Теория параллелилзма

Отчёт

Уравнение теплопроводности

Выполнил Грищенко Александр Михайлович, 21932

03.2023

1 Цели работы

Реализовать решение уравнение теплопроводности (пятиточечный шаблон) в двумерной области на равномерных сетках.
Перенести программу на GPU используя директивы OpenACC.
Произвести профилирование программы и оптимизацию кода.
Сравнить время работы на CPU и GPU.

2 Используемый компилятор

g++ для исполнения в CPU-onecore и pgc++ для исполнения на GPU и CPU-multicore.

3 Используемый профилировщик

nsys (NVIDIA Nsight Systems) с OpenACC trace.

4 Как проводился замер времени работы

Для замера времени работы использовалась библиотека chrono.
Замер времени производился несколько раз, затем бралось среднее время.

5 Выполнение на CPU

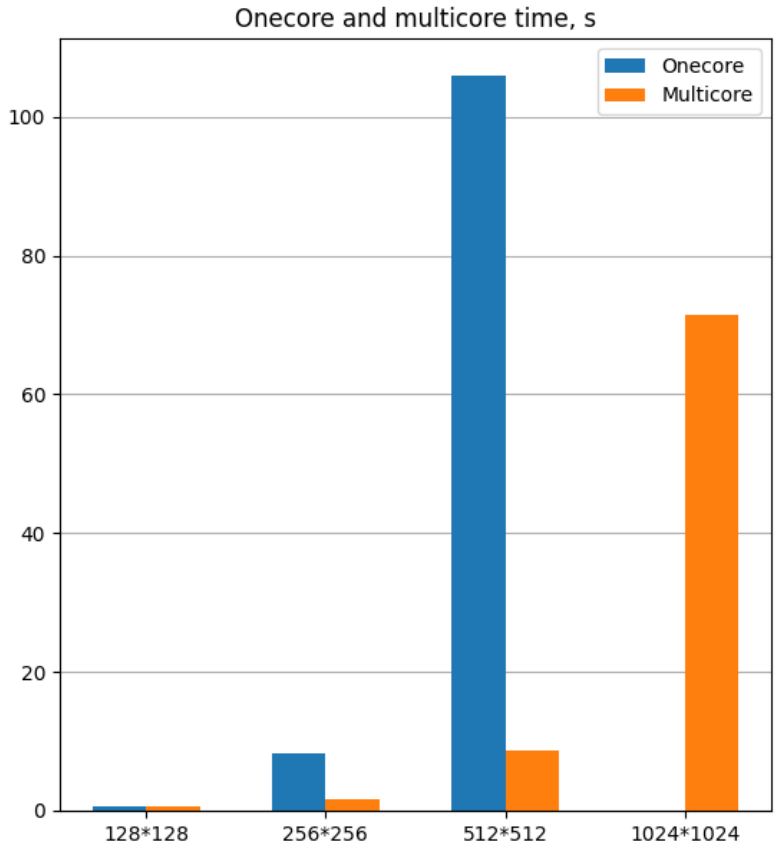
5.1 CPU-onecore

Размер сетки	Время выполнения, с	Точность	Количество операций
128*128	0.72	1e-06	7737
256*256	8.3	1e-06	21679
512*512	106	1e-06	68410

5.2 CPU-multicore

Размер сетки	Время выполнения, с	Точность	Количество операций
128*128	0.6	1e-06	7969
256*256	1.7	1e-06	21322
512*512	8.7	1e-06	68283
1024*1024	71.4	1e-06	212351

