Теория Параллелизма

Отчет

“Уравнение теплопроводности”

Тихонова Виктория Андреевна

Группа 21933

08.03.2023

Цель работы

Реализовать решение уравнение теплопроводности (пятиточечный шаблон) в двумерной
области на равномерных сетках. Научиться пользоваться профилировщиком. Произвести оптимизацию предоставленного кода. Произвести сравнения по времени работы между CPU и GPU.

Используемый компилятор

Для запуска на CPU использовался: gcc -o main task2.c -lm
Для запуска на GPU использовался: pgcc -Minfo=all -fast -acc task2.c -o cparracc

Используемый профилировщик

Профилировщик nsys

Замер времени работы

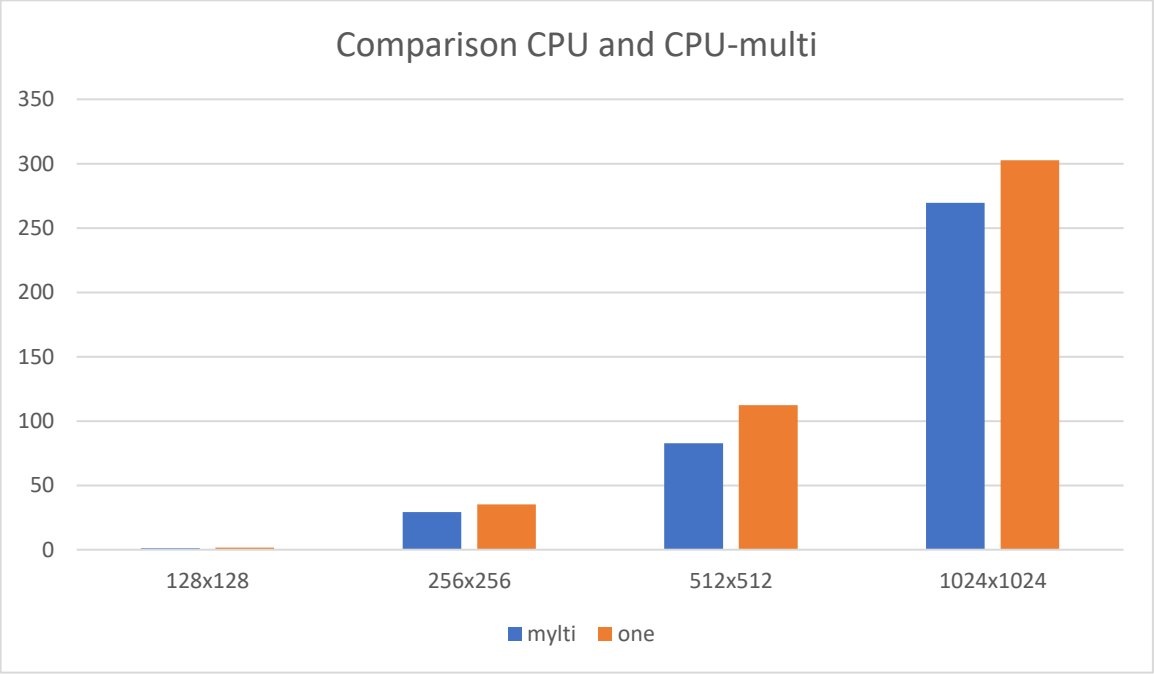
С помощью библиотеки #include <time.h>

(ссылка на гитхаб - https://github.com/ViktoriaTix/TP/blob/master/task2.c)

Таблица результатов выполнения программы

CPU			
Размер сетки	Время выполнения (сек)	Точность	Количество итераций
128x128	≈1.59	0.1E-5	8556
256x256	≈35.2	0.1E-5	89620
512x512	≈112.2	0.1E-5	256189
1024x1024	≈302.625	0.1E-5	1000000

