МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по учебному курсу

"Суперкомпьютеры и параллельная обработка данных"

Задание №1:

Разработка параллельной версии программы Seidel-2d с использованием технологии MPI.

ОТЧЕТ

о выполненном задании

студента 328 учебной группы факультета ВМК МГУ

Фомина Сергея Александровича

Москва, 2019 г.

Оглавление

1 Постановка задачи

Имеется код алгоритма Seidel-2d, работающий с квадратной матрицей. Ставится задача разработки параллельной версии этого алгоритма.

Требуется:

1. Реализовать параллельную версию предложенного алгоритма с использованием технологий MPI.
2. Исследовать масштабируемость полученной параллельной программы: построить графики зависимости времени исполнения от числа процессов для различного объёма входных данных.

2 Описание последовательного алгоритма

Последовательный алгоритм имеет следующий вид:

```
static
void kernel_seidel_2d(int tsteps, int n, float A[n][n]){
   int t, i, j;

   for (t = 0; t <= tsteps - 1; t++)
      for (i = 1; i <= n - 2; i++)
         for (j = 1; j <= n - 2; j++)
            A[i][j] = (A[i - 1][j - 1] + A[i - 1][j] + A[i - 1][j + 1] +
                       A[i][j - 1] + A[i][j] + A[i][j + 1] +
                       A[i + 1][j - 1] + A[i + 1][j] + A[i + 1][j + 1]) / 9.0 f;
}
```

Этот алгоритм является итеративным и ориентирован на последовательное вычисление значений матрицы A. Каждое значение матрицы вычисляется как среднее значение 9 элементов, образующих квадрат, в центре с текущим элементом. Вычисления проводятся многократно во внешнем цикле. Предполагается выполнение $T \times (N-2) \times (N-2)$ операций поиска среднего значения, где N – размерность матрицы, T – количество повторений внешнего цикла.

3 Описание параллельного алгоритма

Ниже приведено описание параллельного алгоритма на MPI. Код программы можно найти в github-репозитории: https://github.com/TotalChest/Program-Parallelization

Прежде всего, стоит заметить, что в каждом вычислении среднего значения участвуют соседние элементы матрицы. Это говорит о цикле с зависимостью по данным. Простое распараллеливание с разделением матрицы на блоки может привести к неправильному результату. Здесь нужно учитывать порядок вычисления элементов массива. Можно заметить, что для вычисления очередного элемента используются четыре значения, посчитанные на текущей итерации внешнего цикла, и четыре из предыдущей итерации (на рисунке 1 : синие клетки – элементы этой итерации, серые - предыдущей).

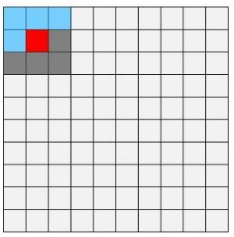


Рисунок 1. Зависимость по данным

В результате распараллеливания задачи с использованием технологии OpenMP было выяснено, что алгоритм распараллеливания внешнего цикла показывает наилучшие результаты и в целом более стабильный. Ниже описан принцип его работы с использованием технологии MPI. Реализованный алгоритм проверялся на корректность и совпадение по результатам с последовательным алгоритмом.

Алгоритм с распараллеливанием внешнего цикла основан на использовании нескольких процессов для одновременного вычисления нескольких итераций. Так как в задаче присутствует зависимость по данным, каждый процесс должен отставать от предыдущего на две строки матрицы, чтобы иметь возможность одновременно выполнять вычислений на разных итерациях. Каждый процесс при обработке очередной строки получает данные от предыдущего процесса (предыдущей итерация), обрабатывает эту строку и передает ее следующему процессу (для выполнения следующей итерации). Синхронизация процессов осуществляется за счет блокирующих операций MPI_Send и MPI_Recv. Результат выполнения последней итерации передается первому процессу и из него можно получить итоговый результат работы алгоритма.