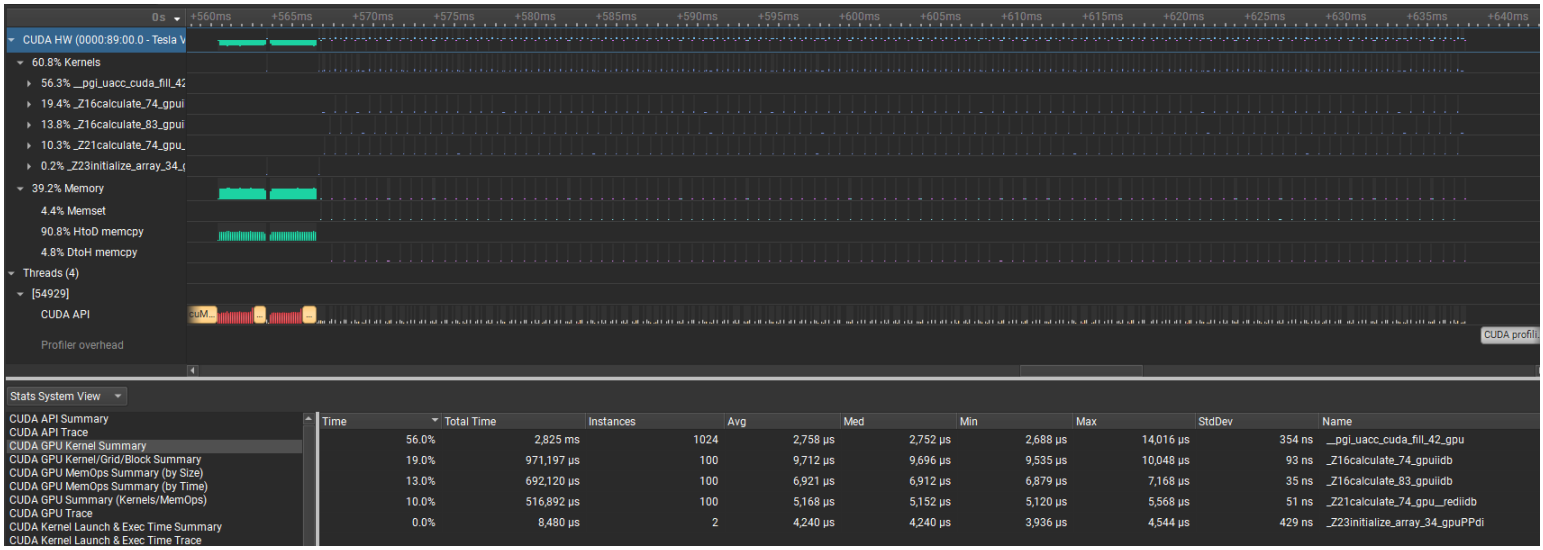
5.3 Диаграмма сравнения время работы CPU-onecore и CPU-multicore



