МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по курсу

"Суперкомпьютеры и параллельная обработка данных"

Разработка параллельной версии программы Sor_3D с использованием технологии OpenMP.

ОТЧЕТ

о выполненном задании

студента 324 учебной группы факультета ВМК МГУ

Яндутова Алексея Владимровича

Москва, 2019 г.

Оглавление

1 Постановка задачи

Дана последовательная программа Sor_3D, реализующая алгоритм релаксации Якоби. Распараллеливание осуществляется с помощью анализа последовательной программы, аналогично анализу распараллеливающего компилятора, поэтому не предполагается знания указанного алгоритма.

Требуется:

1. Реализовать параллельную версию программы для задачи Sor_3D с использованием технологии OpenMP.
2. Исследовать масштабируемость полученных программ и построить графики зависимости времени выполнения программ от числа используемых ядер и объёма входных данных.

2 Реализация

2.1 Исходный код программы

Функции исходной программы:

- `init()` инициализирует трехмерную матрицу
- `verify()` высчитывает контрольную сумму итоговой матрицы
- `relax()` выполняет функционал алгоритма релаксации Якоби

Работа была проведена только с функцией `relax()`, так как она является узким местом программы и требует оптимизации и распараллеливания.

```
void relax()
{

        for(k=1; k<=N-2; k++)
        for(j=1; j<=N-2; j++)
        for(i=1; i<=N-2; i++)
        {
                double e;
                e=A[i][j][k];
                A[i][j][k]=(A[i-1][j][k]+A[i+1][j][k]+A[i][j-
                1][k]+A[i][j+1][k]+A[i][j][k-1]+A[i][j][k+1])/6.;
                eps=Max(eps, fabs(e-A[i][j][k]));
        }
}
```

Этот алгоритм является итеративным, на каждой итерации вычисляется элемент матрицы как среднее ее 6 соседей. Сложность алгоритма O(n**3).

2.2 Параллельный алгоритм

Ниже приведены краткие заметки по реализации параллельной программы на OpenMP. Коды программ можно найти в github-репозитории: https://github.com/iden-alex/msu_supercomputers_2019.

```
void relax()
{
#pragma omp parallel private(nk) shared(A)
    {
        for(nk=3; nk<=3*N-6; ++nk) {
#pragma omp for schedule(static) private(i, j, k) reduction(max: eps)
                for(i=Max(1, nk-2*N+4); i<=Min(N-2, nk-2); ++i) {
                for(j=Max(1, nk-i-N+2); j<=Min(N-2, nk-i-1); ++j) {
                        double e;
                        k = nk - j -i;
                        e=A[i][j][k];
                        A[i][j][k]=(A[i-1][j][k]+A[i+1][j][k]+A[i][j-1]
[k]+A[i][j+1][k]+A[i][j][k-1]+A[i][j][k+1])/6.;
                        eps=Max(eps, fabs(e-A[i][j][k]));
                }
                }
        }
    }
}
```

Были внесены следующие изменения:

1) Изменены местами внешний и внутренний цикл, чтобы происходило обращение к последовательным участкам памяти.

2) Функция в начальном виде имеет цикл с зависимостью по данным, поэтому в исходном виде данный цикл нельзя распараллеливать. Для устранения зависимости по данным был использован метод гиперплоскостей – все пространство итераций разбивается на гиперплоскости размерности 2, таким образом, что все операции, соответствующие точкам одной гиперплоскости, могут выполняться параллельно. Полученные гиперплоскости перпендикулярны вектору (1,1,1), обход матрицы в данном случае начинается с элемента (0,0,0). При помощи функции `verify()` проверено , что преобразованный цикл получает тот же результат, что и исходная функция.

Использованные директивы OpenMP:

- `#pragma omp for` для распределения веток цикла `for` по разным нитям
- `private(i, j, k)` для того, чтобы каждая нить имела свою собственную копию глобальных переменных `i, j, k` для обхода циклов
- `shared(A)` для верной работы с массивом A, который остаётся в общем доступе для всех нитей
- `reduction(max: eps)` для верной работы с переменной `eps`, которая остаётся глобальной и меняется внутри каждой нити

4) Преобразование цикла привело к тому, что в распараллеленном цикле на каждой итерации выполняется различное количество операций(так как границы внутренних циклов зависят от внешних), следствием чего

является дисбаланс нагрузки на нитях. Для решения данной проблемы была применена клауза `schedule(static)`.

3 Результаты замеров времени выполнения

Ниже приведены результаты замеров времени программы на суперкомпьютере Polus: непосредственно в табличной форме и наглядно на 3D-графиках.

Программа была запущена на Polus на 1, 2, 4, 8, 12, 16, 18, 20 процессорах на различных размерах матрицы.

