Теория Параллелизма

Отчет

Уравнение теплопроводности (openacc)

Выполнил 21932, Бабенко Егор Степанович

02.03.2022

Цель работы

Реализовать решение уравнение теплопроводности методом Якоба (пятиточечный шаблон) для двумерной сетки. Произвести профилирование программы для GPU и оптимизацию кода. Произвести сравнение времени работы на CPU и GPU.

Используемый компилятор: pgc++

Используемый профилировщик: nvprof, NsightSystem

Замер времени работы всей программы производился с помощью команды time. Время высчитывалось как среднее между пятью запусками.

CPU-onecore

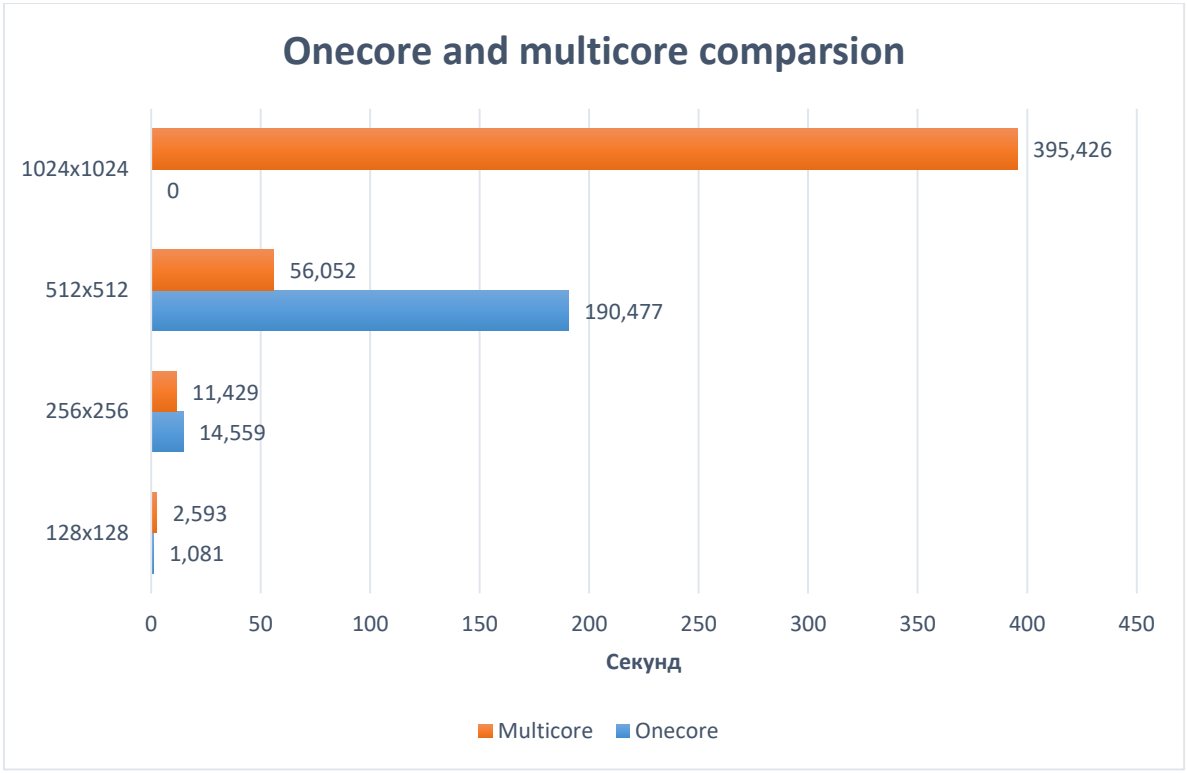
Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	1.081	9.9998e-07	30074
256x256	14.559	9.9993e-07	102885
512x512	190.477	9.99984e-07	339599
1024x1024	2285.354	1.36929e-06	1000000