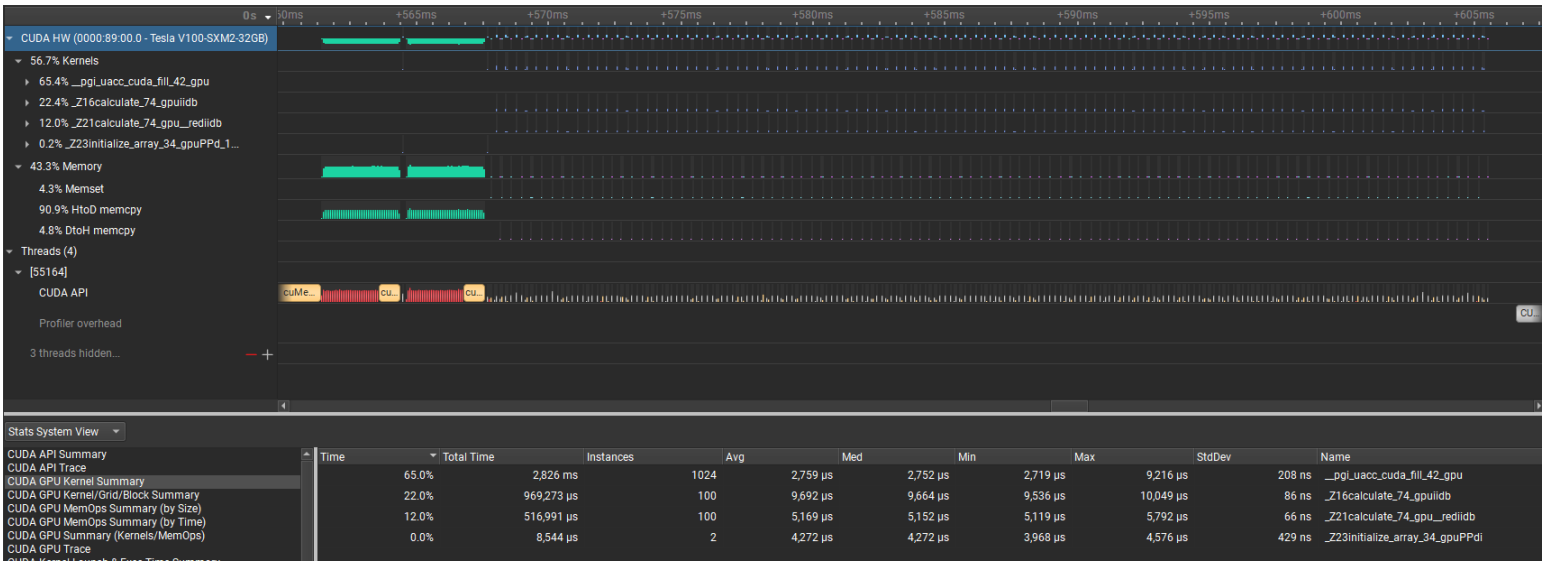
6 Выполнение на GPU

6.1 Этапы оптимизации на сетке 512*512 (количество итераций при профилировании 100)

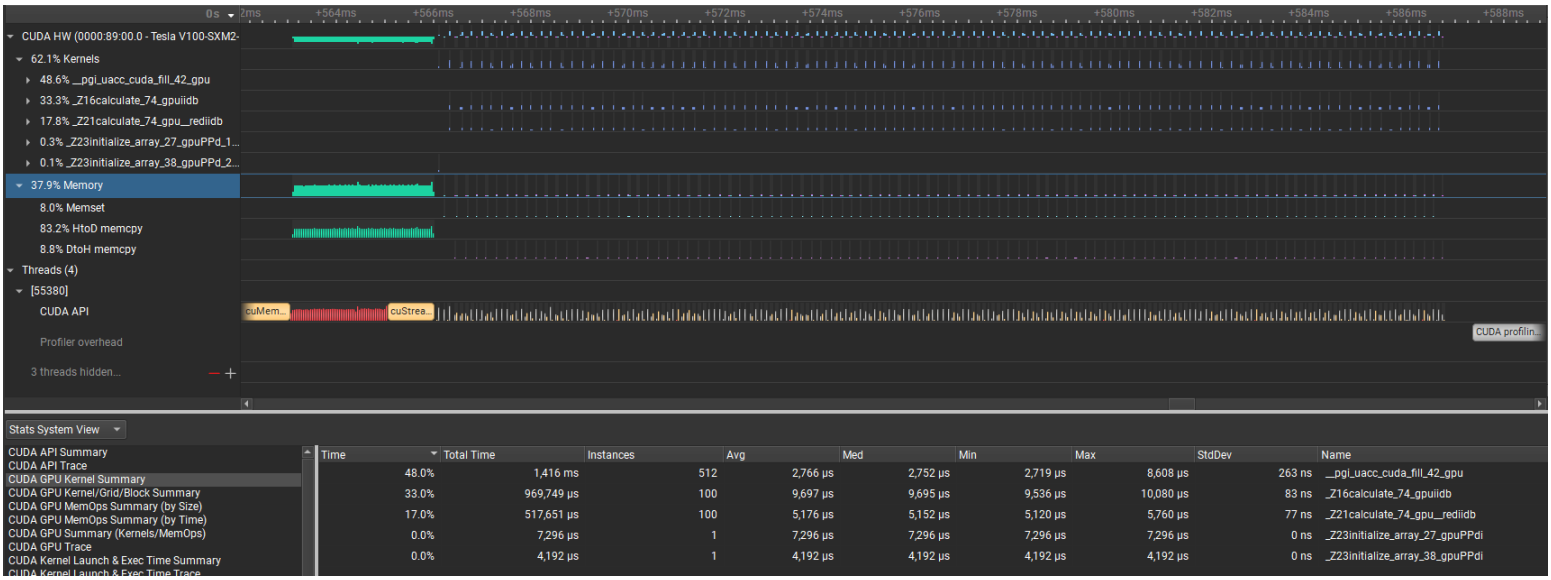
Этап №	Время выполнения, с	Точность	Количество операций	Комментарии (что было сделано)
1	0.28	0.1062	100	Распараллелены циклы, reduction(max:error)
2	0.24	0.1062	100	Замена копирования массива swap'ом через указатели
3	0.22	0.1281	100	Работа только с одной матрицей, изначальная инициализация массива значениями 20



