Московский Государственный Университет им. М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Суперкомпьютеров и Квантовой Информатики

Спецкурс: системы и средства параллельного программирования.

Отчёт № 3.
Анализ параллельной оптимизации решета Эратосфена при помощи MPI

Работу выполнил
Федоров В. В.

Москва 2020

Постановка задачи и формат данных.

Задача: Реализовать параллельный параллельную версию решета Эратосфена — алгоритма поиска простых чисел на отрезке [lo,hi] — при помощи технологий MPI и pthread, 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. Для замера времени в pthread-версии программы использовалась функция gettimeofday из библиотеки sys/time.h.
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, $N \in \{10^6, 10^7, 10^8, 10^9\}$, 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.

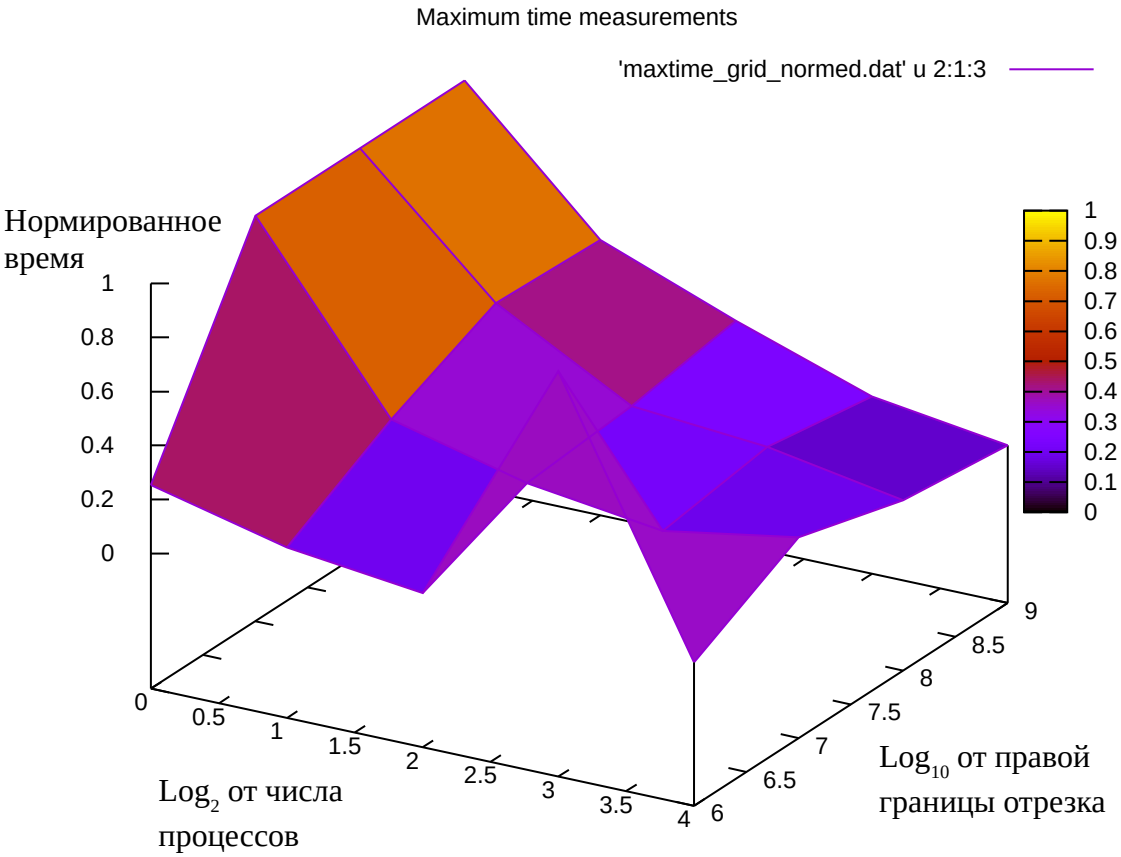
Максимальное время работы, с (MPI)					
	1	**2**	**4**	**8**	**16**
10^6	0.024987	0.012908	0.006933	0.098559	0.002973
10^7	0.371855	0.132001	0.084267	0.059127	0.090179
10^8	2.841460	1.520100	0.748009	0.623084	0.369550
10^9	43.136600	22.376200	14.156100	6.755560	3.599000

