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 OpenMP.

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 OpenMP.
2. Исследовать масштабируемость полученной параллельной программы: построить графики зависимости времени исполнения от числа нитей 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 приведены описания и обоснования параллельных алгоритмов на OpenMP. Коды программ 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 использование одной прагмы 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).

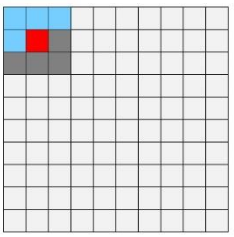


Рисунок 1. Зависимость по данным

В результате исследования задачи были разработаны два подхода к решению. Первый состоит в распараллеливании с задержкой внешнего цикла (подробнее описан ниже). Второй с использованием алгоритма гиперплоскостей (со смещение на две ячейки). Также был написан код, объединяющий в себе оба алгоритма, но из-за больших накладных расходов он не показал хороших результатов. После тестирования

алгоритмов на разных машинах, с разными объемами данных было выяснено, что первый алгоритм (распараллеливание внешнего цикла) показывает наилучшие результаты и в целом более стабильный. Далее рассматривается только он. Реализованные алгоритмы проверялись на корректность и совпадение по результатам с последовательным алгоритмом.

Алгоритм с распараллеливанием внешнего цикла основан на использовании нескольких потоков для одновременного вычисления нескольких итераций. Так как в задаче присутствует зависимость по данным, каждый поток должен отставать от предыдущего на две строки матрицы, чтобы случайным образом не изменить данные, используемые более ранней итерацией. Задержка каждой нити осуществляется путем вычитания из итерируемой переменной номера текущего потока. Чтобы контролировать последовательность обрабатываемых строк используется прагма barrier.

