Теория параллелизма

Отчет

Уравнение теплопроводности на cuBLAS

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

Дата: 27.03.2023

Цели работы

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

Используемый компилятор – pgc++

Используемый профилировщик - nvprof

Как производили замер времени работы – PGI_ACC_TIME, time

Выполнение на CPU

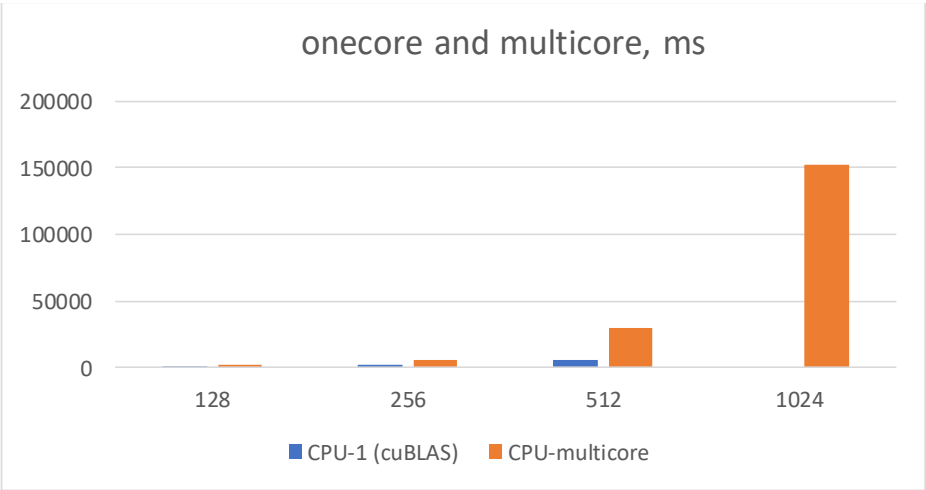
CPU-1 (cuBLAS/без cuBLAS)

Размер сетки	Время выполнения, ms	Точность (для cuBLAS и без него одинакова)	Количество итераций
128*128	983/999	0.000001	30101/30074
256*256	1944/15328	0.000001	102901/102885
512*512	5528/205320	0.000001	339601/339599

CPU-multicore (результаты без использования cuBLAS)

Размер сетки	Время выполнения, ms	Точность	Количество итераций
128*128	2124	0.000001	30074
256*256	5991	0.000001	102885
512*512	30300	0.000001	339599
1024*1024	152905	0.000001	1000000

