



**Спецкурс: системы и средства параллельного
программирования.**

Отчёт № 3.

**Анализ параллельной оптимизации решета Эратосфена
при помощи MPI**

Работу выполнил
Федоров В. В.

Постановка задачи и формат данных.

Задача: Реализовать параллельный параллельную версию решета Эратосфена — алгоритма поиска простых чисел на отрезке $[lo, hi]$ — при помощи технологии MPI, и сделать замеры:

- Максимального времени работы процессов
- Суммарного времени работы процессов

Формат командной строки: <левая граница отрезка lo> <правая граница отрезка hi> <файл для вывода простых чисел> [<файл для вывода суммарного времени работы всеми процессами> <файл для вывода максимального времени работы среди всех процессов>]
В случае отсутствия двух последних аргументов два замера будут выведены на стандартный поток.

Описание алгоритма.

Математическая постановка: Алгоритм решета Эратосфена для поиска простых чисел на отрезке $[1, N]$ можно описать так:

- Для всех чисел k от 2 до N выполнить:
 - Если k не вычеркнуто:
 - То добавить в k в список простых чисел
 - Для всех чисел p от $k+1$ до N :
 - Если p кратно k , то вычеркнуть его

Тогда мы получим список простых чисел на этом отрезке. Поиск простых чисел на отрезке $[M, N]$ отличается лишь тем, что из конечного списка нужно вычеркнуть все числа, меньшие M .

Легко показать, что если число k составное, то у него есть делитель, меньший либо равный квадратному корню из k . Поэтому для простых чисел, больших корня из N , нет смысла проводить цикл вычеркивания. Отсюда следует следующий параллельный алгоритм решета Эратосфена с использованием P процессов:

- Обычным последовательным алгоритмом найти простые числа на $[1, \lfloor \sqrt{N} \rfloor]$
- Поделить отрезок $[\lfloor \sqrt{N} \rfloor + 1, N]$ на P равных частей, отдать каждую часть отдельному процессу
- Каждый процесс на своем подотрезке находит все числа, не кратные ни одному из найденных в первом пункте простые числа

Совокупность полученных чисел и будет являться списком всех простых чисел на $[1, N]$.

Анализ данных программы: Для оценки времени работы программы использовалась функция `MPI_Wtime`, а также функция `MPI_Reduce` для получения максимума/суммы по всем процессам.

Верификация: Для проверки корректности работы программы полученный список простых чисел сверялся со списком, заранее полученным при помощи последовательного алгоритма.

Результаты выполнения.

Результаты:

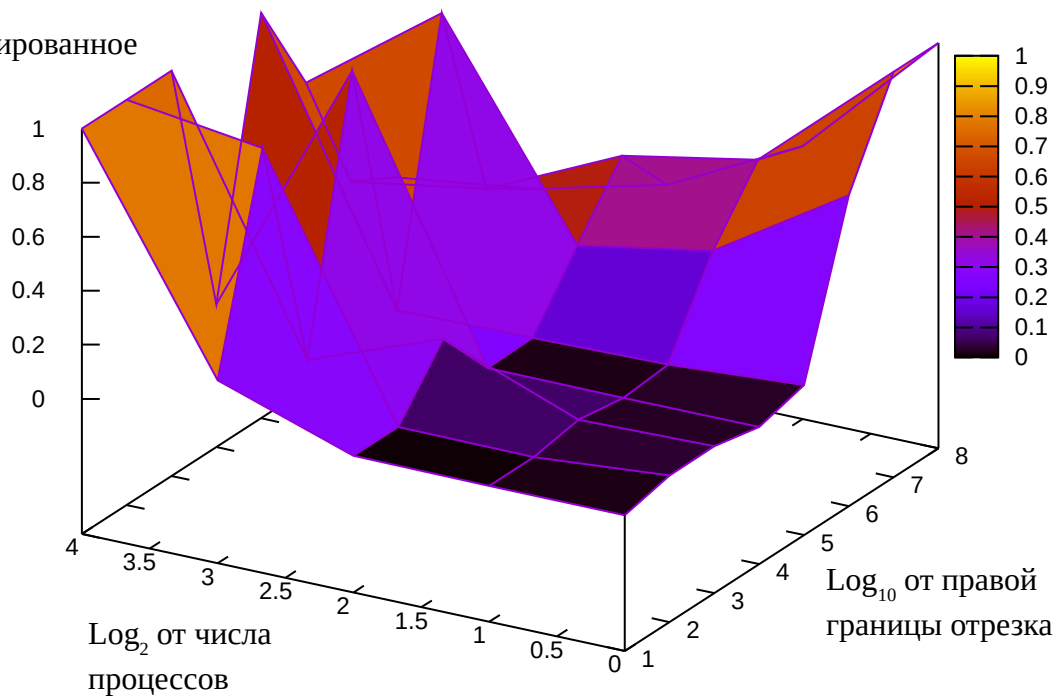
Замеры работы программы проводились для всех пар (N, P) , где N — правая граница отрезка, $N \in 10, 10^2, \dots, 10^8$, P — число процессов, $P \in 1, 2, 4, 8, 16$. Для каждой пары замеры проводились 5 раз, после чего значение усреднялось. Для лучшей наглядности на графиках отображено не абсолютное время, а поделенное на максимальное время для данного N среди всех P . На осях отображены логарифмы от N и P по основаниям 10 и 2 соответственно.