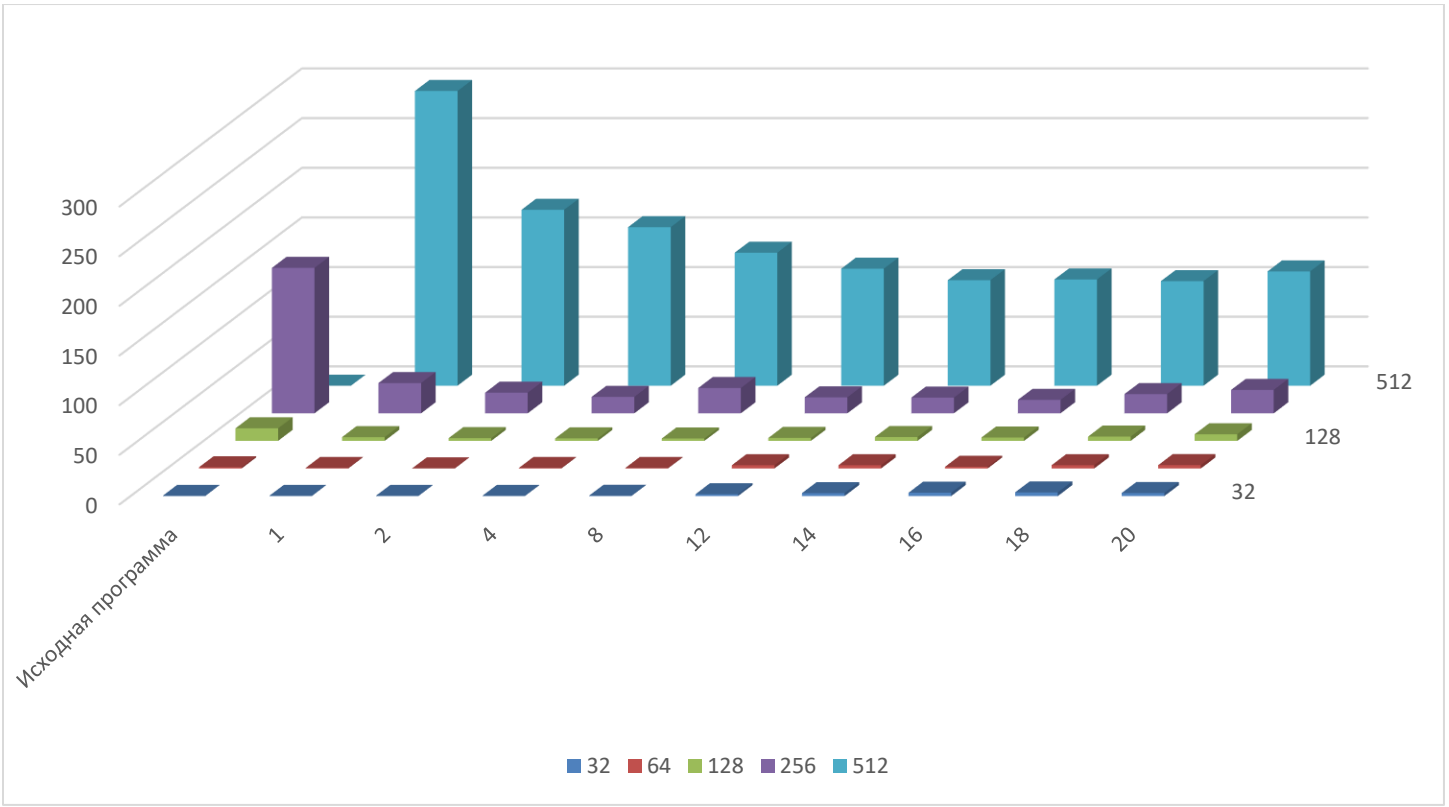
Каждая запуск был произведен 3 раза, ниже приведена таблица с усредненными значениями времени.