Диаграмма сравнения время работы CPU-1(cuBLAS) и CPU-multicore



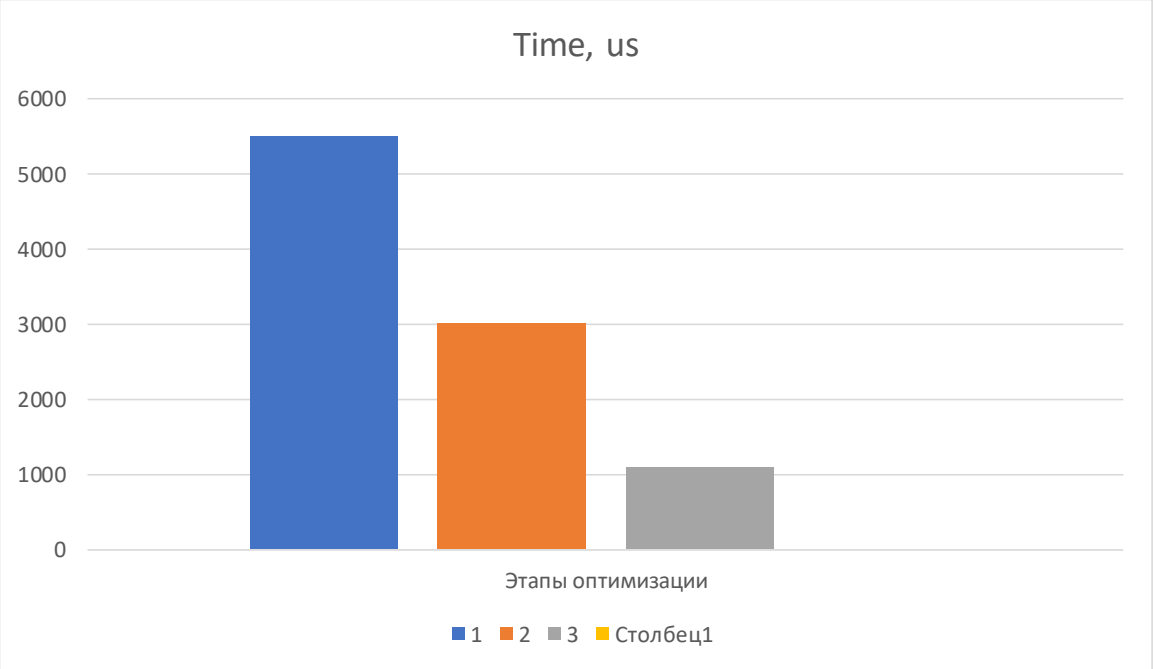
Выполнение на GPU

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	~1100	0.107043	100	Использование cuBLAS + изменение значение точности раз в сто итераций главного цикла

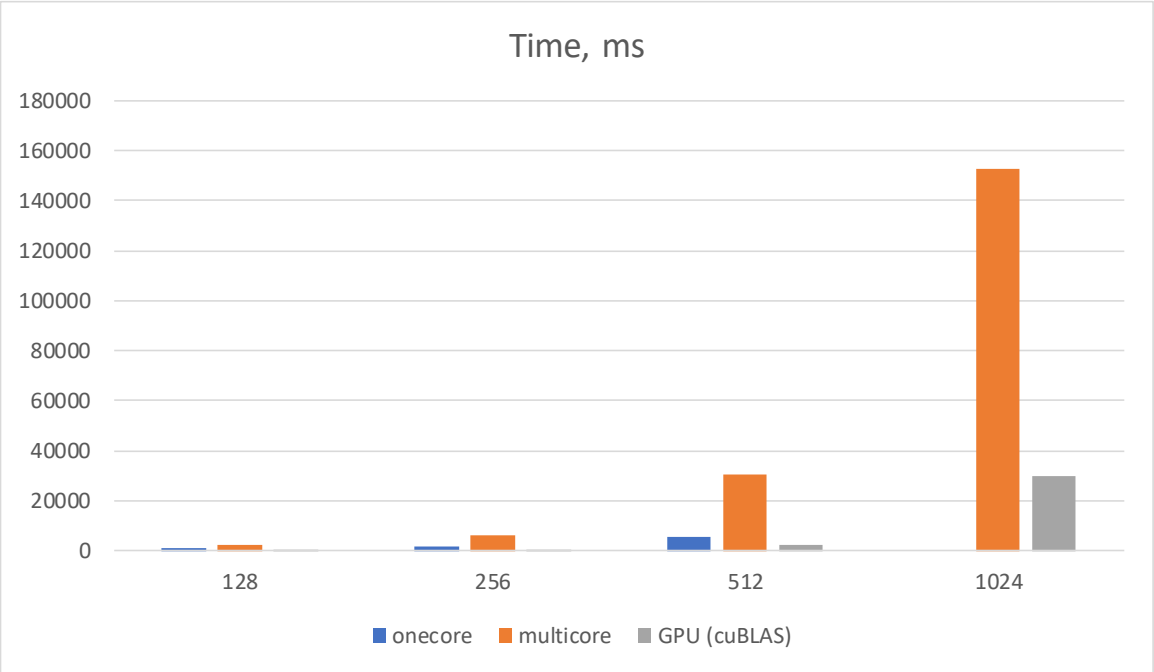
Диаграмма оптимизации (по горизонтали номер этапа; по вертикали время работы)



GPU – оптимизированный вариант с cuBLAS

Размер сетки	Время выполнения, ms	Точность	Количество итераций
128*128	110.6	0.000001	30101
256*256	465.8	0.000001	102901
512*512	2500	0.000001	339601
1024*1024	29700	0.000001	1000000

