

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	3
2. Задание	3
3. Описание работы	4
3.1. Описание алгоритма	4
3.2. Профилирование	5
3.3. Замеры	8
3.3.1. Зависимость времени выполнения от числа узлов	8
3.3.2. Зависимость времени выполнения от числа процессов	8
4. Заключение	10
5. Приложение	10
5.1. Исходный код	10

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 удовлетворения прошения посылает массив с задачами. В случае неудовлетворения запроса также посылает массив, но пустой, и с тэгом, сигнализирующем о том, что запрос не удовлетворен. 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.

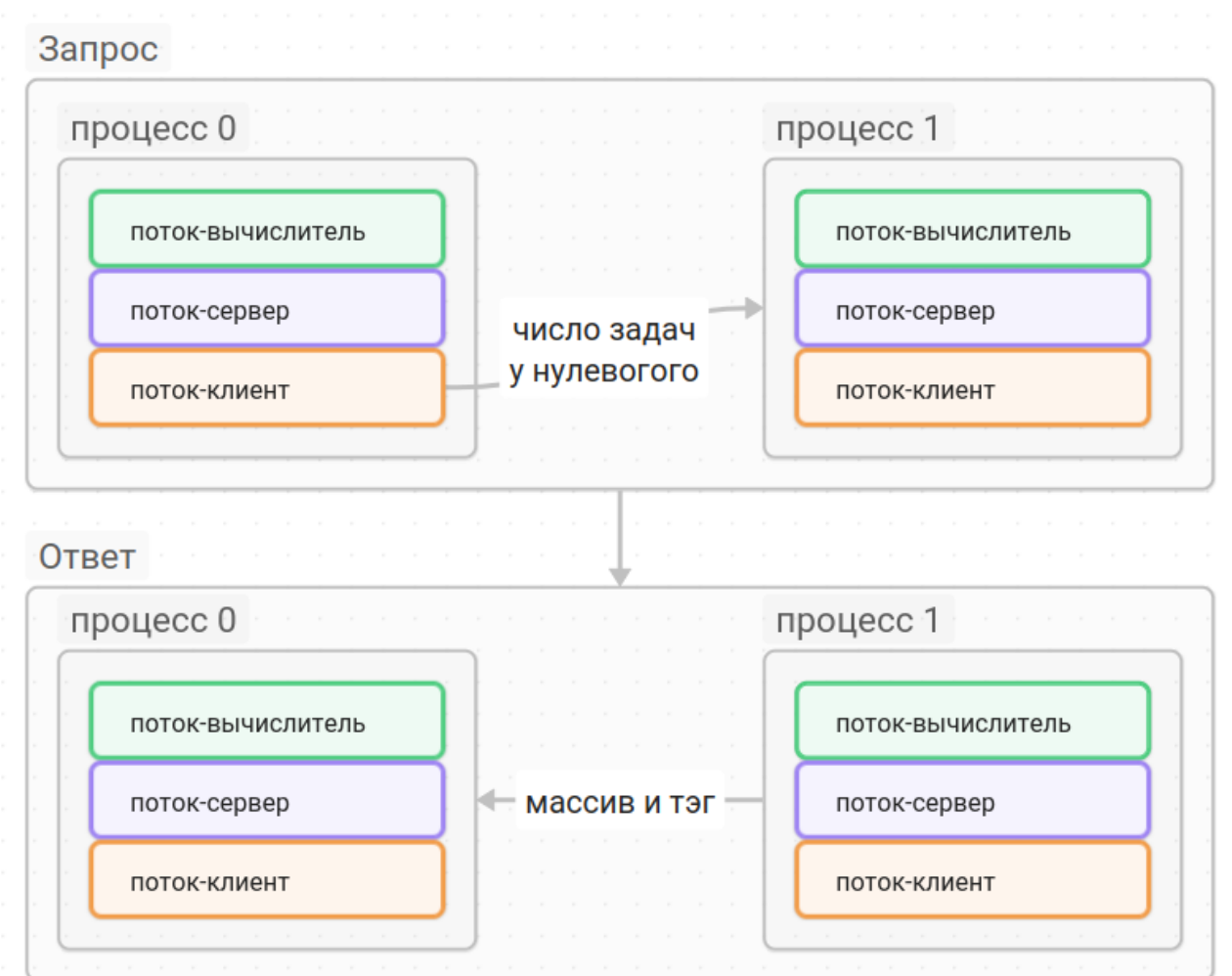


Рисунок 3.1. Описание алгоритма

3.2. Профилирование

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

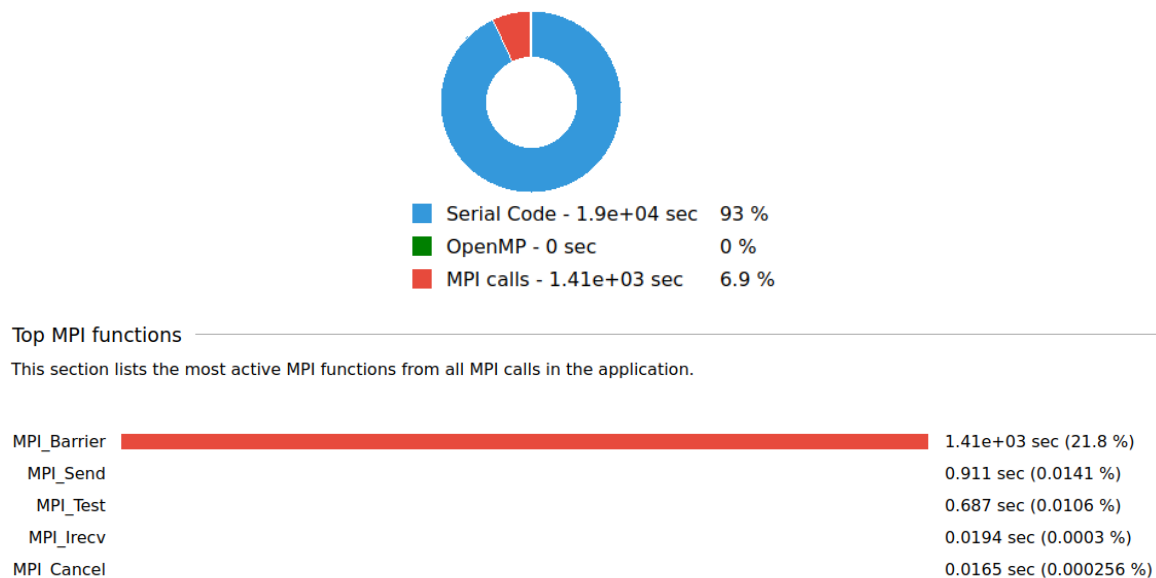


Рисунок 3.2. Статистика MPI вызовов

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

Число MPI_Irecv, MPI_Test обусловлено их постоянным повторением.

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

Flat Profile Load Balance Call Tree Call Graph							
All_Threads							
Name	TSelf	TSelf	TTotat	TTotat	#Calls	#Calls	TSelf /Call
▲ All_Threads							
MPI_Irecv	9.702e-3 s		9.702e-3 s		5788		1.67623e-6 s
MPI_Test	669.373e-3 s		669.373e-3 s		5784		115.728e-6 s
MPI_Cancel	9.018e-3 s		9.018e-3 s		5772		1.56237e-6 s
User Code	9.83747e+3 s		9.83907e+3 s		15		655.832 s
MPI_Send	910.536e-3 s		910.536e-3 s		12		75.878e-3 s
MPI_Scatterv	145e-6 s		145e-6 s		4		36.25e-6 s
MPI_Wtime	4e-6 s		4e-6 s		4		1e-6 s
MPI_Comm_rank	4e-6 s		4e-6 s		4		1e-6 s
MPI_Comm_size	1e-6 s		1e-6 s		4		250e-9 s

Рисунок 3.3. Статистика MPI вызовов по количеству

Можно посмотреть статистику вызовов в одном процессе:

Flat Profile Load Balance Call Tree Call Graph							
Children of All_Threads							
Name	TSelf	TSelf	TTotal	TTotal	#Calls	#Calls	TSelf /Call
Process 0 Thread 0							
MPI_Scatterv	25e-6 s		25e-6 s		1		25e-6 s
MPI_Comm_rank	2e-6 s		2e-6 s		1		2e-6 s
MPI_Comm_size	1e-6 s		1e-6 s		1		1e-6 s
User_Code	725.146 s		725.146 s		1		725.146 s
Process 0 Thread 1							
MPI_Wtime	1e-6 s		1e-6 s		1		1e-6 s
User_Code	725.135 s		725.135 s		1		725.135 s
Process 0 Thread 2							
MPI_Irecv	1.902e-3 s		1.902e-3 s		1447		1.31444e-6 s
MPI_Test	4.822e-3 s		4.822e-3 s		1446		3.33472e-6 s
MPI_Cancel	1.591e-3 s		1.591e-3 s		1443		1.10256e-6 s
User_Code	725.126 s		725.135 s		1		725.126 s
Process 0 Thread 3							
MPI_Send	25e-6 s		25e-6 s		3		8.33333e-6 s
User_Code	620.121 s		620.121 s		1		620.121 s
Process 1 Thread 0							
Process 1 Thread 1							
Process 1 Thread 2							

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

Flat Profile Load Balance Call Tree Call Graph							
Children of All_Threads							
Name	TSelf	TSelf	TTotal	TTotal	#Calls	#Calls	TSelf /Call
Process 0 Thread 0							
User_Code	1.61502e+3 s		1.61502e+3 s		1		1.61502e+3 s
MPI_Comm_size	1e-6 s		1e-6 s		1		1e-6 s
MPI_Comm_rank	2e-6 s		2e-6 s		1		2e-6 s
MPI_Finalize	146e-6 s		146e-6 s		1		146e-6 s
MPI_Barrier	50e-6 s		50e-6 s		1		50e-6 s
MPI_Scatterv	25e-6 s		25e-6 s		1		25e-6 s
Process 0 Thread 1							
User_Code	772.047 s		1.61501e+3 s		1		772.047 s
MPI_Wtime	2e-6 s		2e-6 s		2		1e-6 s
MPI_Barrier	842.964 s		842.964 s		1		842.964 s
MPI_Allreduce	85e-6 s		85e-6 s		2		42.5e-6 s
Process 0 Thread 2							
User_Code	772.316 s		772.325 s		1		772.316 s
MPI_Test	5.142e-3 s		5.142e-3 s		1540		3.33896e-6 s
MPI_Irecv	2.031e-3 s		2.031e-3 s		1541		1.31798e-6 s
MPI_Cancel	1.724e-3 s		1.724e-3 s		1537		1.12167e-6 s
Process 0 Thread 3							
User_Code	667.046 s		667.046 s		1		667.046 s
MPI_Send	25e-6 s		25e-6 s		3		8.33333e-6 s
Process 1 Thread 0							
User_Code	1.61502e+3 s		1.61502e+3 s		1		1.61502e+3 s
MPI_Comm_size	0 s		0 s		1		0 s
MPI_Comm_rank	1e-6 s		1e-6 s		1		1e-6 s
MPI_Finalize	204e-6 s		204e-6 s		1		204e-6 s
MPI_Barrier	54e-6 s		54e-6 s		1		54e-6 s
MPI_Scatterv	12e-6 s		12e-6 s		1		12e-6 s
Process 1 Thread 1							
User_Code	1.19103e+3 s		1.61502e+3 s		1		1.19103e+3 s
MPI_Wtime	3e-6 s		3e-6 s		2		1.5e-6 s
MPI_Barrier	423.984 s		423.984 s		1		423.984 s
MPI_Allreduce	91e-6 s		91e-6 s		2		45.5e-6 s
Process 1 Thread 2							
Process 1 Thread 3							
Process 2 Thread 0							
Process 2 Thread 1							

