Теория параллелизма

Отчет

Уравнение теплопроводности

Выполнил: Царев Алексей Александрович,
группа 21931, 2 курс.

Дата: 04.03.2023

Цели работы

1. Написать код для решения уравнения теплопроводности методом сеток.
2. Ускорить код.
3. Научиться профилировать программу при помощи Nsight Systems (nsys)

Используемый компилятор - pgcc

Используемый профилировщик - nvprof

Как производили замер времени работы – PGI_ACC_TIME, time

Выполнение на CPU

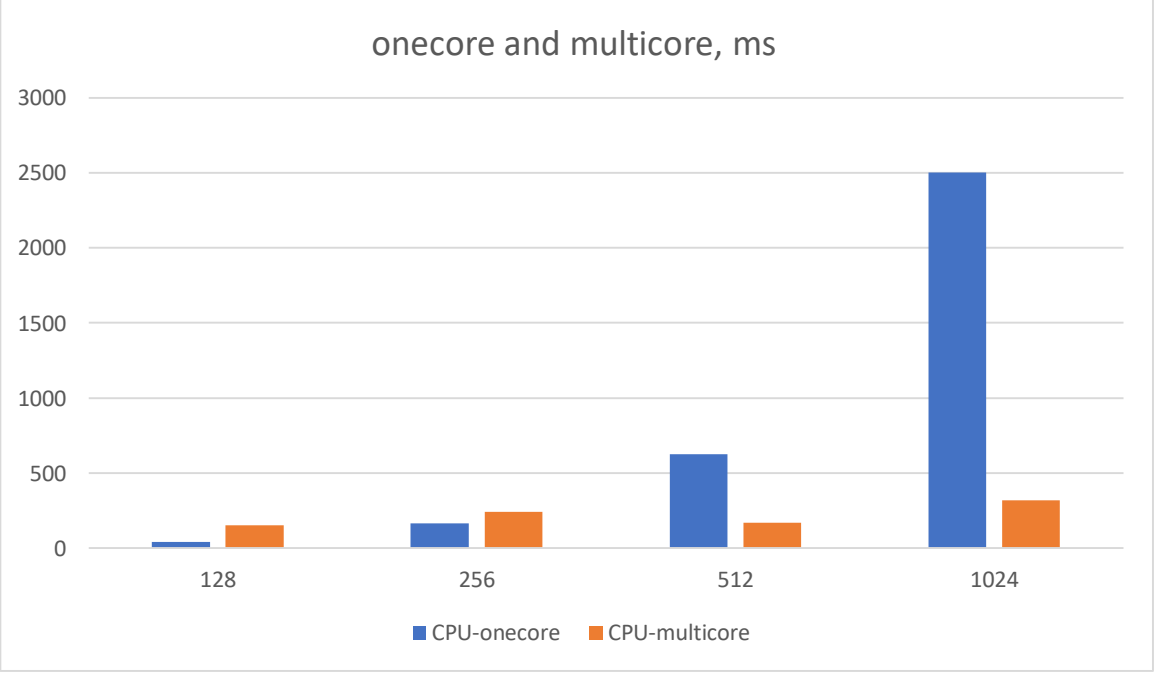
CPU-1

Размер сетки	Время выполнения, ms	Точность	Количество итераций
128*128	40.64	0.009801	1000
256*256	165	0.010302	1000
512*512	627	0.010553	1000
1024*1024	2501	0.010678	1000

CPU-multicore

Размер сетки	Время выполнения, ms	Точность	Количество итераций
128*128	154	0.009801	1000
256*256	242	0.010302	1000
512*512	170	0.010553	1000
1024*1024	320	0.010678	1000