Код параллельного алгоритма:

```
Static void kernel_seidel_2d(int tsteps, int n, float A[ n][n])
{
    int i, j, k, process_per_iteration;
    MPI_Comm_size(MPI_COMM_WORLD, &ranksize);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Barrier(MPI_COMM_WORLD);

    int iterations = (tsteps%ranksize==0)?(int)(tsteps/ranksize):(int)(tsteps/ranksize+1);

    for (k = 0; k < iterations; k++){
        process_per_iteration = (k + 1 == iterations) ? ((tsteps % ranksize == 0) ? \
                                ranksize : tsteps % ranksize) : ranksize;
        if (id == 0 && k != 0)
            MPI_Recv(&A[0][0], n * n, MPI_FLOAT, ranksize-1, 1216, MPI_COMM_WORLD, &status);

        for (i = (-2) * id + 1; i <= n - 2 + 2 * (process_per_iteration - 1 - id); i++){

            if (id != 0 && i >= 0 && i <= n - 3 && id < process_per_iteration)
                MPI_Recv(&A[i+1][1], n-2, MPI_FLOAT, id-1,1215,MPI_COMM_WORLD, &status);

             if (i >= 1 && i <= n - 2 && id < process_per_iteration){

                 for (j = 1; j <= n - 2; j++)
                     A[i][j] = (A[i-1][j-1] + A[i-1][j] + A[i-1][j+1] + \
                                A[i][j-1] + A[i][j] + A[i][j+1] + \
                                A[i+1][j-1] + A[i+1][j] + A[i+1][j+1])/9.0f;

                 if (id != process_per_iteration - 1)
                     MPI_Send(&A[i][1], n - 2, MPI_FLOAT, id + 1, 1215, MPI_COMM_WORLD);
            }
        }

        if (id == ranksize - 1 && k + 1 != iterations)
            MPI_Send(&A[0][0], n * n, MPI_FLOAT, 0, 1216, MPI_COMM_WORLD);
    }

    if (id == 0)
       MPI_Recv(&A[0][0],n*n,MPI_FLOAT,process_per_iteration-1,1217,MPI_COMM_WORLD, &status);
    if (id == process_per_iteration - 1)
       MPI_Send(&A[0][0], n * n, MPI_FLOAT, 0, 1217, MPI_COMM_WORLD);
}
```

4 Результаты замеров времени выполнения

Ниже приведены результаты замеров времени работы программ на суперкомпьютерах Bluegene и Polus: непосредственно в табличной форме и наглядно на 3D-графиках.

Программа была запущена в конфигурациях:

- на Polus - 2, 4, 8, 16, 32 процессов;
- на Bluegene - 2, 4, 8, 16, 32, 64, 96, 128, 256, 512 процессов.

Измерения проводились на различных объемах входных данных.

Название датасета	Итерации внешнего цикла	Размерность матрицы
MINI_DATASET	20	40
SMALL_DATASET	40	120
MEDIUM_DATASET	100	400
LARGE_DATASET	500	2000
EXTRALARGE_DATASET	1000	4000