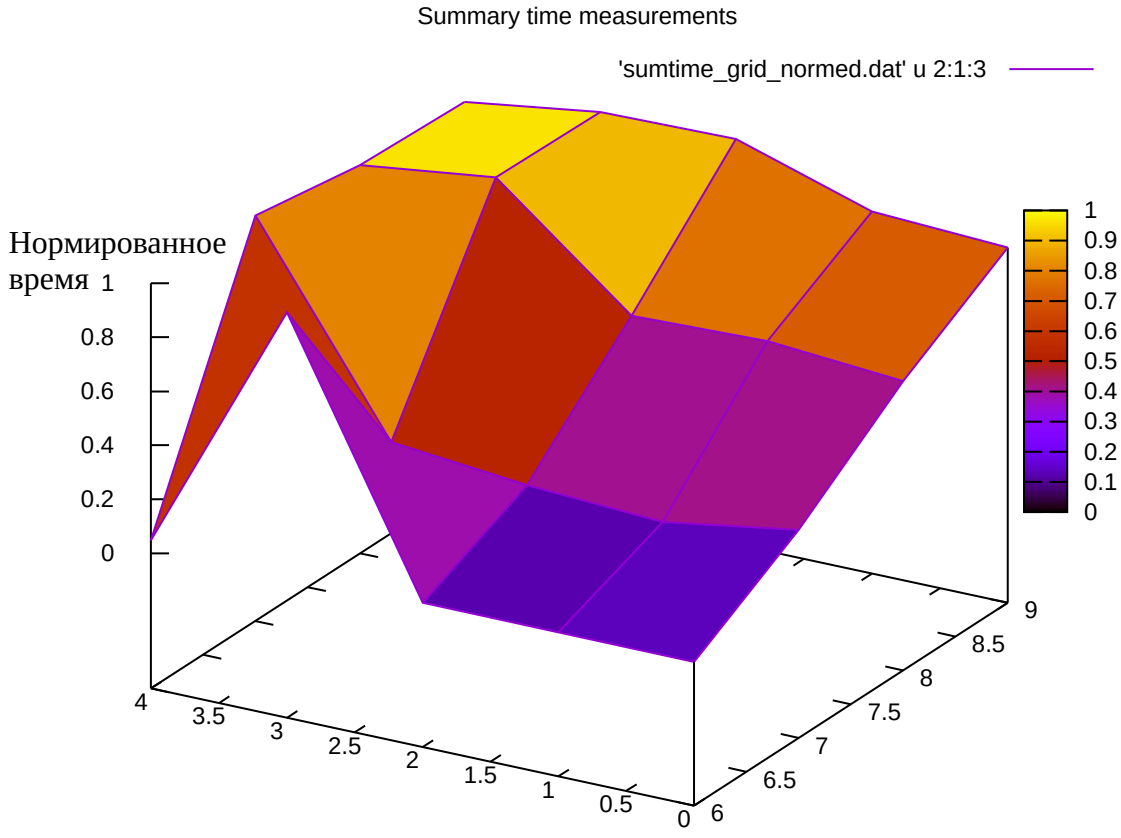
Суммарное время работы, с (MPI)					
	1	**2**	**4**	**8**	**16**
10^6	0.024987	0.025783	0.026899	0.780947	0.03784
10^7	0.371855	0.263553	0.302692	0.371947	1.37861
10^8	2.841460	3.037370	2.961880	4.972250	4.66118
10^9	43.136600	44.427100	52.906700	52.466800	48.67470