Этап 1

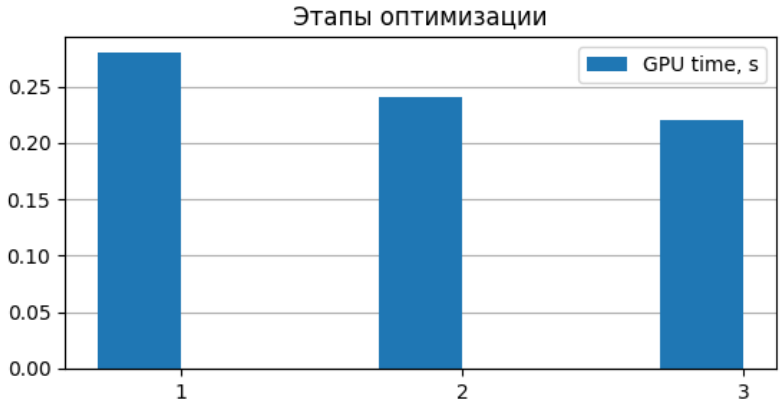

CUDA HW (0000:89:00.0 - Tesla V100-SXM2-32GB)
56.7% Kernels
65.4% __pgi_uacc_cuda_fill_42_gpu
22.4% _Z16calculate_74_gpuiidb
12.0% _Z21calculate_74_gpu__rediidb
0.2% _Z23initialize_array_34_gpuPPd_1...
43.3% Memory
4.3% Memset
90.9% HtoD memcpy
4.8% DtoH memcpy
Threads (4)
[55164]
CUDA API
Profiler overhead
3 threads hidden...
Stats System View
CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Kernel/Grid/Block Summary
CUDA GPU MemOps Summary (by Size)
CUDA GPU MemOps Summary (by Time)
CUDA GPU Summary (Kernels/MemOps)
CUDA GPU Trace
Time | Total Time | Instances | Avg | Med | Min | Max | StdDev | Name
65.0% | 2,826 ms | 1024 | 2,759 µs | 2,752 µs | 2,719 µs | 9,216 µs | 208 ns | __pgi_uacc_cuda_fill_42_gpu
22.0% | 969,273 µs | 100 | 9,692 µs | 9,664 µs | 9,536 µs | 10,049 µs | 86 ns | _Z16calculate_74_gpuiidb
12.0% | 516,991 µs | 100 | 5,169 µs | 5,152 µs | 5,119 µs | 5,792 µs | 66 ns | _Z21calculate_74_gpu__rediidb
0.0% | 8,544 µs | 2 | 4,272 µs | 4,272 µs | 3,968 µs | 4,576 µs | 429 ns | _Z23initialize_array_34_gpuPPdi