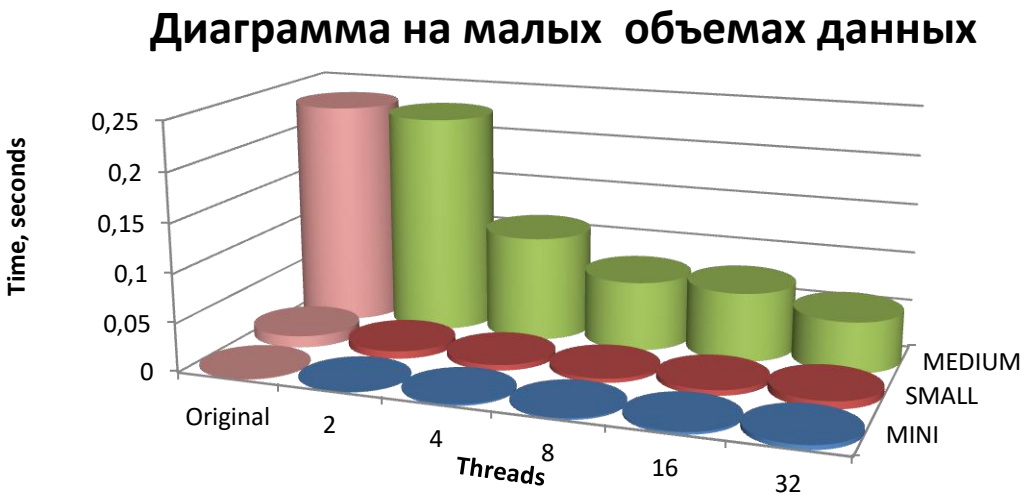
Таблица 1. Данные

Каждая конфигурация была запущена по 5 раз. Для чистоты эксперимента отбрасывались самый большой и самый маленький результат, затем три оставшиеся результата усреднялись.

4.1 Результаты на Polus

Threads	MINI_DATASET	SMALL_DATASET	MEDIUM_DATASET	LARGE_DATASET	EXTRALARGE_DATASET
2	0.000572	0.007811	0.224473	27.051537	190.775093
4	0.001655	0.005696	0.107464	13.189665	128.463952
8	0.001577	0.004219	0.071090	7.920258	72.598072
16	0.002384	0.005569	0.070521	4.064644	47.981660
32	0.004532	0.007851	0.052369	2.679097	29.228115
Original	0.000582	0.011902	0.230371	44.638135	358.371057

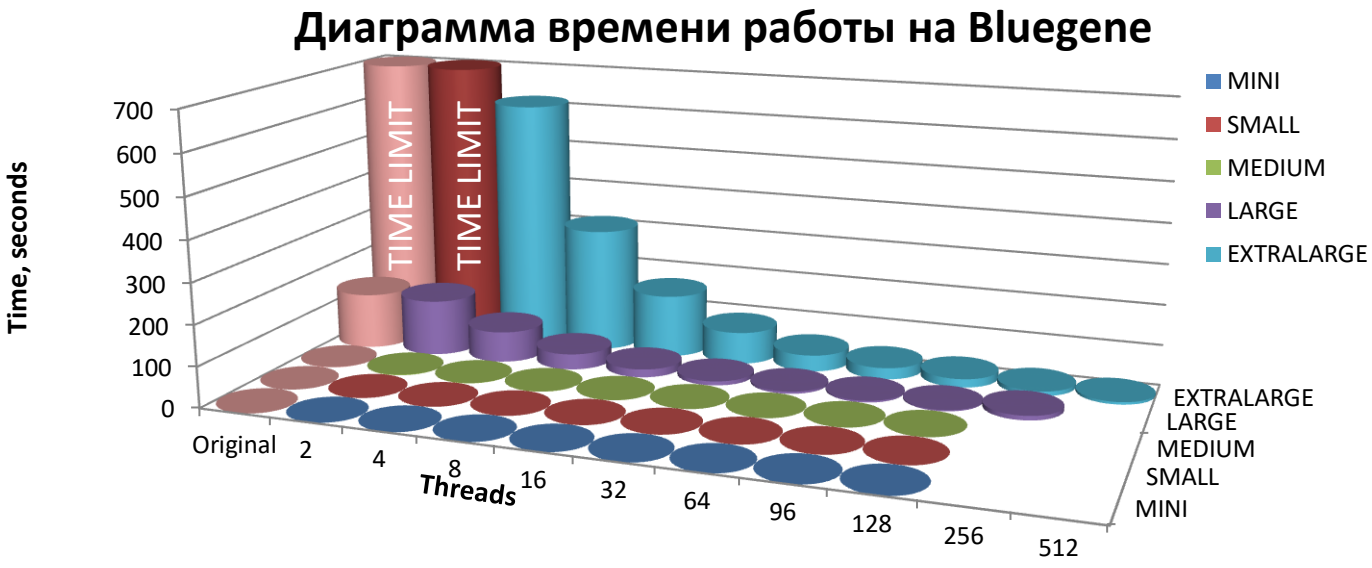
Таблица 2. Время работы на Polus на разных объемах данных

