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

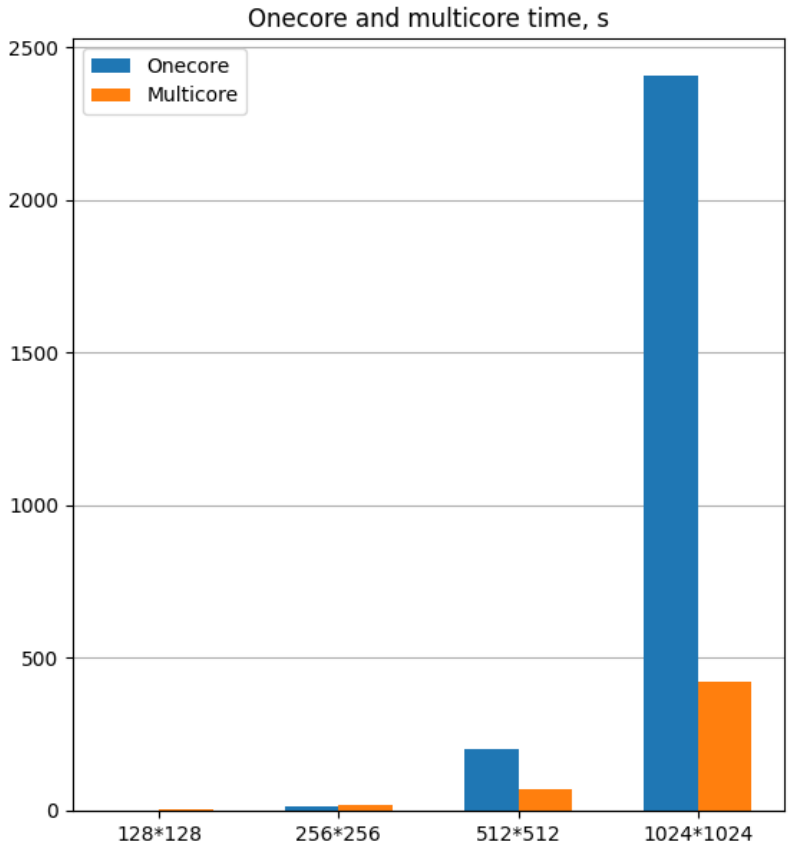
5.1 CPU-onecore

Размер сетки	Время выполнения, с	Точность	Количество операций
128*128	1.1	1e-06	30074
256*256	15.13	9.99e-07	102885
512*512	203.1	1e-06	339599
1024*1024	2408	1.37e-06	1e6

5.2 CPU-multicore

Размер сетки	Время выполнения, с	Точность	Количество операций
128*128	3.52	1e-06	30074
256*256	17.98	9.99e-07	102885
512*512	70.41	1e-06	339599
1024*1024	421.4	1.37e-06	1e6