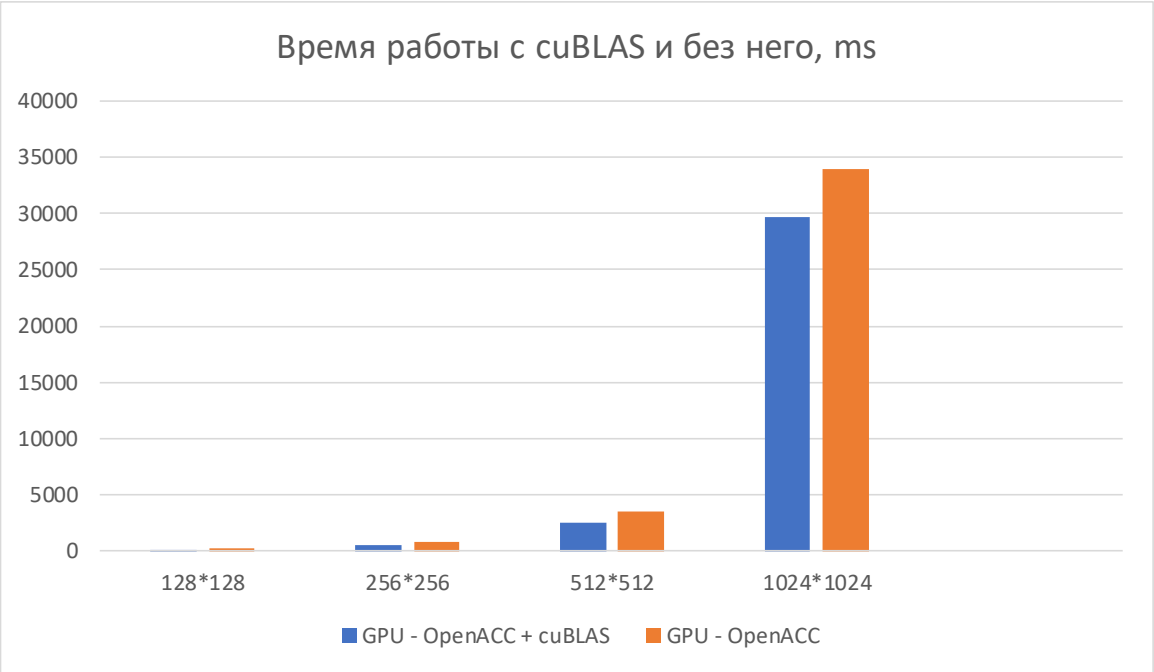
Диаграмма сравнения времени работы CPU-1 (cuBLAS), CPU-multicore, GPU (cuBLAS) - для разных размеров сеток



GPU – без cuBLAS

Размер сетки	Время выполнения, ms	Точность	Количество итераций
128*128	223	0.000001	30074
256*256	818	0.000001	102885
512*512	3559	0.000001	339599
1024*1024	33965	0.000001	1000000

Диаграмма сравнения работы кода на GPU с cuBLAS и без него.



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.

Также при правильном использовании фукнций из библиотеки cuBLAS можно ускорить код

Код (оформелние: Monokai) + ссылка на GitHub

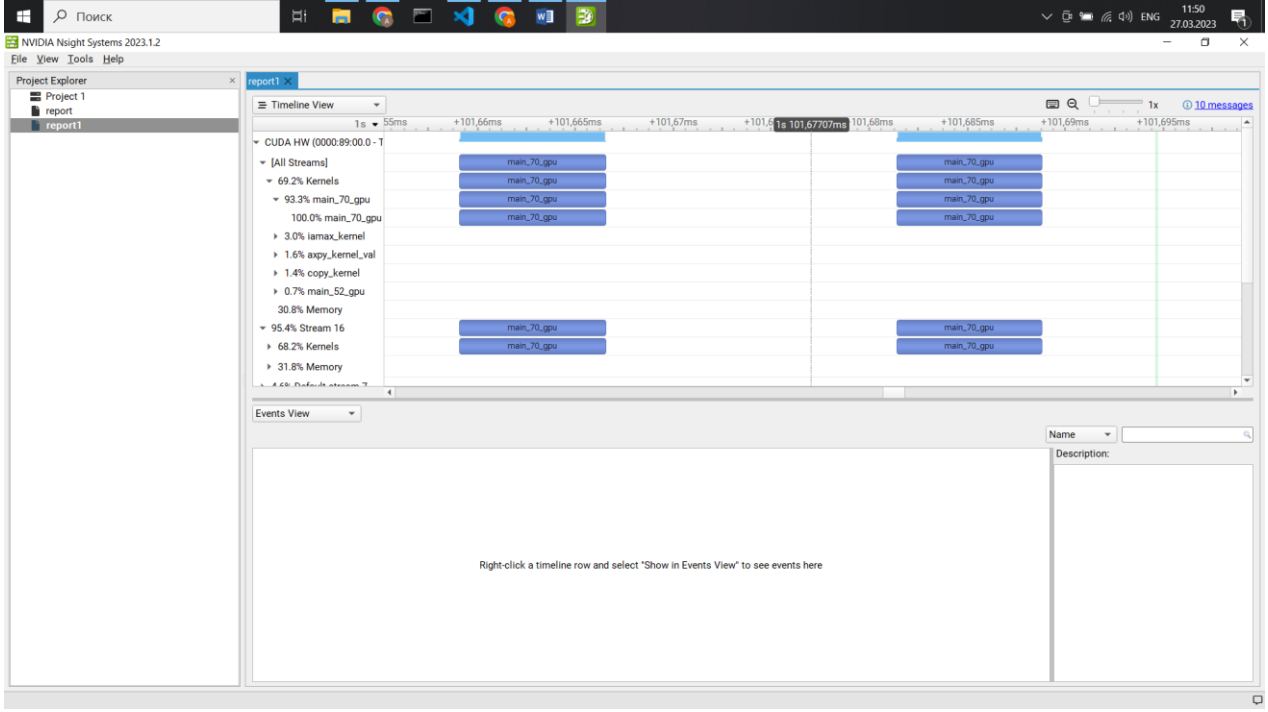
https://github.com/psan3333/parallels_tasks/tree/main/task3
Для CPU и GPU использовался один и тот же код.