Рисунок 3.5. Полная картина MPI вызовов для одного процесса

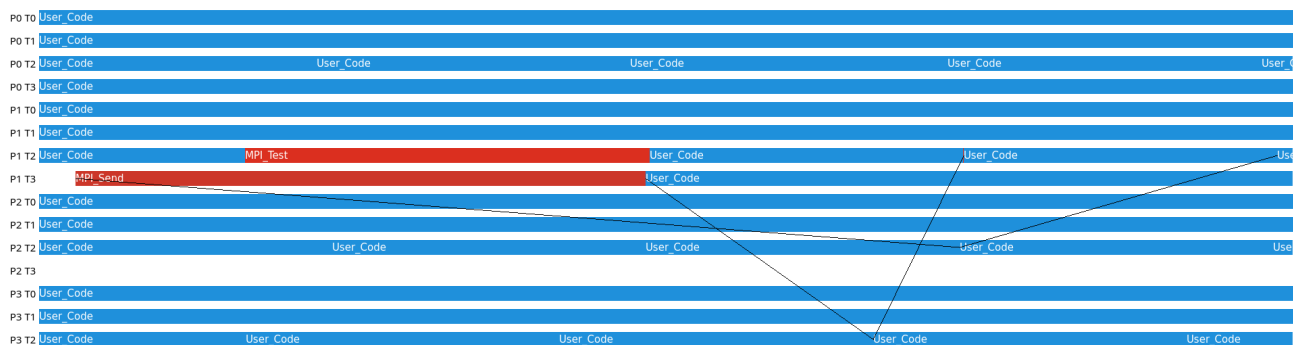


Рисунок 3.6. Пример коммуникации

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				balanced		
count nodes	max time, sec	Disbalance, %		count nodes	max time, sec	Disbalance, %
1	2200,02559	85,22		1	1615,124794	52,19
2	2200	85,2		2	1615,040122	52,19
4	2199,999464	85,22		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				balanced		
count process	max time, sec	disbalance, %		count process	max time, sec	disbalance, %
2	3998	73,11		2	3576,060481	58,77
4	2199,999464	85,22		4	1615,025193	52,19
8	1133,99845	91,97		8	1017,1	82,4

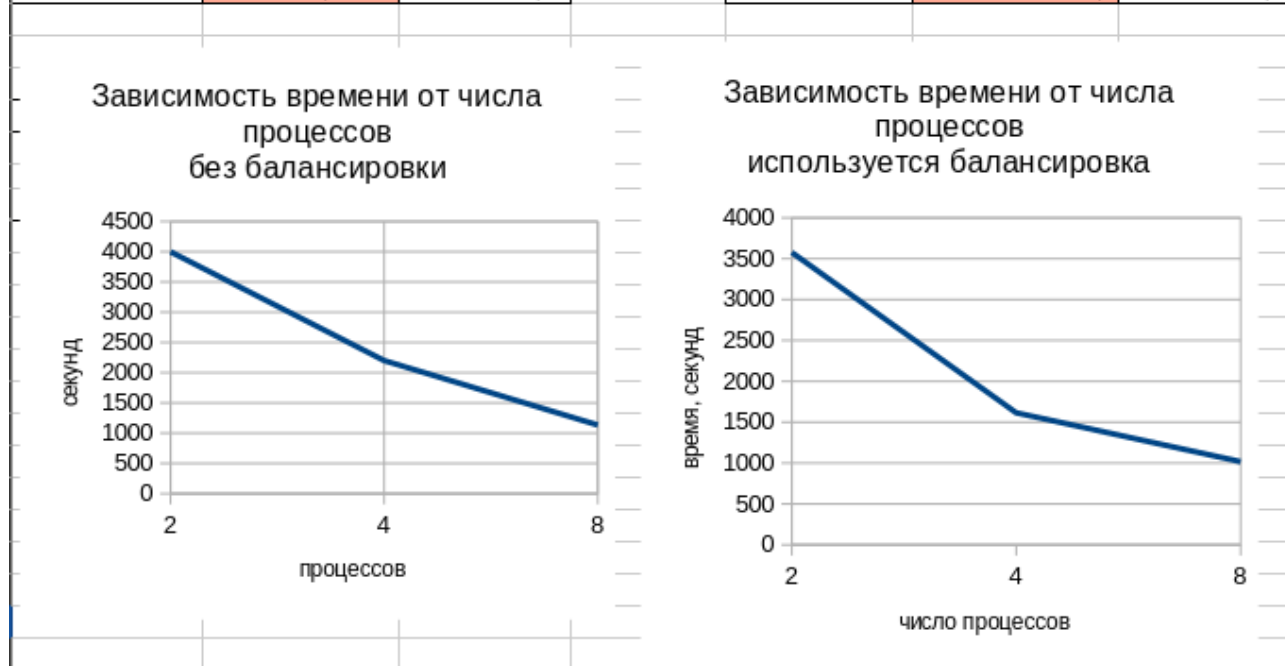


Рисунок 3.8. 100 задач, от 1 до 100, равномерно разделены между процессами

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 дисбаланс 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