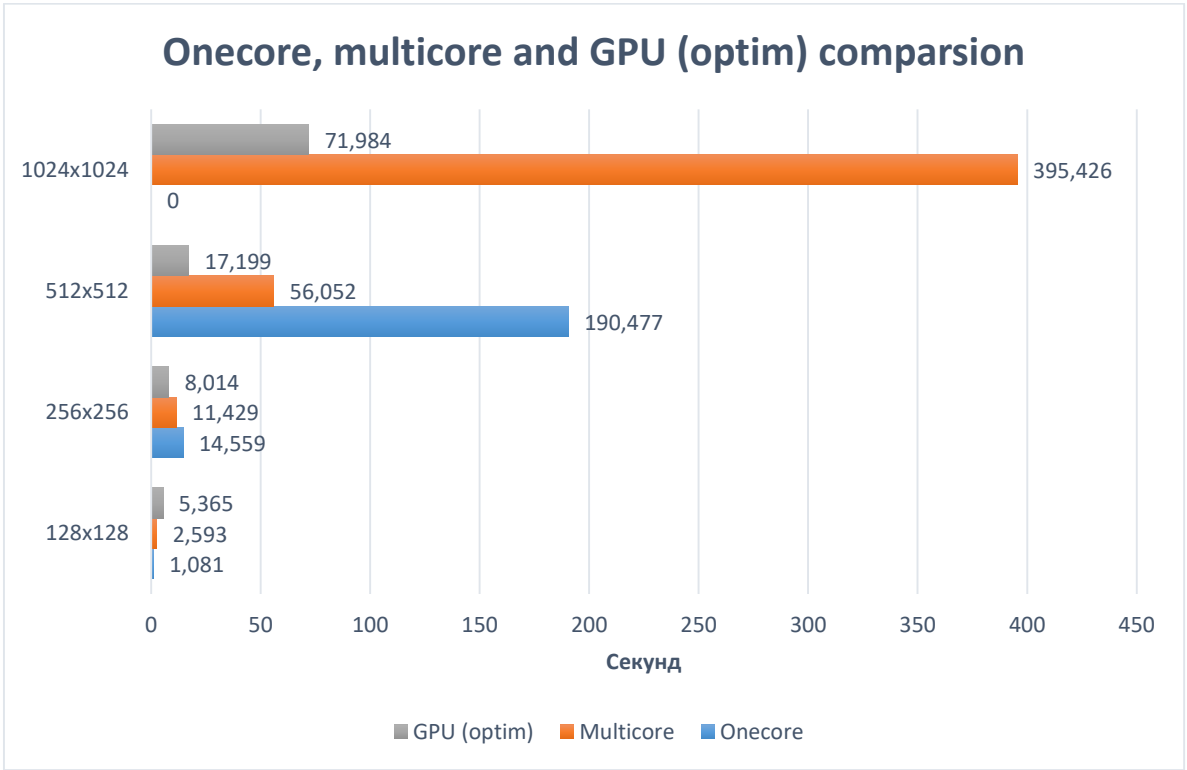
CPU-multicore

Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	2.593	9.9998e-07	30074
256x256	11.429	9.9993e-07	102885
512x512	56.052	9.99984e-07	339599
1024x1024	395.426	1.36929e-06	1000000

GPU-optimized

Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	5.365	9.9998e-07	30074
256x256	8.014	9.9993e-07	102885
512x512	17.199	9.99984e-07	339599
1024x1024	71.984	1.36929e-06	1000000