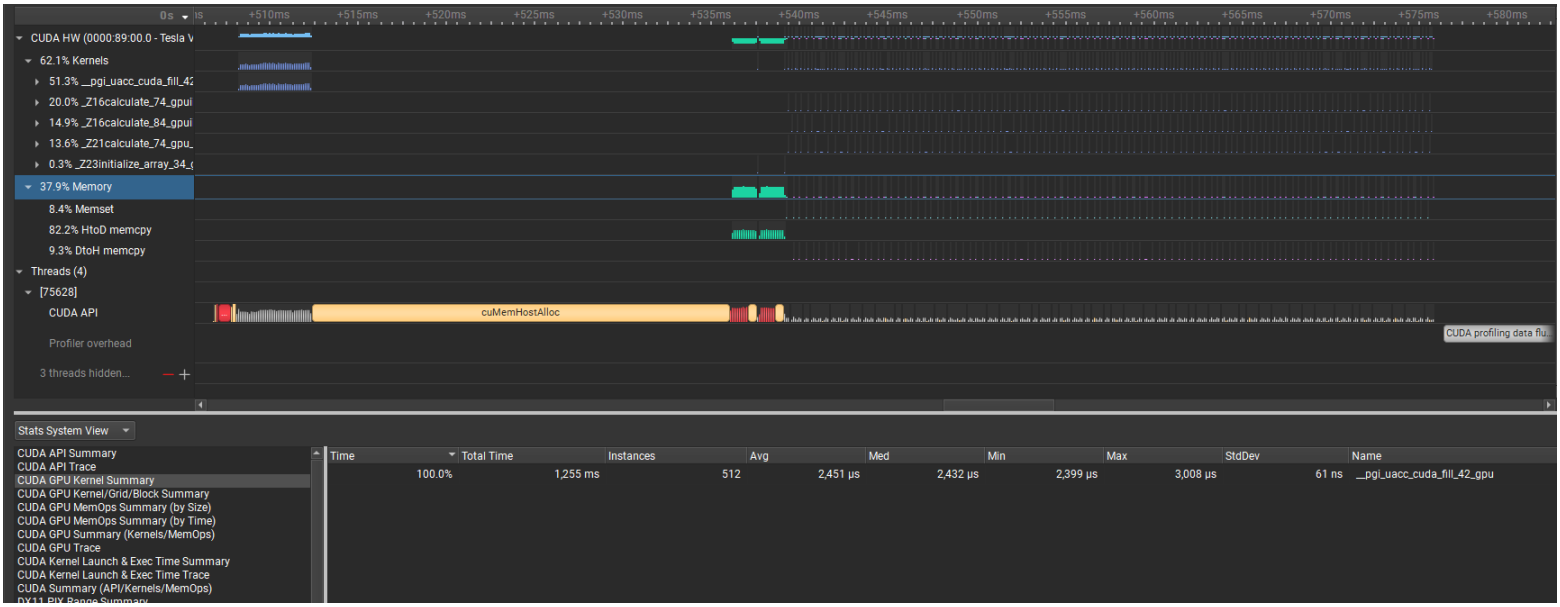
5.3 Диаграмма сравнения время работы CPU-onecore и CPU-multicore