Диаграмма сравнения время работы CPU-1 и CPU-multicore



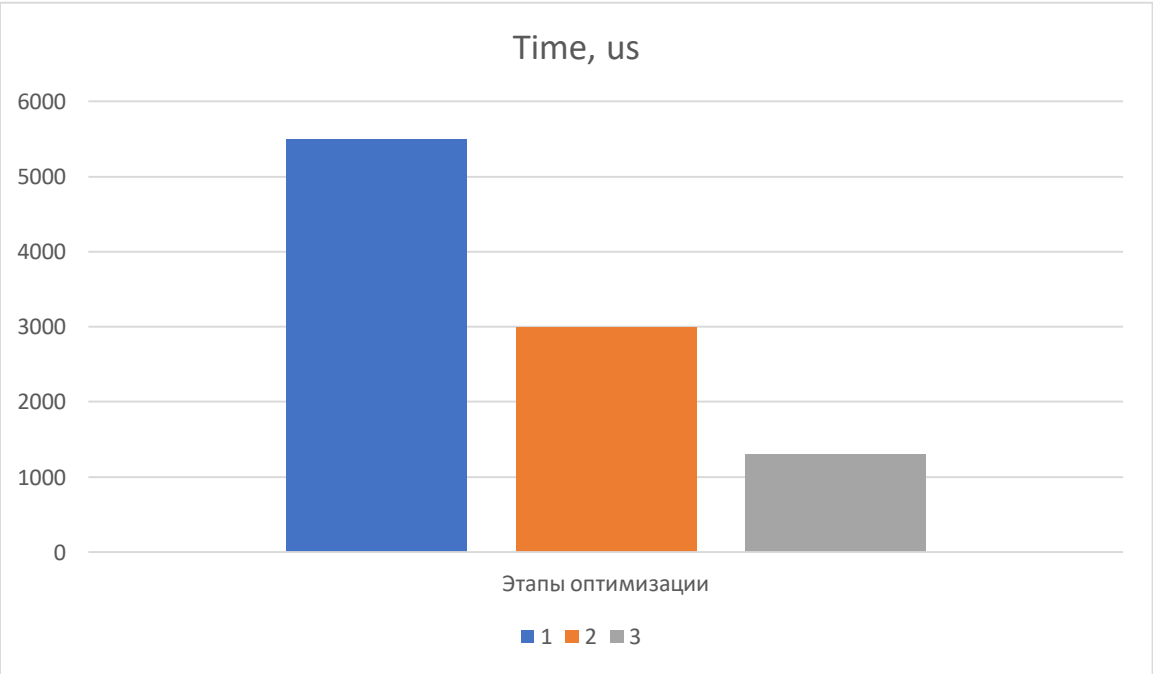
Выполнение на GPU

Этапы оптимизации на сетке 512*512

(количество итераций при профилировании 100)

Этап №	Время выполнения (суммарно),us	Точность	Количество итераций	Комментарии (что было сделано)
1	~ 5500	0.107043	100	
2	~ 3000	0.107043	100	Разделены вычисление средних и поиск макс. ошибки (error).
3	~1300	0.107043	100	Хранение данных в одном массиве размера (size * size) + collapse(2) при вычислении средних.

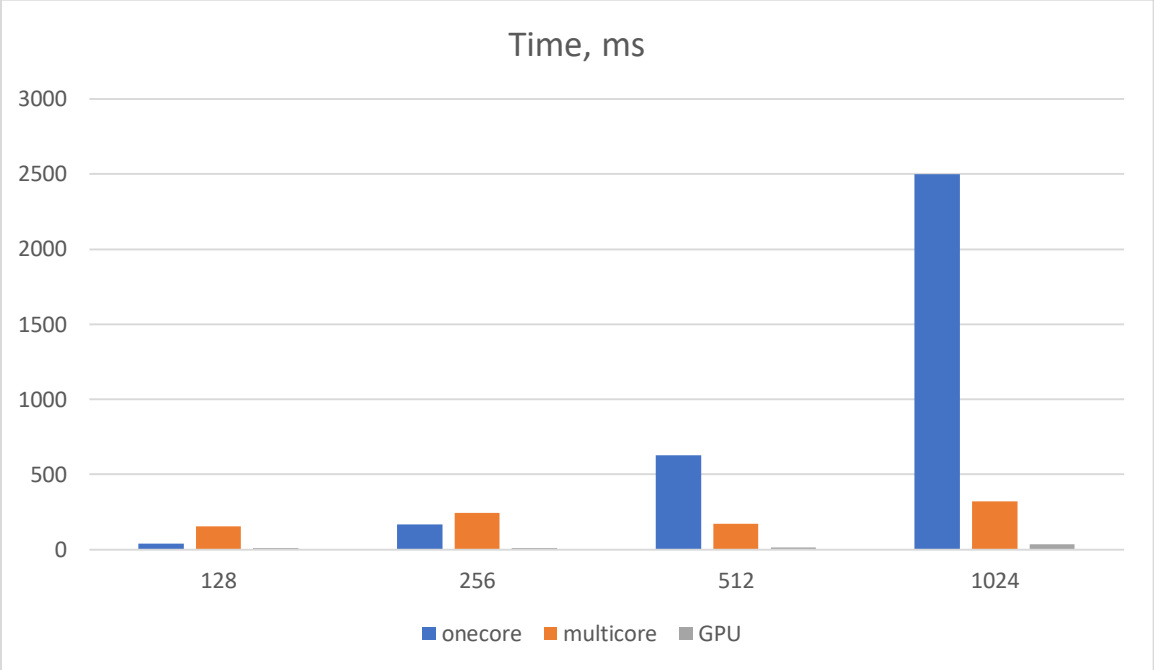
Диаграмма оптимизации (по горизонтали номер этапа; по вертикали время работы)