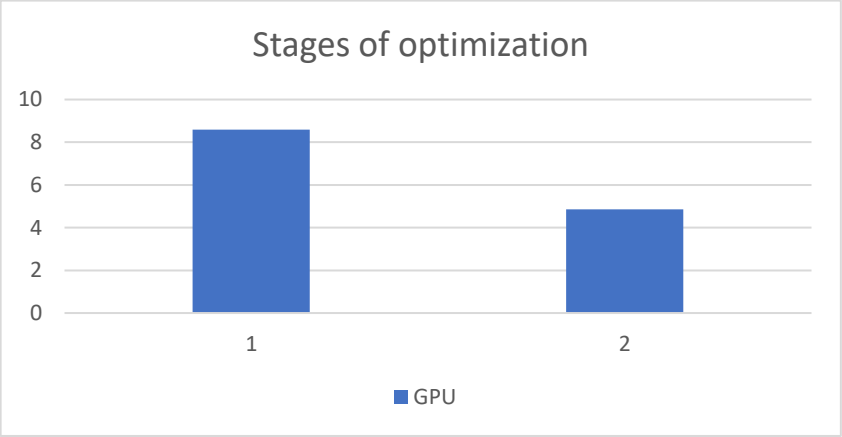
CPU - MULTICORE			
Размер сетки	Время выполнения (сек)	Точность	Количество итераций
128x128	≈1.021	0.1E-5	20009
256x256	≈29.35	0.1E-5	86452
512x512	≈82.789	0.1E-5	183246
1024x1024	≈269.6	0.1E-5	1000000



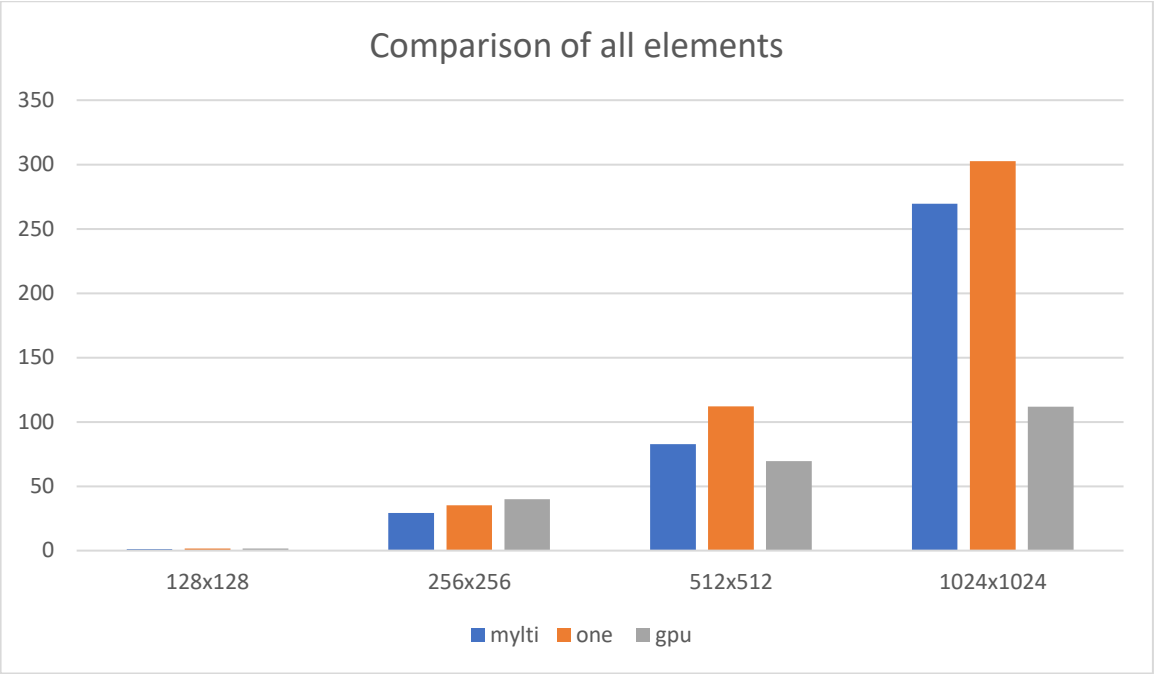
Выполнение на GPU

Этапы оптимизации GPU на сетке 512*512 при профилировании 100

№	Время выполнения (сек)	Точность	Количество итераций	Комментарий
1	8.59	1.49E-3	100	Без использования #pragma
2	4.85	1.49E-3	100	С использованием #pragma



GPU – оптимизированный вариант			
Размер ячейки	Время выполнения (сек)	Точность	Количество итераций
128x128	≈1.582	0.1E-5	25024
256x256	≈39.91	0.1E-5	58521
512x512	≈69.483	0.1E-5	225678
1024x1024	≈111.897	0.1E-5	1000000



Вывод:

Опираясь на диаграммы, сравнивающие время работы программы на разных процессорах, можно сделать вывод, что в большинстве случаев по времени работы выигрывает GPU. Но при этом для сеток небольшой размерности разница не велика, следовательно на таких примерах можно использовать любой процессор.