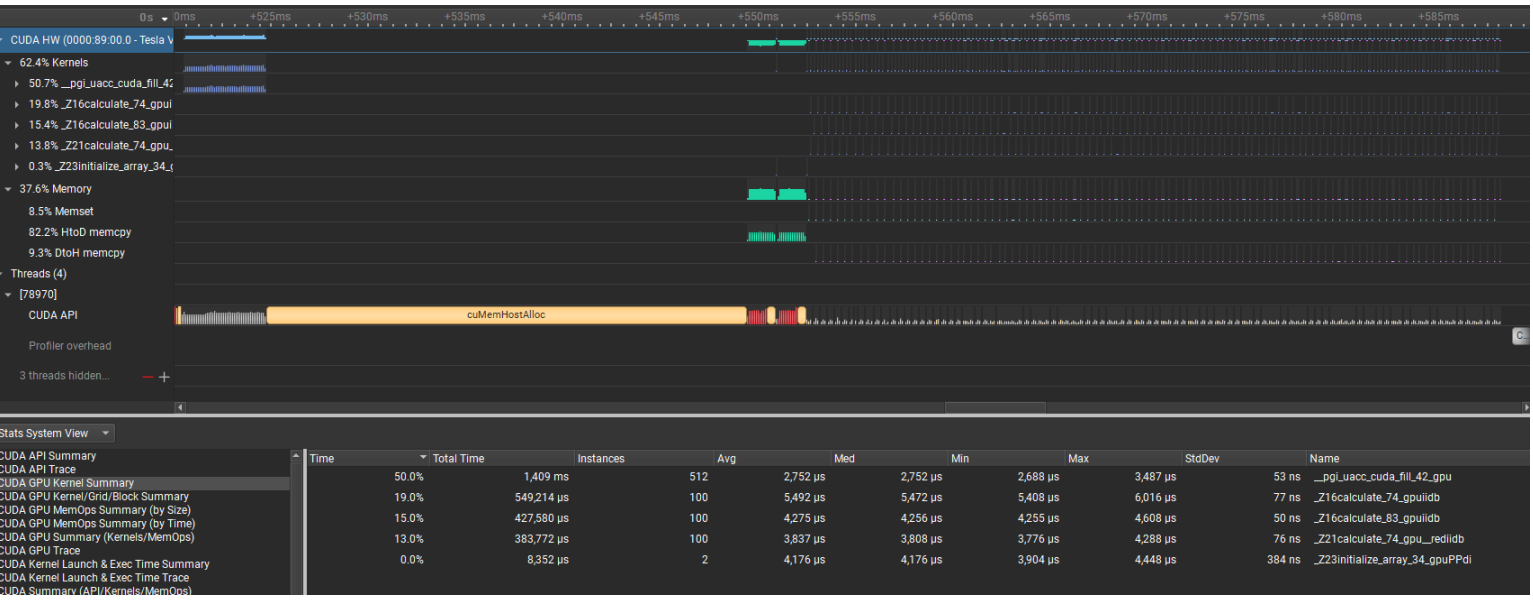
6 Выполнение на GPU

6.1 Этапы оптимизации на сетке 256*256 (количество итераций при профилировании 100)

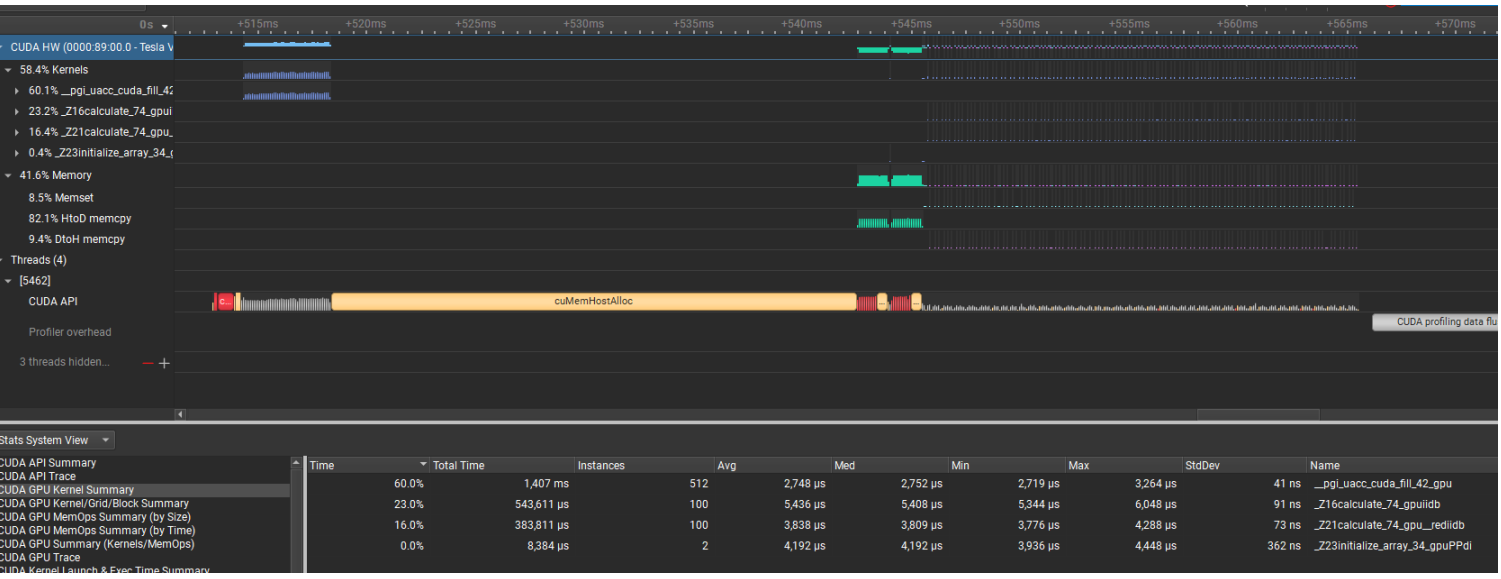
Этап №	Время выполнения, мс	Точность	Количество операций	Комментарии (что было сделано)
1	1.3	0.1062	100	Распараллелены циклы
2	3	0.1062	100	collapse(2), reduction(max:error)
3	2.3	0.1062	100	Замена копирования массива swap'ом через указатели