GPU – оптимизированный вариант

Размер сетки	Время выполнения, ms	Точность	Количество итераций
128*128	8.32	0.009801	1000
256*256	9.16	0.010302	1000
512*512	12.2053	0.010553	1000
1024*1024	35.6138	0.010678	1000

Диаграмма сравнения времени работы CPU-1, CPU-multicore, GPU - для разных размеров сеток



Вывод:

GPU выполняет задачу намного быстрее т.к. обладает большим кол-вом потоков + хранение данных в виде одномерного массива ускоряет доступ к данным и их очистку.

Код (оформелние: Monokai) + ссылка на GitHub

https://github.com/psan3333/parallels_tasks/tree/main/task2
Для CPU и GPU использовался один и тот же код.

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

int main(int argc, char* argv[]) {

    int size = atoi(argv[2]);

    int max_iter_input = atoi(argv[3]);
    int iter = 0, max_iter = 1000000;
    if (max_iter_input > max_iter){
        printf("Count of iterations mustn't exceed 10^6 operations");
        return 0;
    }

    double precision = atof(argv[1]);
    double max_precision = 0.000001, error = precision + 1.0;
    if (precision < max_precision) {
        printf("Precision mustn't be lower than 10^-6");
        return 0;
    }

    double* A = (double*)calloc(size * size, sizeof(double));
    double* Anew = (double*)calloc(size * size, sizeof(double));

    memset(A, 0, size * size * sizeof(double));
    memset(Anew, 0, size * size * sizeof(double));

    A[0] = 10.0;
    A[size - 1] = 20.0;
    A[size * size - 1] = 30.0;
    A[size * (size - 1)] = 20.0;

    Anew[0] = 10.0;
    Anew[size - 1] = 20.0;
    Anew[size * size - 1] = 30.0;
    Anew[size * (size - 1)] = 20.0;

    #pragma acc data copy(A[0:size*size], Anew[0:size*size])
    {
        clock_t start = clock();
        double step = 10.0 / (size - 1);

        #pragma acc parallel loop vector worker num_workers(4) vector_length(32)
        for (int i = 1; i < size - 1; i++)
        {
            A[i] = A[0] + i * step;
```

```
            A[i * size] = A[0] + i * step;
            A[size - 1 + size * i] = A[size - 1] + i * step;
            A[size * (size - 1) + i] = A[size * (size - 1)] + i * step;

            Anew[i] = Anew[0] + i * step;
            Anew[i * size] = Anew[0] + i * step;
            Anew[size - 1 + size * i] = Anew[size - 1] + i * step;
            Anew[size * (size - 1) + i] = Anew[size * (size - 1)] + i * step;
        }

        while (error > precision && iter < max_iter_input) {

            error = 0.0;
            ++iter;

            if (iter % 2) {
                #pragma acc parallel loop vector vector_length(256) gang num_gangs(256)
collapse(2) reduction(max:error)
                for (int i = 1; i < size - 1; i++){
                    for (int j = 1; j < size - 1; j++) {
                        Anew[i * size + j] = 0.25 * (A[i * size + j - 1] + A[(i - 1) * size +
j] + A[(i + 1) * size + j] + A[i * size + j + 1]);
                        error = fmax(error, fabs(Anew[i * size + j] - A[i * size + j]));
                    }
                }
                printf("%lf %d\n", error, iter);
            }
            else {
                #pragma acc parallel loop vector vector_length(256) gang num_gangs(256)
collapse(2) reduction(max:error)
                for (int i = 1; i < size - 1; i++){
                    for (int j = 1; j < size - 1; j++) {
                        A[i * size + j] = 0.25 * (Anew[i * size + j - 1] + Anew[(i - 1) * size
+ j] + Anew[(i + 1) * size + j] + Anew[i * size + j + 1]);
                        error = fmax(error, fabs(Anew[i * size + j] - A[i * size + j]));
                    }
                }
                printf("%lf %d\n", error, iter);
            }

        }

        clock_t end = clock();
        printf("%lf\n", (double)(end - start) / CLOCKS_PER_SEC);

    }

    printf("%d %lf\n", iter, error);

    free(Anew);
    free(A);

    return 0;
}
```