балансировки меньше времени на работу.

Очевидно, что изначальный дизбаланс сильно влияет на то, на сколько эффективной будет балансировка. Однако было предположено, что эффективность балансировки зависит и от числа задач на процессе.

balanced			
count process	max time, sec	tasks count	disbalance, %
2	531	40	45
4	945	80	40
8	1931	160	35

Рисунок 3.9. Равномерный рост числа задач

По результатам замеров (9) можно предположить, что выдвинутая гипотеза верна.

4. Заключение

- Был реализован алгоритм балансировки.
- Была произведена оценка эффективности алгоритма балансировки.

5. Приложение

5.1. Исходный код

https://github.com/BigCubeCat/bpp_labs.git