Код:

```
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <stdio.h>


void heat_equation_2d(int n, double k, int max_iters, double tol, int* iters, double* err)
{
   // Allocate memory for the grid with ghost cells
   double* u = (double*)malloc((n + 2) * (n + 2) * sizeof(double));
   // Initialize the grid with the boundary conditions
   u[0*(n+2)+0] = 10;
   u[0*(n+2)+(n+1)] = 20;
   u[(n+1)*(n+2)+0] = 30;
   u[(n+1)*(n+2)+(n+1)] = 20;
   for (int i = 1; i <= n; i++) {
      u[0*(n+2)+i] = 10 + i * (20 - 10) / (double)(n + 1);
      u[(n+1)*(n+2)+i] = 20 + i * (30 - 20) / (double)(n + 1);
      u[i*(n+2)+0] = 10 + i * (30 - 10) / (double)(n + 1);
      u[i*(n+2)+(n+1)] = 30 + i * (20 - 30) / (double)(n + 1);
   }
   // Calculate the grid spacing
   double h = 1 / (double)(n + 1);
   // Initialize the error and iteration counter
   double err_max = INFINITY;
   *iters = 0;
   // Perform the iterative updates
   while (err_max > tol && *iters < max_iters) {
      double* u_new = (double*)malloc((n + 2) * (n + 2) * sizeof(double));
#pragma acc parallel loop
      for (int i = 1; i <= n; i++) {
```

```
#pragma acc loop
            for (int j = 1; j <= n; j++) {
                double* u_ij = &u[i*(n+2)+j];
                double* u_new_ij = &u_new[i*(n+2)+j];
                double* u_im1j = &u[(i-1)*(n+2)+j];
                double* u_ip1j = &u[(i+1)*(n+2)+j];
                double* u_ijm1 = &u[i*(n+2)+j-1];
                double* u_ijp1 = &u[i*(n+2)+j+1];
                *u_new_ij = (1 / 4.0) * (*u_im1j + *u_ip1j + *u_ijm1 + *u_ijp1)
                    - (k * h * h / 4.0) * (*u_ijp1 + *u_ijm1 + *u_im1j + *u_ip1j - 4 * (*u_ij));
            }
        }
        // Calculate the maximum error
        err_max = 0.0;
#pragma acc parallel loop reduction(max:err_max)
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                 double* u_ij = &u[i*(n+2)+j];
                double* u_new_ij = &u_new[i*(n+2)+j];
                err_max = fmax(err_max, fabs(*u_new_ij - *u_ij));
            }
        }
        // Copy the updated grid to the old grid and free the memory for the new grid
#pragma acc parallel loop collapse(2)
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                u[i*(n+2)+j] = u_new[i*(n+2)+j];
            }
        }
        free(u_new);

        // Update the iteration counter and error
```

```
        *iters += 1;
        *err = err_max;
    }
    // Free the memory for the grid
    free(u);
}

int main(int argc, char** argv) {
    double time_spent1 = 0.0;
    clock_t begin1 = clock();
    // Parse command line arguments
    if (argc != 4) {
        printf("Usage: %s <accuracy> <grid size> <number of iterations>\n", argv[0]);
        return 1;
    }
    double tol = atof(argv[1]);
    int n = atoi(argv[2]);
    int max_iters = atoi(argv[3]);
    // Perform the heat equation computation
    int iters;
    double err;
    heat_equation_2d(n, 0.1, max_iters, tol, &iters, &err);
    // Print the results
    clock_t end1 = clock();
    time_spent1 += (double)(end1 - begin1) / CLOCKS_PER_SEC;
    printf("Iterations: %d\n", iters);
    printf("Error: %.10f\n", err);
    printf("%f\n", time_spent1);
    return 0;
}
```