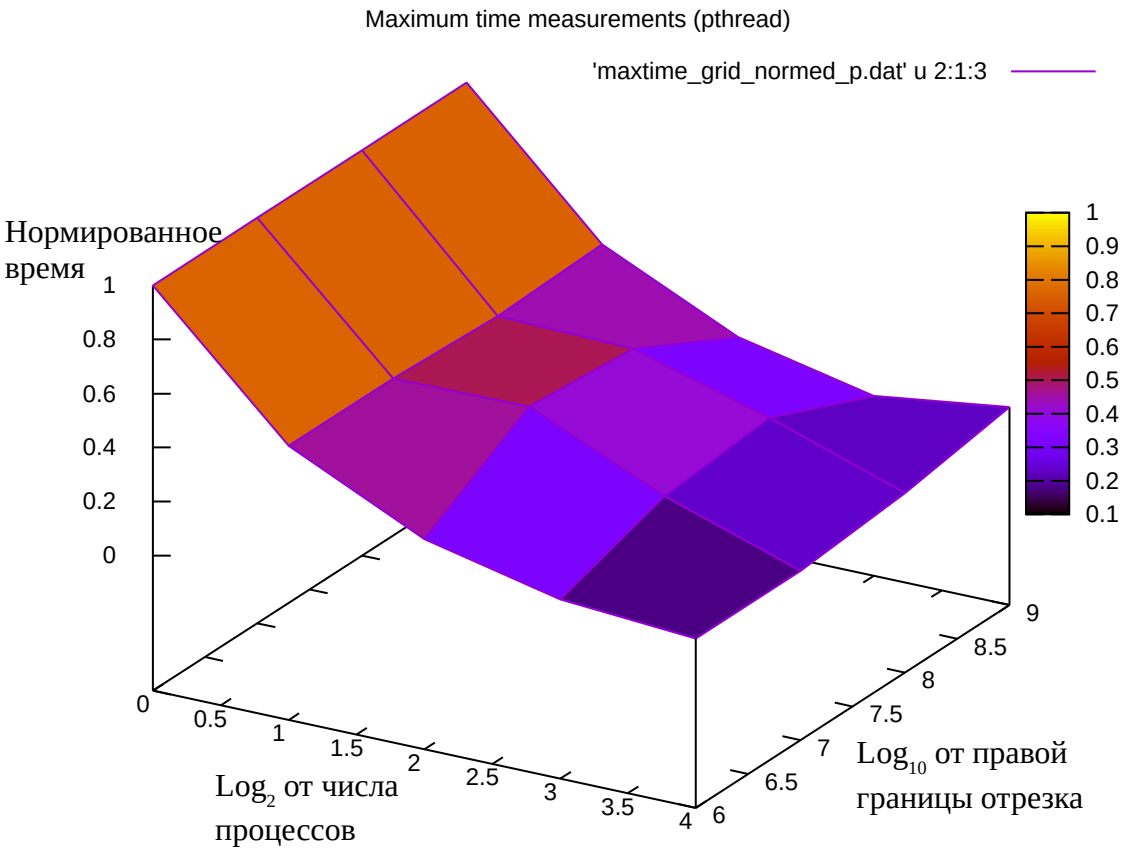
Максимальное время работы, с (pthread)					
	1	**2**	**4**	**8**	**16**
10^6	0.024473	0.012601	0.006786	0.003975	0.003092
10^7	0.251150	0.129045	0.130259	0.073961	0.031859
10^8	3.063050	1.514120	1.478950	1.029260	0.503605
10^9	43.743600	22.300600	12.090800	7.278540	10.211300