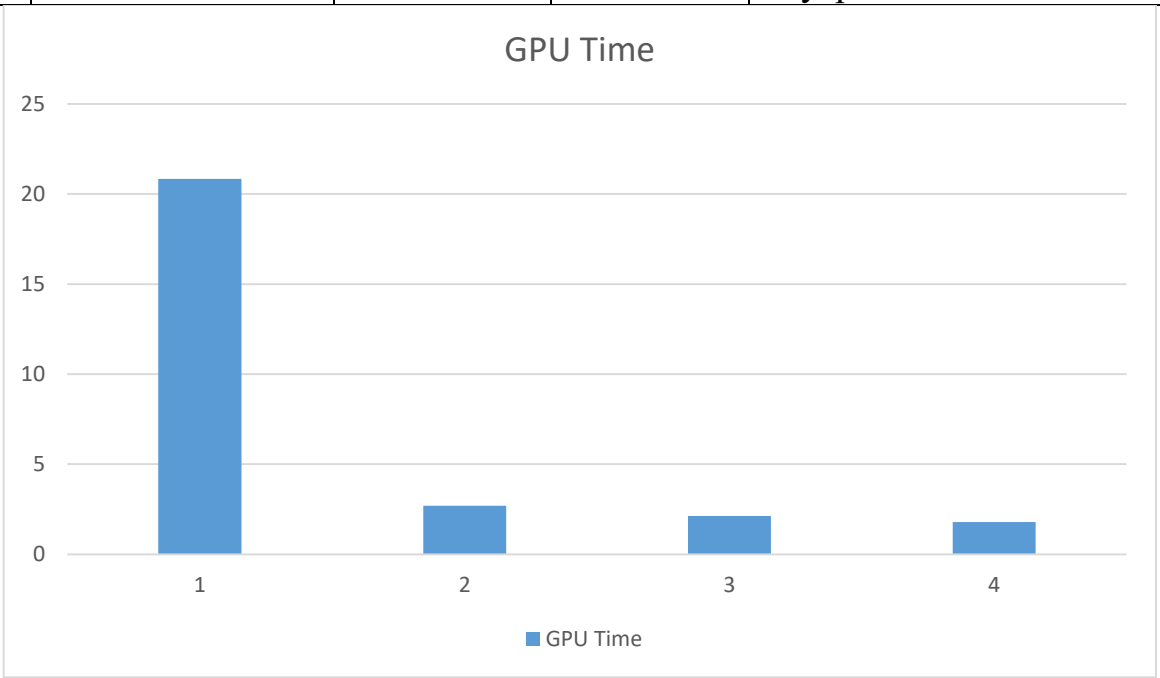

Onecore and multicore comparsion
1024x1024
512x512
256x256
128x128
395,426
0
56,052
190,477
11,429
14,559
2,593
1,081
0
50
100
150
200
250
300
350
400
450
Секунд
Multicore
Onecore


Onecore, multicore and GPU (optim) comparsion
1024x1024
512x512
256x256
128x128
71,984
395,426
0
17,199
56,052
190,477
8,014
11,429
14,559
5,365
2,593
1,081
0
50
100
150
200
250
300
350
400
450
Секунд
GPU (optim)
Multicore
Onecore

Этапы оптимизации кода на GPU

Этапы оптимизации на сетке 512x512

№	Время выполнения кода на GPU	Ошибка	Кол-во итераций	Комментарий
1	20.844 мс	0.107043	100	Попытка распараллелить весь цикл while
2	2.698 мс	0.107043	100	Распараллеливание только внутреннего цикла
3	2.132 мс	0.107043	100	Reduction на error, ручное управление памятью
4	1.796 мс	0.107043	100	Замена копирование DtoD на swap указателей внутри GPU



Вывод:

Для обработки сетки небольшого размера будет целесообразнее использовать однопоточный вариант, т.к. все возможные дополнительные издержки на синхронизацию потоков занимают порядочное количество времени. Но при увеличении размера сетки видно, что распараллеливание задачи более эффективно, чем выполнение в одном потоке.

Возможной оптимизацией кода на GPU еще может быть уменьшение количества обменов данных между CPU и GPU. Т.е. производить полный расчет всего цикла while на GPU.

