МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по курсу

"Суперкомпьютеры и параллельная обработка данных"

Разработка параллельной версии программы с использованием OpenMP

ОТЧЕТ

студента 321 учебной группы факультета ВМК МГУ

Сенюшкиной Алины Константиновны

Москва, 2023 г.

Оглавление

1 Цель работы

1. Разработка параллельной версии программы с использованием технологии OpenMP.
2. Исследовать масштабируемость полученной параллельной программы, построив графики зависимости времени выполнения от числа используемых ядер для различного объема входных данных.
3. Определить основные причины недостаточной масштабируемости программы при максимальном числе используемых ядер/процессоров.

2 Алгоритм

2.1 Базовый алгоритм

- На вход подается матрица размера N×N
- В функции init() происходит инициализация матрицы изначально заданным способом
- В функции relax() происходит основная часть работы алгоритма – релаксация исходной матрицы
- В функции verify() происходит финальное суммирование для вычисления ответа

2.2 Параллельная реализация с помощью директивы for

В функции relax() заданного алгоритма вычисление значения к матрице `A[i][j]`

- в первом вложенном цикле вычисляется по двум ближайшим соседям в столбце:

```
A[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i - 2][j] + A[i + 2][j]) / 4.;
```

- во втором вложенном цикле вычисляется по двум ближайшим соседям в строке:

```
A[i][j] = (A[i][j - 1] + A[i][j + 1] + A[i][j - 2] + A[i][j + 2]) / 4.;
```

Таким образом, в первом вложенном цикле вычисления выполняются независимо в цикле по j, а во втором по i. Для вычисления максимума среди нитей используется секция `critical`, так как в ней может находиться одновременно не более одной нити.

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
#define  Max(a,b) ((a)>(b)?(a):(b))

float maxeps = 0.1e-7;
int itmax = 100;
int i,j,k;

float eps;
float A [N][N];

void relax();
void init();
void verify();

int main(int argc, char **argv)
{
   omp_set_num_threads((int)strtol(argv[1], NULL, 10));
   double avg_time = 0.;

   int it;
   init();
   double start = omp_get_wtime();

   for (it = 1; it <= itmax; it++) {
       eps = 0;
       relax();
       if (!(it % 1000)) {
           printf("it=%4i   eps=%.6f\n", it, eps);
       }
       if (eps < maxeps) break;
   }
```

```
    double finish = omp_get_wtime();
    avg_time += finish - start;
    verify();
    printf("%lf\n", avg_time);
    return 0;
}

void init()
{
    for(i=0; i<=N-1; i++) {
        for (j = 0; j <= N - 1; j++) {
            if (i == 0 || i == N - 1 || j == 0 || j == N - 1) A[i][j] = 0.;
            else A[i][j] = (1. + i + j);
        }
    }
}

void relax()
{
#pragma omp parallel shared(A, eps) private(i, j) default(none)
    {
        float e;

        for (i = 2; i <= N - 3; i++) {
#pragma omp for schedule(static)
            for (j = 1; j <= N - 2; j++) {
                A[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i - 2][j] + A[i + 2][j]) / 4.;
            }
        }
#pragma omp barrier

#pragma omp for schedule(static)
        for (i = 1; i <= N - 2; i++) {
            for (j = 2; j <= N - 3; j++) {
                float tmp = A[i][j];
                A[i][j] = (A[i][j - 1] + A[i][j + 1] + A[i][j - 2] + A[i][j + 2]) / 4.;
                e = Max(e, fabs(tmp - A[i][j]));
            }
        }
#pragma omp critical
        eps = Max(eps, e);
    }
}