Основные выводы.

Maximum time measurements

'maxtime_grid.dat' u 2:1:3

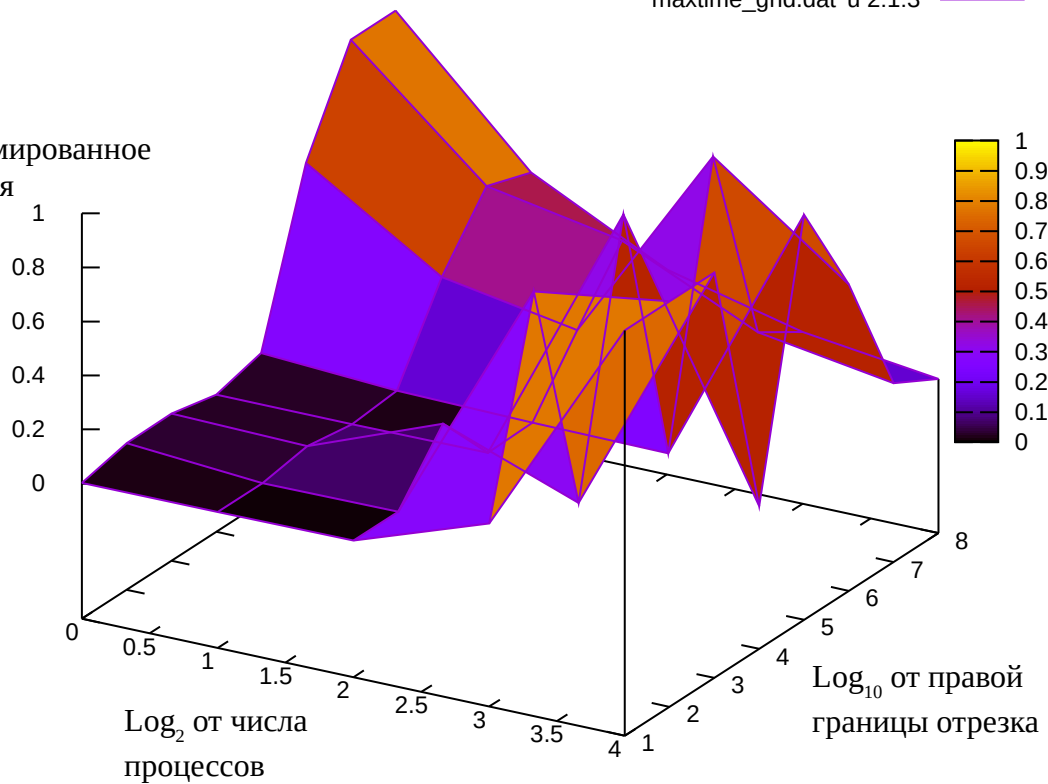
Нормированное
время

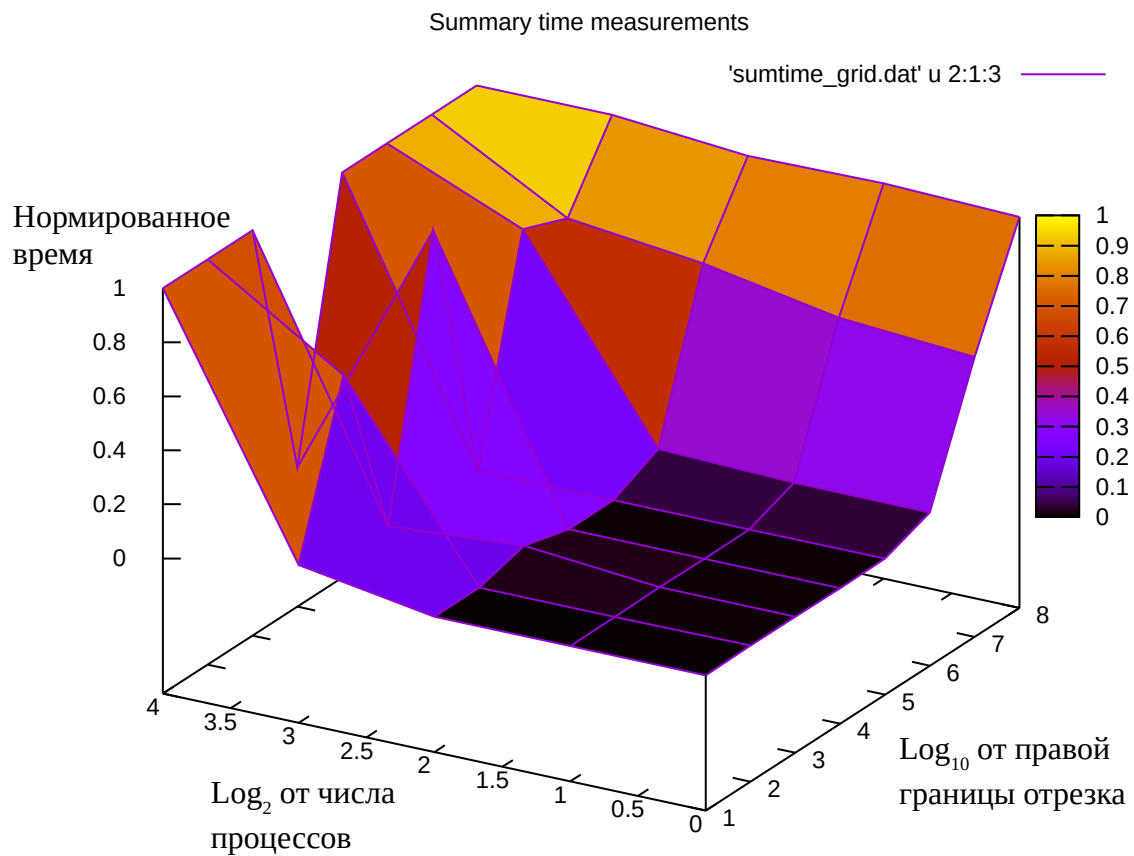


Maximum time measurements (from dif-t perspective)

'maxtime_grid.dat' u 2:1:3

Нормированное
время





Из графиков видно, что накладные расходы на создание 2-4 процессов практически нулевые, в то время как создание 8 или 16 процессов требует значительно большего, к тому же более случайного, времени. Затраты на создание такого большого числа процессов оправдываются только при $N = 10^7 - 10^8$, где суммарное время работы всех процессов близко ко времени работы программы с одним процессом, а максимальное меньше в несколько раз.