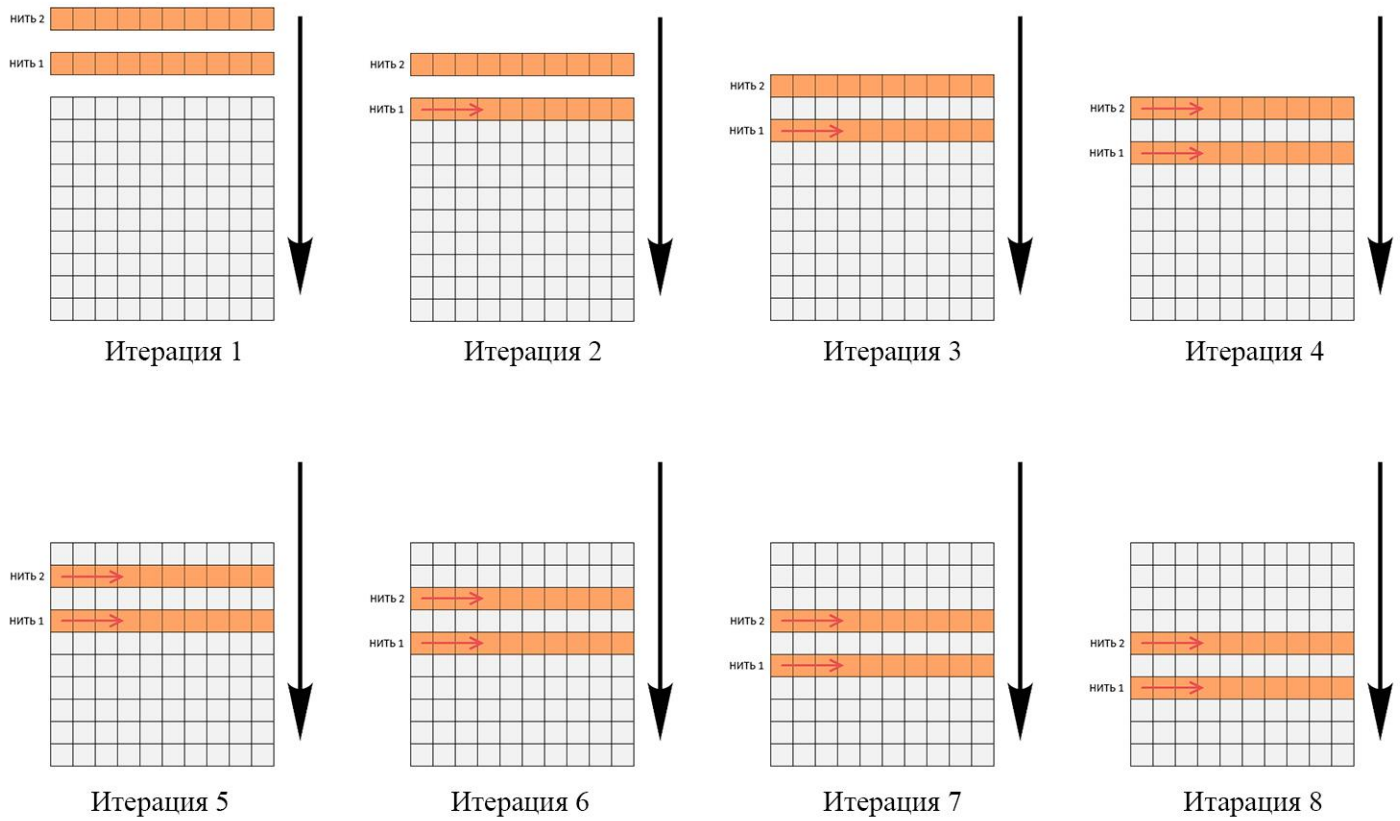


Рисунок 2. Пример работы параллельного алгоритма

На рисунке 2 приведен пример работы одной итерации алгоритма для матрицы размерности 10 и двумя нитями. Каждая нить полностью проходит весь массив сверху вниз, обрабатывая его построчно. Ускорение достигается тем, что следующая итерация внешнего цикла не ждет завершения предыдущей, а начинает обрабатывать массив параллельно с отставание в две строки.

Код параллельного алгоритма:

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

    int OMP_NUM_THREADS = atoi(getenv("OMP_NUM_THREADS"));
    int numt = (tsteps > OMP_NUM_THREADS) ? OMP_NUM_THREADS : tsteps;
    omp_set_num_threads(numt);
    printf("OMP threads: %d\n", numt);