Этап 1

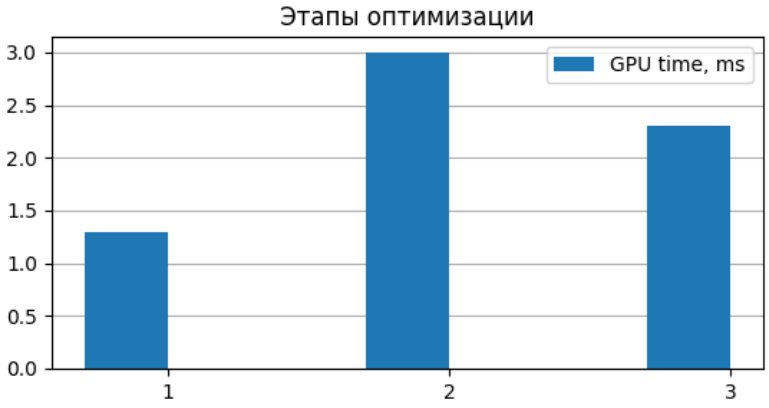

0s
0ms
+525ms
+530ms
+535ms
+540ms
+545ms
+550ms
+555ms
+560ms
+565ms
+570ms
+575ms
+580ms
+585ms
CUDA HW (0000:89:00.0 - Tesla V
62.4% Kernels
50.7% __pgi_uacc_cuda_fill_42
19.8% _Z16calculate_74_gpui
15.4% _Z16calculate_83_gpui
13.8% _Z21calculate_74_gpu_
0.3% _Z23initialize_array_34_(
37.6% Memory
8.5% Memset
82.2% HtoD memcpy
9.3% DtoH memcpy
Threads (4)
[78970]
CUDA API
cuMemHostAlloc
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
CUDA Kernel Launch & Exec Time Trace
CUDA Summary (API/Kernels/MemOps)
Time
Total Time
Instances
Avg
Med
Min
Max
StdDev
Name
50.0% 1,409 ms 512 2,752 µs 2,752 µs 2,688 µs 3,487 µs 53 ns __pgi_uacc_cuda_fill_42_gpu
19.0% 549,214 µs 100 5,492 µs 5,472 µs 5,408 µs 6,016 µs 77 ns _Z16calculate_74_gpuiidb
15.0% 427,580 µs 100 4,275 µs 4,256 µs 4,255 µs 4,608 µs 50 ns _Z16calculate_83_gpuiidb
13.0% 383,772 µs 100 3,837 µs 3,808 µs 3,776 µs 4,288 µs 76 ns _Z21calculate_74_gpu__rediidb
0.0% 8,352 µs 2 4,176 µs 4,176 µs 3,904 µs 4,448 µs 384 ns _Z23initialize_array_34_gpuPPdi

Этап 2


0s
+515ms
+520ms
+525ms
+530ms
+535ms
+540ms
+545ms
+550ms
+555ms
+560ms
+565ms
+570ms
CUDA HW (0000:89:00.0 - Tesla V
58.4% Kernels
60.1% __pgi_uacc_cuda_fill_42
23.2% _Z16calculate_74_gpui
16.4% _Z21calculate_74_gpu_
0.4% _Z23initialize_array_34_(
41.6% Memory
8.5% Memset
82.1% HtoD memcpy
9.4% DtoH memcpy
Threads (4)
[5462]
CUDA API
cuMemHostAlloc
CUDA profiling data flu
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
60.0% 1,407 ms 512 2,748 µs 2,752 µs 2,719 µs 3,264 µs 41 ns __pgi_uacc_cuda_fill_42_gpu
23.0% 543,611 µs 100 5,436 µs 5,408 µs 5,344 µs 6,048 µs 91 ns _Z16calculate_74_gpuiidb
16.0% 383,811 µs 100 3,838 µs 3,809 µs 3,776 µs 4,288 µs 73 ns _Z21calculate_74_gpu__rediidb
0.0% 8,384 µs 2 4,192 µs 4,192 µs 3,936 µs 4,448 µs 362 ns _Z23initialize_array_34_gpuPPdi

