Отчёт по 5-му заданию спецкурса "Параллельное программирование для высокопроизводительных вычислительных систем".

Подготовил: студент 428 группы Манушин Дмитрий Валерьевич.

1. Описание задачи.

В задании требовалось разработать параллельную программу с использованием технологий MPI и OpenMP, реализующую решение задачи Дирихле методом SOR. Провести исследование эффективности разработанной программы на системе BlueGene/P. Провести визуализацию полученного решения.

2. Описание решения.

В ходе выполнения задания разработана параллельная программа, решающая задачу Дирихле методом SOR.

При запуске на n процессах алгоритм решения задачи Дирихле состоит в следующем.

1) Разбиваем область решения равномерной сеткой на заданное количество точек.

2) Проводим 1D декомпозицию — разбиваем сетку на n горизонтальных лент в соответствии с количеством процессов.

3) В каждом процессе будем хранить матрицу, количество строк и столбцов которой соответствует ленте + по одному дополнительному столбцу и одной дополнительной строки с каждой стороны. Дополнительные столбцы используются для левого и правого краевых условий. Верхняя строка в 0 процессе — для верхнего краевого условия, нижняя строка в n-ном процессе — для нижнего краевого условия. Остальные дополнительные строки для синхронизации решений после очередной итерации.

4) Далее проводится итерации до достижения заданной точности:

a) Проводится расчет значений во всей ленте, кроме дополнительных строк и столбцов, по методу SOR с использованием текущего и 4 соседних значений. При расчете внутри одного процесса выполняется распараллеливание циклов потоками OpenMP.

b) Проводится обмен значениями первой и последними строками с верхним (i-1) и нижним (i+1) процессами с помощью функции MPI_Sendrecv. При этом полученные значения записываются в дополнительные строки. Таким образом, достигается синхронизация решения во всей области, как если бы расчетами занимался один процесс.

Точность определяется как корень из суммы квадратов разностей текущего и предыдущего значений. Для определения точности по всей области выполняется функция MPI_Reduce.

5) Каждый процесс сохраняет свою часть матрицы области решения в файл.

Описанный алгоритм реализуется в функции main файла main.c.

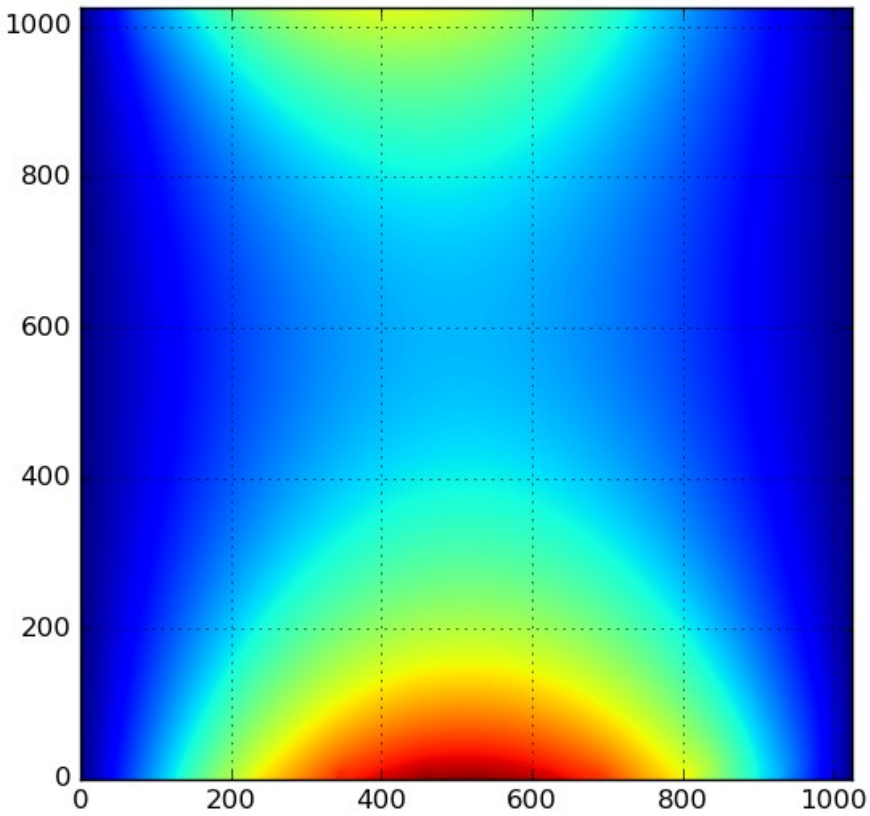
Также при выполнении данного задания в соответствии с требованиями задачи изменены программы и скрипты, описанные в предыдущем отчёте. Запустить вычисления можно выполнив программу main, принимающую 3 аргумента: число точек для разбиения по одному разбиению, точность, имя файла для сохранения результата.

Время и количество итераций выводится на стандартный поток вывода и анализируется скриптом show.py.

Визуализация выполняется с помощью библиотеки matplotlib на Python в скрипте plot.py, выполняющем визуализацию файла result с матрицей результата.