Github: https://github.com/JooudDoo/Parallelism-Tasks

—

120 lines (98 sloc) | 3.24 KB — Raw | Blame

```cpp
#include <iostream>
#include <cstring>
#include <sstream>
#include <math.h>
#include <cmath>

#ifdef OPENACC__
#include <openacc.h>
#endif

#define at(arr, x, y) (arr[(x)*n+(y)])

void initArrays(double* mainArr, double* subArr, int& n, int& m){
    std::memset(mainArr, 0, sizeof(double)*(n)*(m));
    at(mainArr, 0, 0) = 10;
    at(mainArr, 0, m-1) = 20;
    at(mainArr, n-1, 0) = 20;
    at(mainArr, n-1, m-1) = 30;
    for(int i = 1; i < n-1; i++){
        at(mainArr,0,i) = (at(mainArr,0,m-1)-at(mainArr,0,0))/(m-1)*i+at(mainArr,0,0);
        at(mainArr,i,0) = (at(mainArr,n-1,0)-at(mainArr,0,0))/(n-1)*i+at(mainArr,0,0);

        at(mainArr,n-1,i) = (at(mainArr,n-1,m-1)-at(mainArr,n-1,0))/(m-1)*i+at(mainArr,n-1,0);
        at(mainArr,i,m-1) = (at(mainArr,n-1,m-1)-at(mainArr,0,m-1))/(m-1)*i+at(mainArr,0,m-1);
    }
    std::memcpy(subArr, mainArr, sizeof(double)*(n)*(m));
}

template<typename T>
T extractNumber(char* arr){
    std::stringstream stream;
    stream << arr;
    T result;
    if (!(stream >> result)){
        throw std::invalid_argument("Wrong argument type");
    }
    return result;
}

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

    bool showResultArr = false;
    double eps = 1E-6;
    int iterations = 1E6;
    int n = 10;
    int m = n;

    for(int arg = 0; arg < argc; arg++){
        if(std::strcmp(argv[arg], "-eps") == 0){
            eps = extractNumber<double>(argv[arg+1]);
            arg++;
        }
        else if(std::strcmp(argv[arg], "-i") == 0){
            iterations = extractNumber<int>(argv[arg+1]);
            arg++;
        }
        else if(std::strcmp(argv[arg], "-s") == 0){
            n = extractNumber<int>(argv[arg+1]);
            m = n;
            arg++;
        }
        else if(std::strcmp(argv[arg], "-show") == 0){
            showResultArr = true;
        }
    }

    std::cout << "Current settings:" << std::endl;
    std::cout << "\tEPS: " << eps << std::endl;
    std::cout << "\tMax iteration: " << iterations << std::endl;
    std::cout << "\tSize: " << n << 'x' << m << std::endl << std::endl;

    double* F = new double[n*m];
    double* Fnew = new double[n*m];

    initArrays(F, Fnew, n, m);

    double error = 0;
    int iteration = 0;

    #pragma acc enter data copyin(Fnew[:n*m], F[:n*m])

    do {
        error = 0;

        #pragma acc parallel loop collapse(2) present(Fnew[:n*m], F[:n*m]) reduction(max:error) vector_lengt
        for(int x = 1; x < n-1; x++){
            for(int y= 1; y < m-1; y++){
                at(Fnew,x,y) = 0.25 * (at(F, x+1,y) + at(F,x-1,y) + at(F,x,y-1) + at(F,x,y+1));
                error = fmax(error, fabs(at(Fnew,x,y) - at(F,x,y)));
            }
        }

        double* swap = F;
        F = Fnew;
        Fnew = swap;

#ifdef OPENACC__
        acc_attach((void**)F);
        acc_attach((void**)Fnew);
#endif

        iteration++;
    } while(iteration < iterations && error > eps);

    #pragma acc exit data delete(Fnew[:n*m]) copyout(F[:n*m])

    std::cout << "Iterations: " << iteration << std::endl;
    std::cout << "Error: " << error << std::endl;
    for(int x = 0; x < n && showResultArr; x++){
        for(int y = 0; y < m; y++){
            std::cout << at(F,x, y) << ' ';
        }
        std::cout << std::endl;
    }

    delete[] F;
    delete[] Fnew;

    return 0;
}
```