Этап 2


CUDA HW (0000:89:00.0 - Tesla V100-SXM2-
62.1% Kernels
48.6% __pgi_uacc_cuda_fill_42_gpu
33.3% _Z16calculate_74_gpuiidb
17.8% _Z21calculate_74_gpu__rediidb
0.3% _Z23initialize_array_27_gpuPPd_1...
0.1% _Z23initialize_array_38_gpuPPd_2...
37.9% Memory
8.0% Memset
83.2% HtoD memcpy
8.8% DtoH memcpy
Threads (4)
[55380]
CUDA API
Profiler overhead
3 threads hidden...
Stats System View
CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Kernel/Grid/Block Summary
CUDA GPU MemOps Summary (by Size)
CUDA GPU MemOps Summary (by Time)
CUDA GPU Summary (Kernels/MemOps)
CUDA GPU Trace
CUDA Kernel Launch & Exec Time Summary
Time | Total Time | Instances | Avg | Med | Min | Max | StdDev | Name
48.0% | 1,416 ms | 512 | 2,766 µs | 2,752 µs | 2,719 µs | 8,608 µs | 263 ns | __pgi_uacc_cuda_fill_42_gpu
33.0% | 969,749 µs | 100 | 9,697 µs | 9,695 µs | 9,536 µs | 10,080 µs | 83 ns | _Z16calculate_74_gpuiidb
17.0% | 517,651 µs | 100 | 5,176 µs | 5,152 µs | 5,120 µs | 5,760 µs | 77 ns | _Z21calculate_74_gpu__rediidb
0.0% | 7,296 µs | 1 | 7,296 µs | 7,296 µs | 7,296 µs | 7,296 µs | 0 ns | _Z23initialize_array_27_gpuPPdi
0.0% | 4,192 µs | 1 | 4,192 µs | 4,192 µs | 4,192 µs | 4,192 µs | 0 ns | _Z23initialize_array_38_gpuPPdi

Этап 3

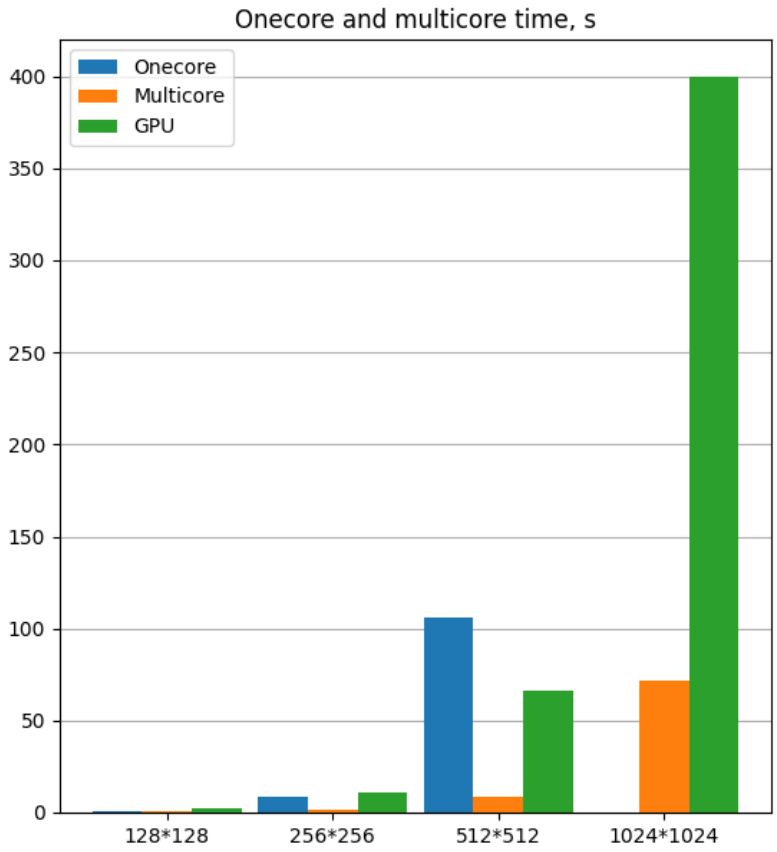
6.2 Диаграмма оптимизации (по горизонтали номер этапа; по вертикали время работы)



6.3 GPU – оптимизированный вариант

Размер сетки	Время выполнения, с	Точность	Количество опреаций
128*128	2	1e-06	30077
256*256	10.6	1e-06	102885
512*512	66.5	1e-06	350712
1024*1024	400	2e-06	1000000

7 Диаграмма сравнения времени работы CPU-one, CPU-multi, GPU (оптимизированный вариант) для разных размеров сеток



8 Вывод

CPU-multicore справляется с этой задачей лучше всех.

