

Теория параллелизма

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

pgc++

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

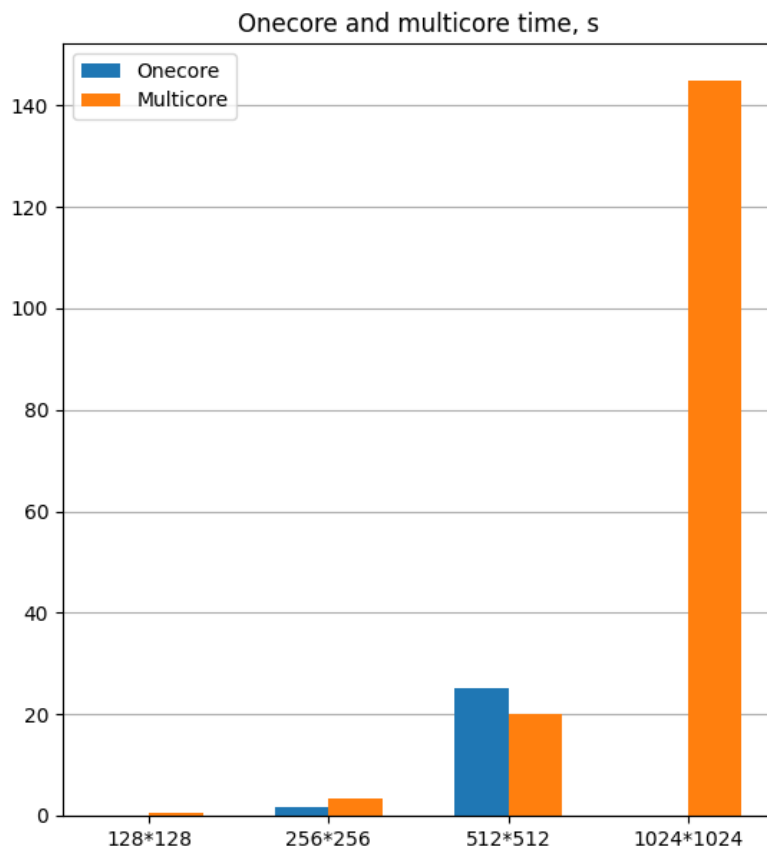
5.1 CPU-onecore

Размер сетки	Время выполнения, с	Точность	Количество операций
128*128	0.1	9.5e-07	11136
256*256	1.8	9.8e-07	37376
512*512	25	9.8e-07	120832

5.2 CPU-multicore

Размер сетки	Время выполнения, с	Точность	Количество операций
128*128	0.5	9.5e-07	11136
256*256	3.5	9.8e-07	37376
512*512	20	9.8e-07	120832
1024*1024	145	9.89e-07	365568