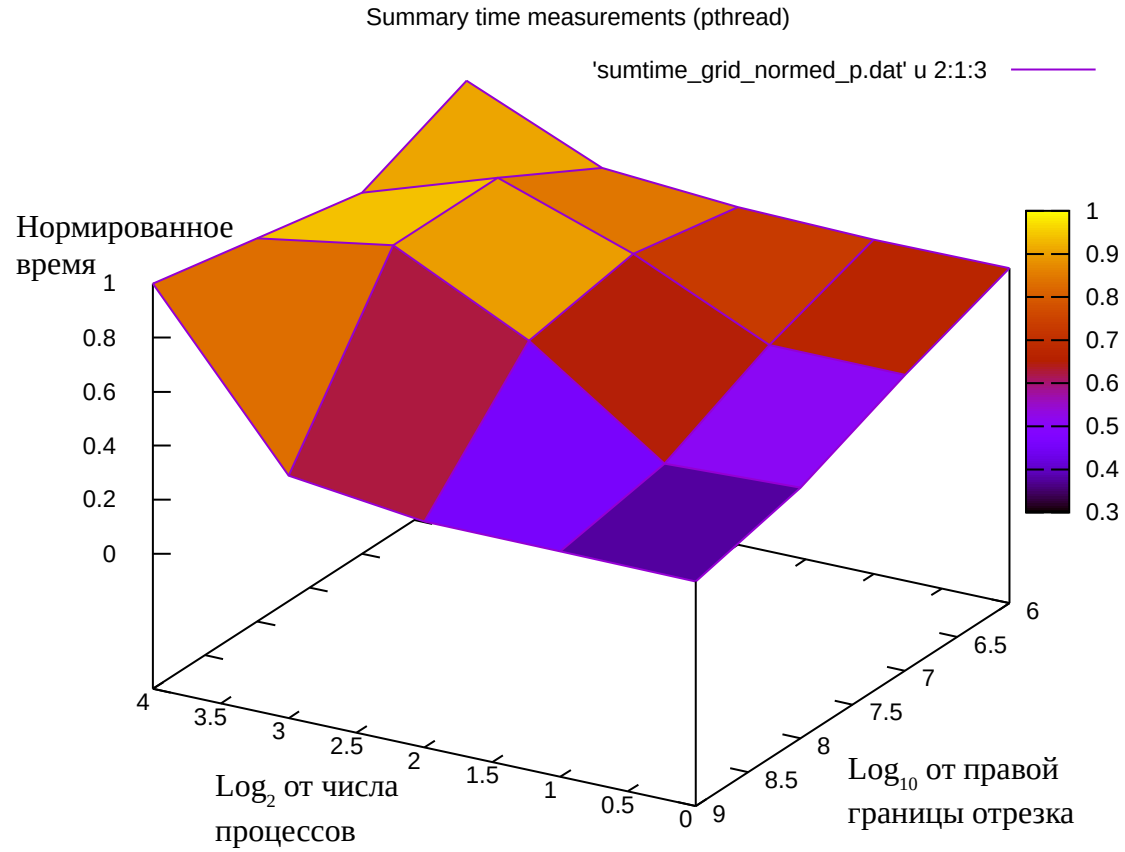
Суммарное время работы, с (pthread)					
	1	**2**	**4**	**8**	**16**
10^6	0.024473	0.024403	0.024792	0.026017	0.033075
10^7	0.251150	0.252044	0.348756	0.421692	0.352647
10^8	3.063050	2.927950	5.411200	7.168590	6.577020
10^9	43.743600	44.116000	44.334300	52.558900	132.316000

Основные выводы.






Maximum time measurements (pthread)
'maxtime_grid_normed_p.dat' u 2:1:3
Нормированное время
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5
4
Log2 от числа процессов
6
6.5
7
7.5
8
8.5
9
Log10 от правой границы отрезка
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Summary time measurements (pthread)
'sumtime_grid_normed_p.dat' u 2:1:3
Нормированное время
1
0.8
0.6
0.4
0.2
0
4
3.5
3
2.5
2
1.5
1
0.5
0
Log2 от числа процессов
9
8.5
8
7.5
7
6.5
6
Log10 от правой границы отрезка
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3

Для MPI, ожидаемо, увеличение числа процессов привело к значительному ускорению программы: для $N=10^9$ для 2, 4, 8 и 16 процессов время уменьшилось примерно в 2, 3, 6 и 12 раз соотвественно. Для pthread результаты получились схожими.