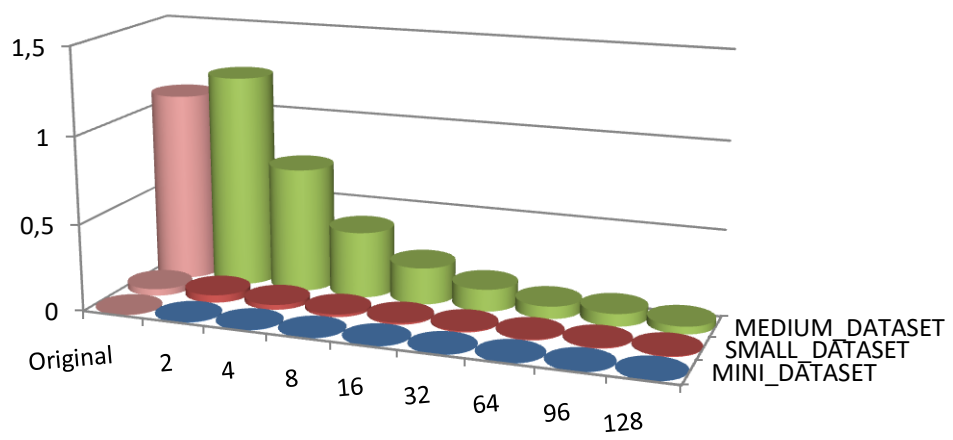


4.2 Результаты на Bluegene

Threads	MINI_DATASET	SMALL_DATASET	MEDIUM_DATASET	LARGE_DATASET	EXTRALARGE_DATASET
2	0.003079	0.043310	1.224332	138.22283	TL
4	0.002405	0.028106	0.717514	76.196473	614.863817
8	0.001657	0.014968	0.382424	38.532127	307.375129
16	0.001300	0.009743	0.212415	19.718673	155.455255
32	0.000955	0.007231	0.127720	10.002459	79.498390
64	0.000957	0.004940	0.071562	5.139755	40.294811
96	0.000961	0.004842	0.071874	3.924698	28.044373
128	0.000956	0.004935	0.043655	2.709144	20.694350
256				10.90447	10.904337
512					6.0036703
Original	0.002013	0.038545	1.09322	137.726650	TL

Таблица 3. Время работы на Bluegene на разных объемах данных





5 Анализ результатов

На основе полученных результатов видно, что задача прекрасно поддалась распараллеливанию и зависимость скорости работы программы от числа нитей близка к линейной.

Следует заметить, что результаты запуска программы на машине Polus не впечатляют, на малых объемах данных показатели крайне нестабильны. Вероятно, это связанно со структурой вычислительного ресурса. Однако, при масштабных вычислениях, качественный подбор параметров суперкомпьютера дает ощутимый прирост производительности.

Результаты работы на суперкомпьютере Bluegene показали, что система хорошо работает с многопроцессными программами. Из графиков видно, что время вычислений на машине Bluegene перестает уменьшаться при достижении некоторого предельного значения процессов. Это связанно с тем, что при увеличении количества процессов накладные расходы на передачу сообщений превышают выгоду от их использования.

6 Выводы

Выполнена работа по разработке параллельной версии алгоритма Seidel-2d. Изучена технология написания параллельных алгоритмов MPI. Проанализировано время выполнения алгоритмов на различных вычислительных системах.

Технология MPI крайне удобна в использовании, причем дает колоссальный прирост производительности на системах, рассчитанных на многопроцессные вычисления. Распараллеливание программы дало выигрыш по времени в 100 раз на больших объемах данных.