Этап 3

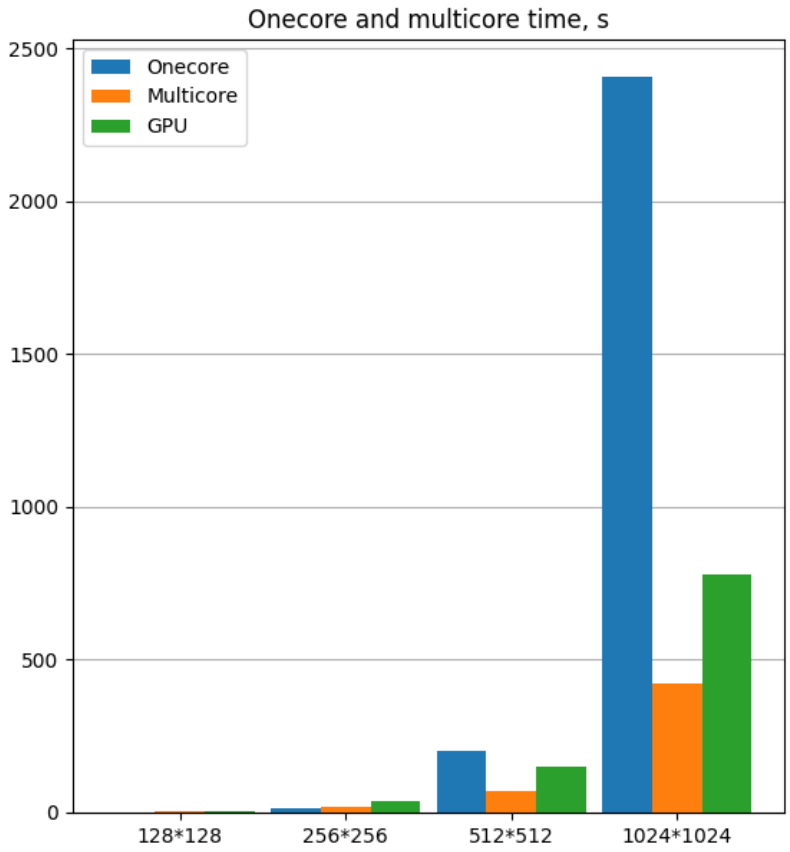
6.2 Диаграмма оптимизации (по горизонтали номер этапа; по вертикали время работы)



6.3 GPU – оптимизированный вариант

Размер сетки	Время выполнения, с	Точность	Количество опреаций
128*128	4.53	1e-06	30074
256*256	36.12	9.99e-07	102885
512*512	152.43	1e-06	339599
1024*1024	778	1.37e-06	1000000

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



8 Вывод

Для маленьких размеров сетки лучше всего подходит CPU-onecore, в больших размерах сетки лучше справился GPU-multicore

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)
{
    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
        A[size - 1][i] = A[size - 1][0] + addend; // horizontal
        A[i][0] = A[0][0] + addend; // vertical
        A[i][size - 1] = A[0][size - 1] + addend; // vertical
    }
}

```

```
45
46  void delete_2d_array(T **A, int size){
47      for (int i = 0; i < size; i++)
48          delete[] A[i];
49      delete[] A;
50  }
51
52  void calculate(int net_size = 12, int iter_max = 1e6, T accuracy = 1e-6, bool res=false) {
53      // Initialization
54      T **Anew = new T *[net_size],
55            **A = new T *[net_size];
56      for (int i = 0; i < net_size; i++)
57          A[i] = new T[net_size];
58      for (int i = 0; i < net_size; i++)
59          Anew[i] = new T[net_size];
60
61  #pragma acc enter data create(A[:net_size][:net_size], Anew[:net_size][:net_size])
62
63      // 10 20 30 20
64      initialize_array(A, net_size);
65      initialize_array(Anew, net_size);
66
67      //print_array(Anew, 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) 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                  // Average
81                  Anew[j][i] = (A[j][i + 1] + A[j][i - 1] + A[j - 1][i] + A[j + 1][i]) * 0.25;
82                  error = MAX(error, std::abs(Anew[j][i] - A[j][i]));
83              }
84
85          double** temp = A;
86          A = Anew;
87          Anew = temp;
88
89          iter++;
90  #pragma acc update host(error)
91      } while (error > accuracy && iter < iter_max);
92
93      std::cout << "iter=" << iter << ",\terror=" << error << std::endl;
94      if(res) print_array(A, net_size);
95  #pragma acc exit data delete(A[:net_size][:net_size], Anew[:net_size][:net_size], error)
96      delete_2d_array(A, net_size);
97      delete_2d_array(Anew, net_size);
98  }
99
```

```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;
}
```