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

    #pragma omp parallel private(i, j)
    {
        int k;
        int id = omp_get_thread_num();
        for (k = 0; k < iterations; k++) {
            threads_per_iteration = (k + 1 == iterations) ? \
                                    ((tsteps % numt == 0) ? numt : tsteps % numt): numt;
            for (i = (-2) * id + 1; i <= n - 2 + 2 * (numt - 1 - id); i++) {
                if (i >= 1 && i <= n - 2 && id < threads_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;
                #pragma omp barrier
            }
        }
    }
}
```

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 - 1, 2, 4, 8, 16, 32, 64, 96, 112, 128 потоков;
- на Bluegene - 1, 2, 4 потоков.

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

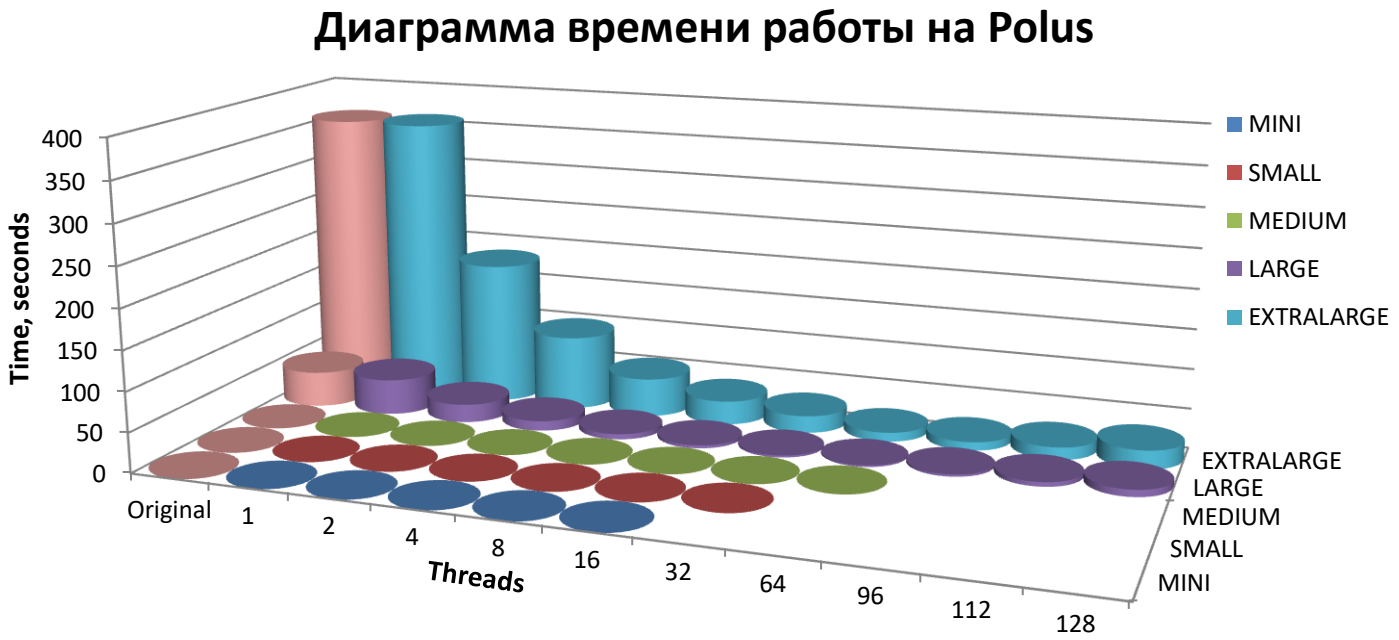
Таблица 1. Данные

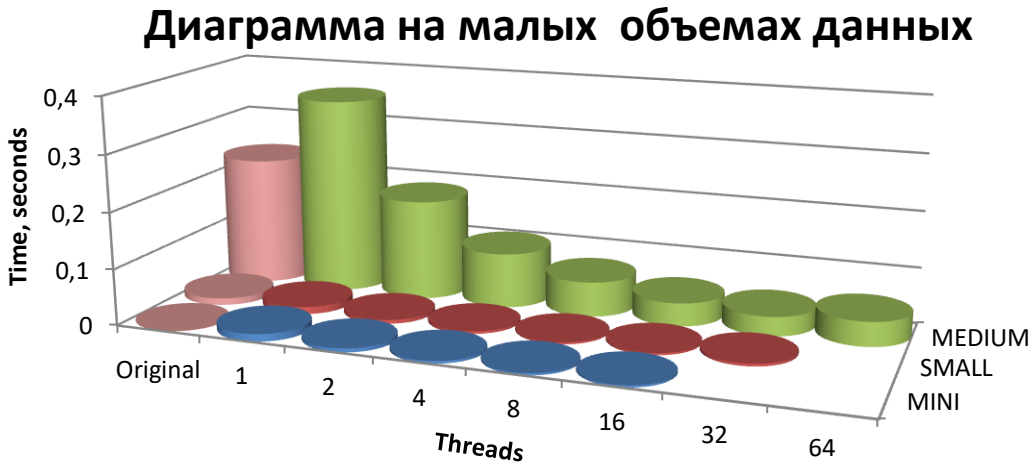
Каждая конфигурация была запущена по 5 раз. Для чистоты эксперимента отбрасывались самый большой и самый маленький результат, затем три оставшиеся результата усреднялись.

4.1 Результаты на Polus

Threads	MINI_DATASET	SMALL_DATASET	MEDIUM_DATASET	LARGE_DATASET	EXTRALARGE_DATASET
1	0.013246	0.013191	0.351007	44.802964	358.371057
2	0.006953	0.006936	0.179140	22.922163	179.907436
4	0.004486	0.004515	0.097597	12.303377	93.667910
8	0.003553	0.003314	0.061944	6.666930	48.55104
16	0.003465	0.003531	0.041190	3.722205	29.841152
32		0.004620	0.034461	2.516182	21.567350