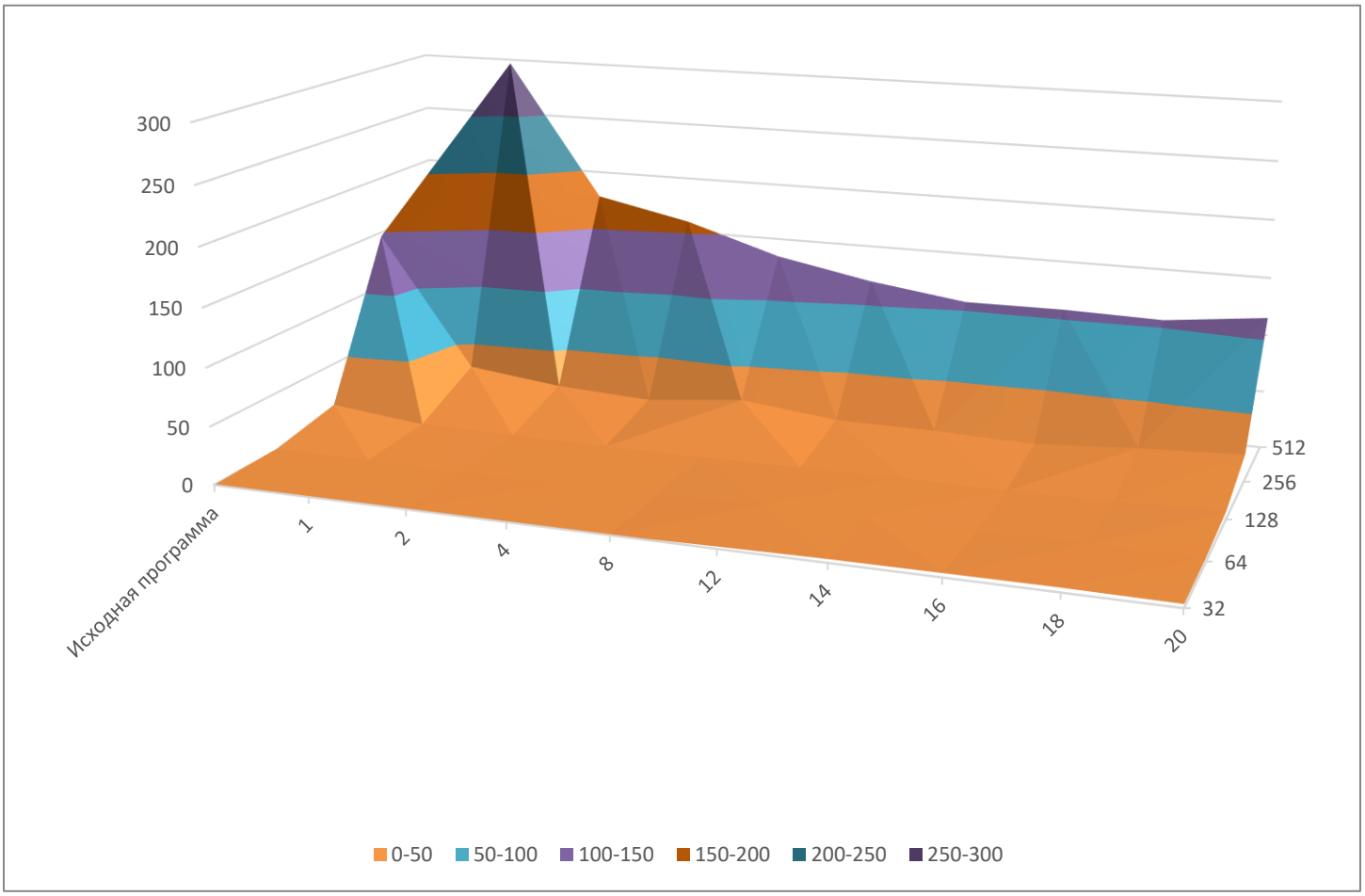
3.1 Таблица

Значения времени выполнения в таблице приведены в секундах

Количество ядер/ **Размер матрицы**	**32*32*32**	**64*64*64**	**128*128*128**	**256*256*256**	**512*512*512**
Исходная программа	0,15	1,24	12,67	146,42	>TL
1	0,08	0,47	3,79	30,44	296,69
2	0,09	0,1	2,73	20,72	177,13
4	0,12	0,43	2,48	16,48	159,61
8	0,22	0,52	2,41	25,42	133,87
12	1,81	3,27	2,96	16,1	117,81
14	2,96	3,25	3,88	15,81	106,23
16	3,34	1,65	3,42	13,5	106,97
18	3,68	3,3	4,3	19,34	105,31
20	3,05	3,34	6,67	23,49	115,08

3.2 3D графики


300
250
200
150
100
50
0
Исходная программа
1
2
4
8
12
14
16
18
20
32
128
512
32
64
128
256
512


300
250
200
150
100
50
0
Исходная программа
1
2
4
8
12
14
16
18
20
32
64
128
256
512
0-50
50-100
100-150
150-200
200-250
250-300

Замечание к графику: на размере 512*512*512 время выполнения исходной программы превысило Time limit = 15 min.

4 Анализ результатов

Параллельная версия программы увеличила скорость выполнения исходного алгоритма в среднем в 7,1 раза.

На маленьких размерах матрицы программа дала наибольшую производительность на 1-2 процессах.

На матрицах больших размеров заметен рост скорости выполнения с ростом процессов, наименьшее время достигается на 16-18 процессорах, однако на 20 процессорах время выполнения начинает расти. Увеличение времени выполнения программы с ростом количества используемых ядер объясняется накладными расходами при синхронизации нитей.

Можно заметить спад темпа роста производительности с увеличением нитей на матрицах большего размера, что связано с тем, что распараллеленный цикл имеет форму, отличную от параллелепипеда(так как границы внутренних циклов меняются в зависимости от внешних). Также можно ожидать, что на матрицах еще большего размера данная параллельная версия будет неэффективна, так как элементы гиперплоскостей больших матриц сильно разнесены друг от друга в памяти, что приводит к неэффективному использованию кэша. Данная проблема может быть решена повышением локальности данных за счет разбиения пространства итераций на трехмерные фигуры и применением метода гиперплоскостей к каждому из них.

5 Выводы

Выполнена работа по разработке параллельной версии алгоритма релаксации Якоби. Изучены технологии написания параллельных алгоритмов OpenMP. Проанализировано время выполнения алгоритмов на различных количествах процессоров.