9 Приложение

9.1 Ссылка на GitHub

https://github.com/busyhedg03/ParallelismTheory/tree/master/task_2

```cpp
#include <iostream>
#include <cmath>
#include <chrono>

#ifdef _FLOAT
#define T float
#define MAX std::fmaxf
#else
#define T double
#define MAX std::fmax
#endif

void print_array(T **A, int size)
{
#pragma acc update host(A[:size][:size])
    std::cout.precision(4);
    for (int i = 0; i < size; i += 1)
    {
        for (int j = 0; j < size; j += 1)
            std::cout << A[i][j] << "\t";
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

void initialize_array(T **A, int size)
{
#pragma acc parallel loop collapse(2) present(A[:size][:size])
    for (int i = 1; i < size - 1; i++)
        for (int j = 1; j < size - 1; j++)
            A[i][j] = 20; // mode
    A[0][0] = 10.0;
    A[0][size - 1] = 20.0;
    A[size - 1][size - 1] = 30.0;
    A[size - 1][0] = 20.0;
#pragma acc update device(A[:size][:size])

    T step = 10.0 / (size - 1);
#pragma acc parallel loop present(A[:size][:size])
    for (int i = 1; i < size - 1; i++)
    {
        T addend = step * i;
        A[0][i] = A[0][0] + addend; // horizontal left
        A[size - 1][i] = A[size - 1][0] + addend; // horizontal right
        A[i][0] = A[0][0] + addend; // vertical left
        A[i][size - 1] = A[0][size - 1] + addend; // vertical right
    }
}

void delete_2d_array(T **A, int size){
    for (int i = 0; i < size; i++)
        delete[] A[i];
    delete[] A;
}
```

```
56  void calculate(int net_size=12, int iter_max=1e6, T accuracy=1e-6, bool res=false) {
57      // Initialization
58      T **A = new T *[net_size];
59      for (int i = 0; i < net_size; i++)
60          A[i] = new T[net_size];
61
62  #pragma acc enter data create(A[:net_size][:net_size])
63
64      // 10 20 30 20
65      initialize_array(A, net_size);
66
67      //print_array(A, net_size); //check initialization
68
69      T error;
70      int iter = 0;
71  #pragma acc enter data create(error)
72      std::cout.precision(4);
73      do {
74          error = 0.0;
75  #pragma acc update device(error)
76  #pragma acc parallel loop collapse(2) independent reduction(max:error)
77          for (int j = 1; j < net_size - 1; j++)
78              for (int i = 1; i < net_size - 1; i++)
79              {
80                  double temp = A[j][i];
81                  // Average
82                  A[j][i] = (A[j][i + 1] + A[j][i - 1] + A[j - 1][i] + A[j + 1][i]) * 0.25;
83                  error = MAX(error, std::abs(temp - A[j][i]));
84              }
85
86          iter++;
87  #pragma acc update host(error)
88      } while (error > accuracy && iter < iter_max);
89
90      std::cout << "iter=" << iter << ",\terror=" << error << std::endl;
91      if(res) print_array(A, net_size);
92  #pragma acc exit data delete(A[:net_size][:net_size], error)
93      delete_2d_array(A, net_size);
94  }
```

```cpp
int main(int argc, char *argv[])
{
    auto begin_main = std::chrono::steady_clock::now();
    int net_size = 12, iter_max = 1e6;
    T accuracy = 1e-6;
    bool res = false;
    for(int arg = 1; arg < argc; arg++) {
        std::string str = argv[arg];
        if(!str.compare("-a")) {
#ifdef _FLOAT
            accuracy = std::stof(argv[arg + 1]);
#else
            accuracy = std::stod(argv[arg + 1]);
#endif
            arg++;
        }
        else if(!str.compare("-i")) {
            iter_max = (int)std::stod(argv[arg + 1]); //1e6
            arg++;
        }
        else if(!str.compare("-s")) {
            net_size = std::stoi(argv[arg + 1]);
            arg++;
        }
        else if(!str.compare("-res")) {
            res = true;
        }
    }
    calculate(net_size, iter_max, accuracy, res);
    auto end_main = std::chrono::steady_clock::now();
    int time_spent_main = std::chrono::duration_cast<std::chrono::milliseconds>(end_main - begin_main).count();
    std::cout << "The elapsed time is:\nmain\t\t\t" << time_spent_main << " ms\n";
    return 0;
}
```