void verify()
{
    float s;
```

```
    s=0.;
    for (i=0; i<=N-1; i++) {
    for (j=0; j<=N-1; j++) {
        s = s + A[i][j] * (i + 1) * (j + 1) / (N * N);
    }
    }
    printf("\tS = %f\n", s);
}
```

2.3 Параллельная реализация с помощью директивы task

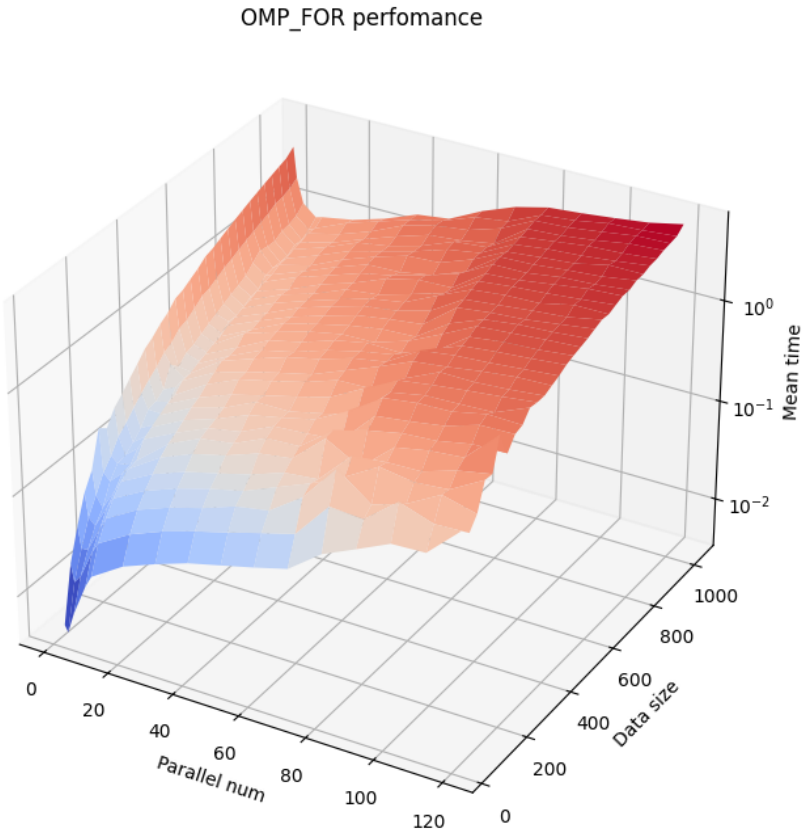
Нить попавшая в блок `single` создает пул задач, из которого остальные нити берут задачи на выполнение. Время выполнения существенно увеличивается с ростом количества задач, поэтому количество задач было уменьшено с помощью переменной `per_task`, которая равна количеству итераций в одном taske.

Далее приведен код только функции relax(), так как остальной код идентичен предыдущему.

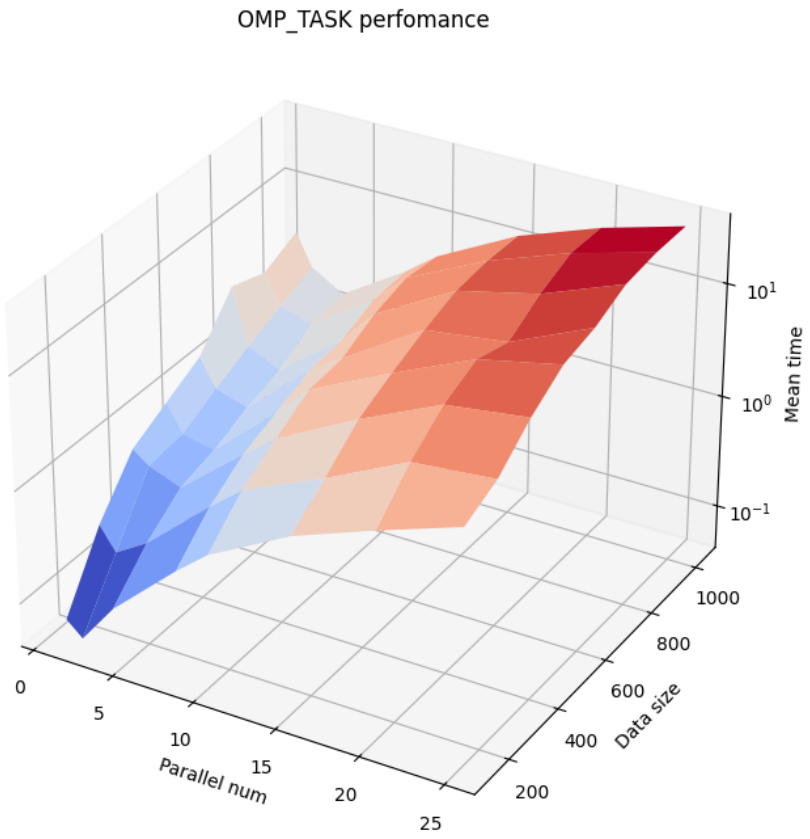
```
void relax() {
#pragma omp parallel shared(A, eps, num_of_threads) private(i, j, k) default(none)
    {
        float e;
#pragma omp single
        {
            int per_task = (N - 2) / num_of_threads;
            int num_of_tasks = (N - 2) / per_task;
            if ((N - 2) % per_task != 0) {
                num_of_tasks += 1;
            }
            for (i = 2; i <= N - 3; i++) {
                for (k = 0; k < num_of_tasks; k++) {
#pragma omp task
                    {
                        for (j = k * per_task + 1; j <= Min((k + 1) * per_task, N - 2); j++) {
                            A[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i - 2][j] + A[i + 2][j]) / 4.;
                        }
                    }
                }
#pragma omp taskwait
            }

            for (k = 0; k < num_of_tasks; k++) {
#pragma omp task private(e)
                {
                    for (i = k * per_task + 1; i <= Min((k + 1) * per_task, N - 2); i++) {
                        for (j = 2; j <= N - 3; j++) {
                            float tmp = A[i][j];
                            A[i][j] = (A[i][j - 1] + A[i][j + 1] + A[i][j - 2] + A[i][j + 2]) / 4.;
                            e = Max(e, fabs(tmp - A[i][j]));
                        }
```

```
                    }
#pragma omp critical
                    eps = Max(eps, e);
            }}}}}
```

3 Графики



директива for OpenMP



директива task OpenMP

4 Описание результатов

Удалось успешно написать две версии программы на OpenMP с использованием директивы for и task. По полученным графикам можно сделать вывод, что за счет распараллеливания программа ускоряется до числа потоков = 8 и далее замедляется из-за накладных расходов связанных с выделением большего числа нитей независимо от объема входных данных.