64			0.043710	1.941274	11.423715
96				2.706941	10.593664
112				5.656241	15.593664
128				8.823547	23.504251
Original	0.000582	0.011902	0.230371	44.638135	358.371057

Таблица 2. Время работы на Polus на разных объемах данных

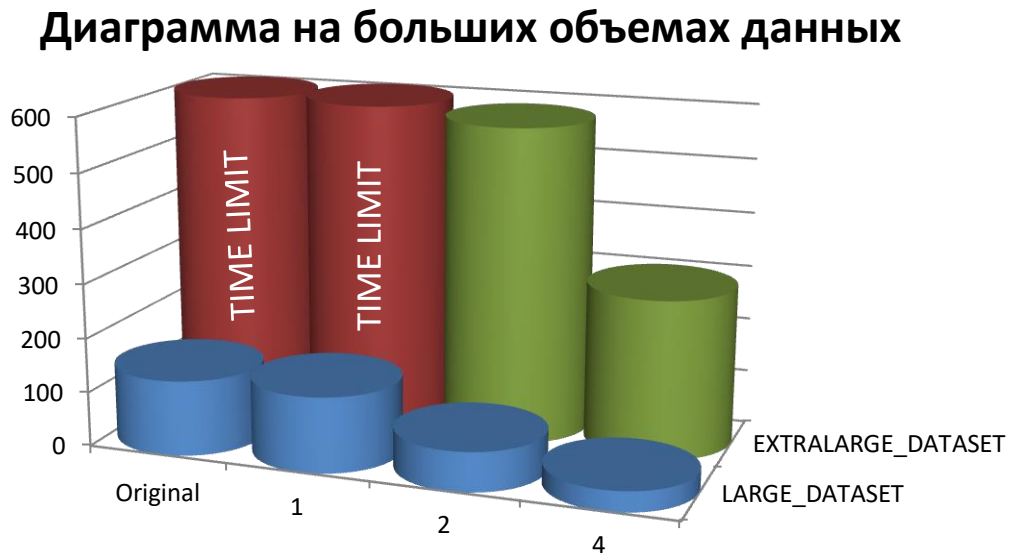




4.2 Результаты на Bluegene

Threads	MINI_DATASET	SMALL_DATASET	MEDIUM_DATASET	LARGE_DATASET	EXTRALARGE_DATASET
1	0.009506	0.045957	1.102304	137.953675	TL
2	0.010910	0.053840	0.633599	71.918874	577. 397616
4	0.010603	0.026460	0.346804	36.975848	290.638288
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 Bluegene не впечатляют, особенно на малых объемах данных. Это связанно с тем, что система заточена не под многопоточные вычисления, а под многопроцессорные.

Из графиков видно, что время вычислений на машине Polus перестает уменьшаться при увеличении числа нитей до 64. Это связанно с тем, что при увеличении количества нитей накладные расходы на создание и взаимодействия потоков превышают выгоду от их использования. Также эффект от накладных расходов заметен при запуске параллельного алгоритма на одной нити. Заметен проигрыш работы по сравнению с последовательным алгоритмом (Original).

6 Выводы

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

Технология OpenMP крайне удобна в использовании, причем дает колоссальный прирост производительности на системах, рассчитанных на многопоточные вычисления. Распараллеливание программы дало выигрыш по времени в 30 раз на больших объемах данных.