3. Результаты.

Визуализация результата:

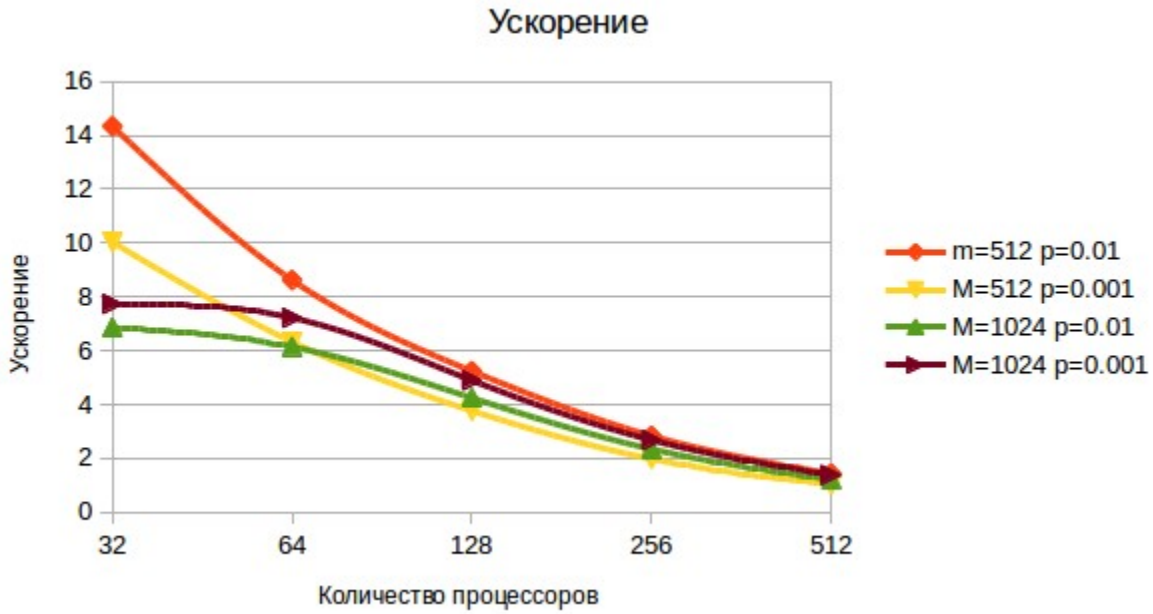


В ходе выполнения задания было замерено время выполнения алгоритма для разных параметров задачи и разных количеств процессоров. Данные приведены в таблице.

Размер сетки	Точность	Количество процессоров	Время	Ускорение	Количество итераций
512x512	0.01	32	6.246168	14.33305380	3000
512x512	0.01	64	10.387034	8.619078555	3000

512x512	0.01	128	17.09635	5.236595062	3000
512x512	0.01	256	31.613857	2.831880399	3000
512x512	0.01	512	62.683778	1.428226969	3000
512x512	0.01	512 произвольный мэппинг	> 300		
512x512	0.001	32	8.927284	10.02843216	4000
512x512	0.001	64	14.159597	6.322684325	4000
512x512	0.001	128	23.774355	3.765682055	4000
512x512	0.001	256	45.265474	1.977813421	4000
512x512	0.001	512	87.490211	1.023276329	4000
512x512	0.001	512 произвольный мэппинг	>300		
1024x1024	0.01	32	29.261708	6.852869354	6000
1024x1024	0.01	64	32.661785	6.139488763	6000
1024x1024	0.01	128	47.253693	4.243618842	6000
1024x1024	0.01	256	85.654449	2.341112041	6000
1024x1024	0.01	512	166.474129	1.204551501	6000
1024x1024	0.01	512 произвольный мэппинг	>300		
1024x1024	0.001	32	45.313257	7.735631583	10000
1024x1024	0.001	64	48.563028	7.217973763	10000
1024x1024	0.001	128	71.623589	4.894011412	10000
1024x1024	0.001	256	130.983197	2.676119304	10000
1024x1024	0.001	512	253.87174	1.380723439	10000
1024x1024	0.001	512 произвольный мэппинг	>300		

По данным таблицы были построены графики ускорения в зависимости от количества процессоров для различных параметров задачи.



4. Выводы.

По графикам можно заметить, что использование нескольких процессоров для умножения матриц позволяет значительно ускорить процесс умножения.

Однако, стоит заметить, что при увеличении количества процессоров больше 32, ускорение уменьшается. Это происходит из-за того, что нарушается баланс между вычислениями и коммуникациями — на коммуникации (особенно на MPI_Reduce) уходит значительно больше времени, чем на вычисления (тем более вычисления ускоряются с помощью потоков OpenMP).

Для разных параметров задачи графики ускорения практически не отличаются, что свидетельствует о стабильном (робастном) поведении параллельного алгоритма.

Таким образом, предложенный алгоритм позволяет довольно эффективно вычислять решение задачи Дирихле на высокопроизводительных вычислительных системах с распределённой памятью, какой является суперкомпьютер BlueGene/P.