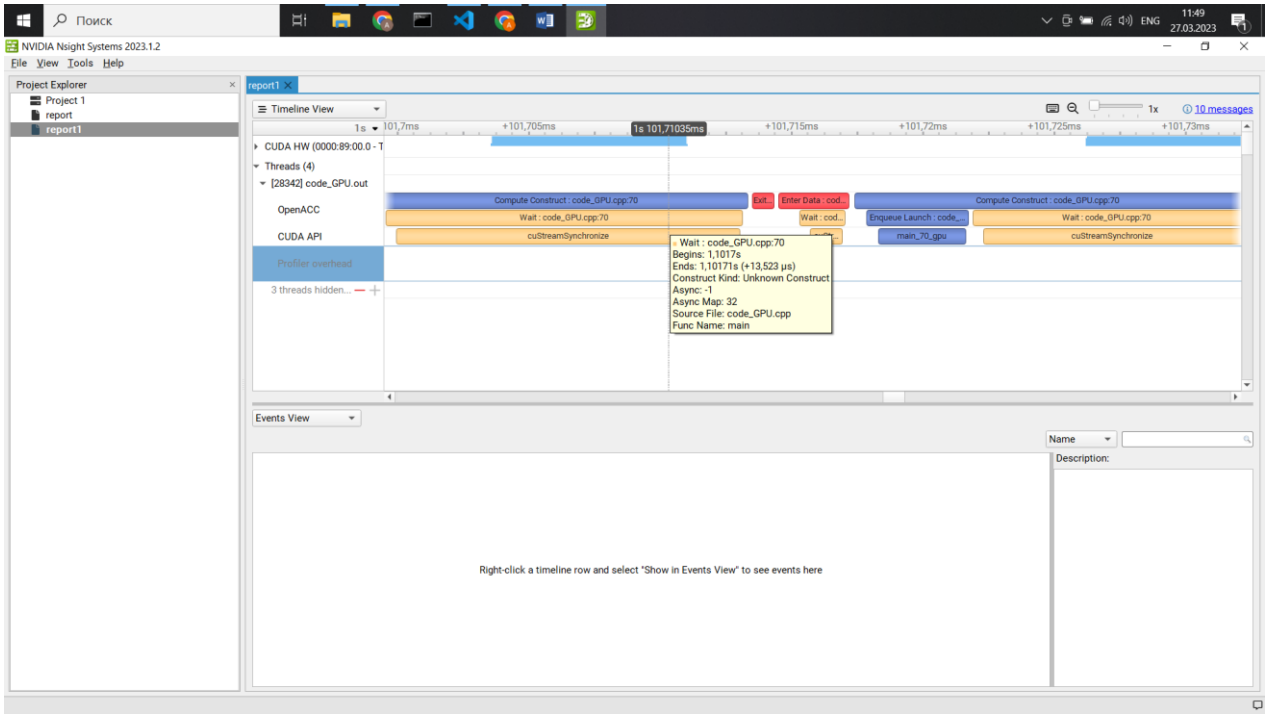
```cpp
#include <iostream>
#include <cstring>
#include <sstream>
#include <cmath>
#include <assert.h>
#include <cublas_v2.h>


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

    //parse command prompt arguments
    int size = std::stoi(argv[2]);
    int max_iter_input = std::stoi(argv[3]);
    int max_iter = 1000000;
    if (max_iter_input > max_iter) {
        std::cerr << "Count of iterations mustn't exceed 10^6 operations" << std::endl;
        exit(EXIT_FAILURE);
    }

    double precision = std::stod(argv[1]);
    double max_precision = 0.000001;
    if (precision < max_precision) {
        std::cerr << "Precision mustn't be lower than 10^-6" << std::endl;
        exit(EXIT_FAILURE);
    }

    //allocate memory and set A = 0
    size_t matrixSize = size * size;
    double* A = new double[matrixSize];
    double* Anew = new double[matrixSize];
    std::memset(A, 0, matrixSize * sizeof(double));
```

```
    //init angles values
    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;

    //init data for main loop execution
    double step = 10.0 / (size - 1);
    int iter = 0;
    double error = precision + 1.0;
    cublasHandle_t handle;
    cublasCreate(&handle);

    #pragma acc enter data copyin(A[:matrixSize], Anew[:matrixSize])
    {

        //interpolation for angles
        #pragma acc parallel loop
        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;
        }

        //main loop
        while (iter < max_iter_input && error > precision) {

            //calculating average values
            #pragma acc data present(A[:matrixSize], Anew[:matrixSize]) //updating pointers on
GPU
            #pragma acc parallel loop independent collapse(2) vector vector_length(256) gang
num_gangs(256)
            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]);
                }
            }

            //updating error only 1/100 of main loop iterations
            if (iter % 100 == 0 || iter + 1 == max_iter_input) {
                error = 0.0;
                double a = -1;
                int idxMax = 0;

                cublasStatus_t stat1, stat2, stat3;
```

```
                #pragma acc host_data use_device(A, Anew)
                {
                    stat1 = cublasDaxpy(handle, matrixSize, &a, Anew, 1, A, 1);
                    stat2 = cublasIdamax(handle, matrixSize, A, 1, &idxMax);
                }

                //check for failure
                if(stat1 != CUBLAS_STATUS_SUCCESS)
                    exit(EXIT_FAILURE);
                if(stat2 != CUBLAS_STATUS_SUCCESS)
                    exit(EXIT_FAILURE);

                #pragma acc update host(A[idxMax - 1])
                error = std::abs(A[idxMax - 1]);

                #pragma acc host_data use_device(A, Anew)
                {
                    stat3 = cublasDcopy(handle, matrixSize, Anew, 1, A, 1);
                }
                if(stat3 != CUBLAS_STATUS_SUCCESS)
                    exit(EXIT_FAILURE);
            }

            std::swap(A, Anew);
            iter++;
        }

        printf("Iterations: %d\nPrecision: %lf\n", iter, error);
    }

    cublasDestroy(handle);
    #pragma acc exit data delete(A[:matrixSize], Anew[:matrixSize])
    delete[] Anew;
    delete[] A;

    return 0;
}
```

Диаграммы Nsight Systems (стека 512*512, кол-во итераций - 1000)


Поиск
ENG
11:50
27.03.2023
NVIDIA Nsight Systems 2023.1.2
File View Tools Help
Project Explorer
Project 1
report
report1
report1
Timeline View
1x
10 messages
1s 101,67707ms
CUDA HW (0000:89:00.0 - T
[All Streams]
69.2% Kernels
93.3% main_70_gpu
100.0% main_70_gpu
3.0% iamax_kernel
1.6% axpy_kernel_val
1.4% copy_kernel
0.7% main_52_gpu
30.8% Memory
95.4% Stream 16
68.2% Kernels
31.8% Memory
main_70_gpu
Events View
Name
Description:
Right-click a timeline row and select "Show in Events View" to see events here


Поиск
ENG
11:49
27.03.2023
NVIDIA Nsight Systems 2023.1.2
File View Tools Help
Project Explorer
Project 1
report
report1
report1
Timeline View
1x
10 messages
1s 101,71035ms
CUDA HW (0000:89:00.0 - T
Threads (4)
[28342] code_GPU.out
OpenACC
CUDA API
Profiler overhead
3 threads hidden...
Compute Construct : code_GPU.cpp:70
Wait : code_GPU.cpp:70
cuStreamSynchronize
main_70_gpu
Wait : code_GPU.cpp:70
Begins: 1,1017s
Ends: 1,10171s (+13,523 µs)
Construct Kind: Unknown Construct
Async: -1
Async Map: 32
Source File: code_GPU.cpp
Func Name: main
Events View
Name
Description:
Right-click a timeline row and select "Show in Events View" to see events here