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

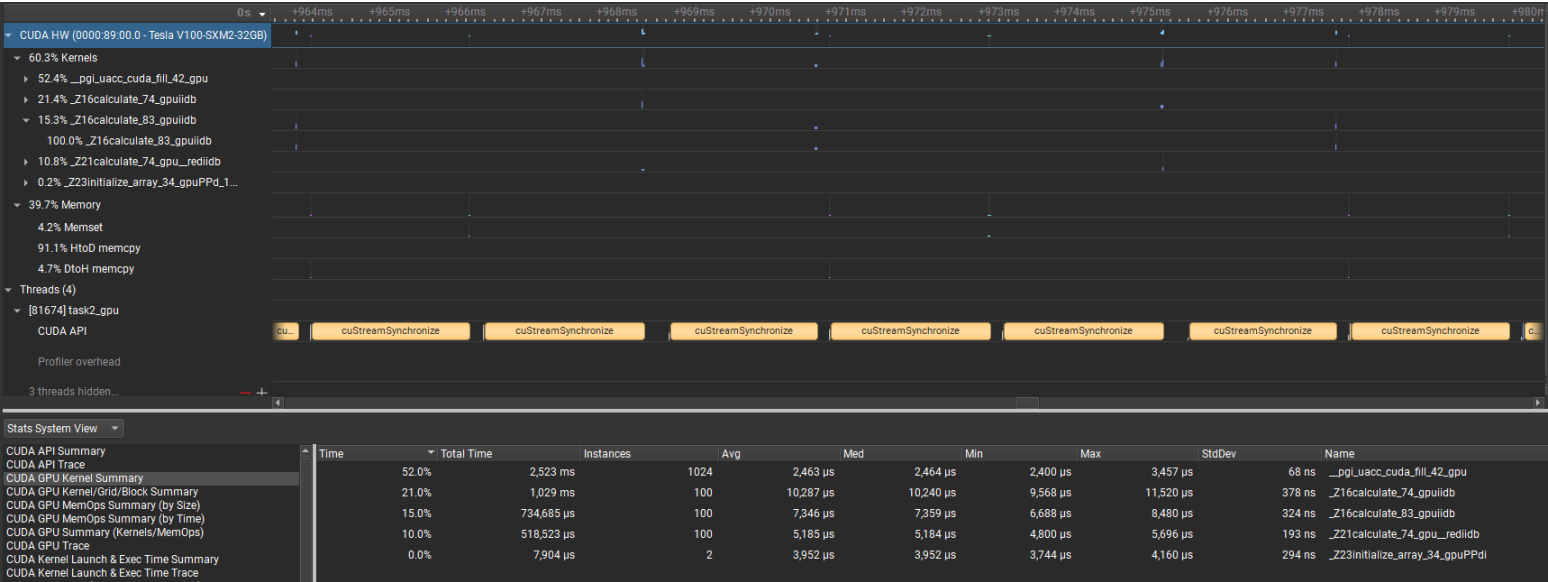


6 Выполнение на GPU

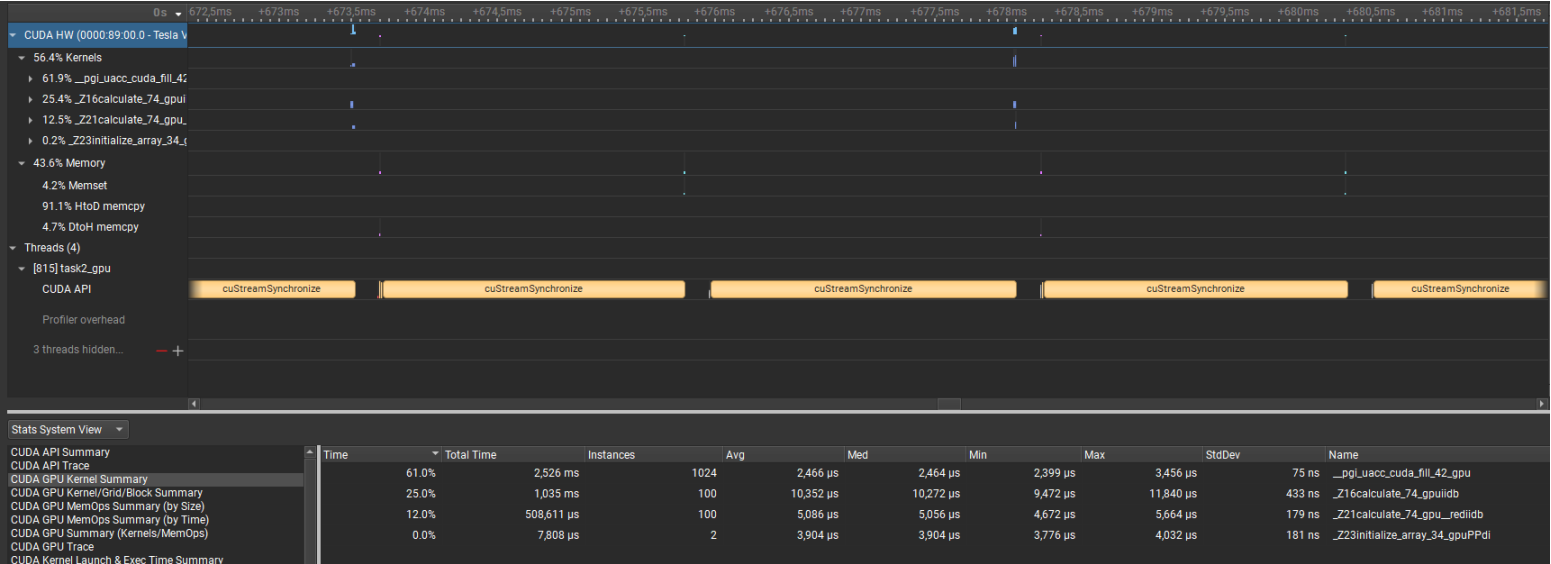
6.1 Этапы оптимизации на сетке 512*512
 (количество итераций при профилировании 100)

Этап №	Время вы- полнения, с	Точность	Количество операций	Комментарии (что было сделано)
1	0.95	0.107	100	Распараллелены циклы, reduction(max:error)
2	0.73	0.107	100	Замена копирования массива swar'ом через указатели
3	0.7	0.035	100	Изначальная инициализация массива значениями 20
4	0.5	0.036	100	Асинхронный запуск ядер
5	0.25	N/A*	100	Подсчет ошибки не каждую итера- цию
6	0.23	N/A*	100	Работа с матрицами происходит толь- ко на GPU. Операции с ошибкой ста- ли периодическими и происходят в основном, на девайсе, ошибка обнов- ляется на хосте только для проверки.

* Из-за того, что период пересчёта ошибки намного меньше размера сетки,
нет возможности определить точность на сотой итерации.



Этап 1



Этап 2

Почему инициализация значениями 20? Для начала посмотрим на финальный вывод (для простоты взята сетка 8x8):

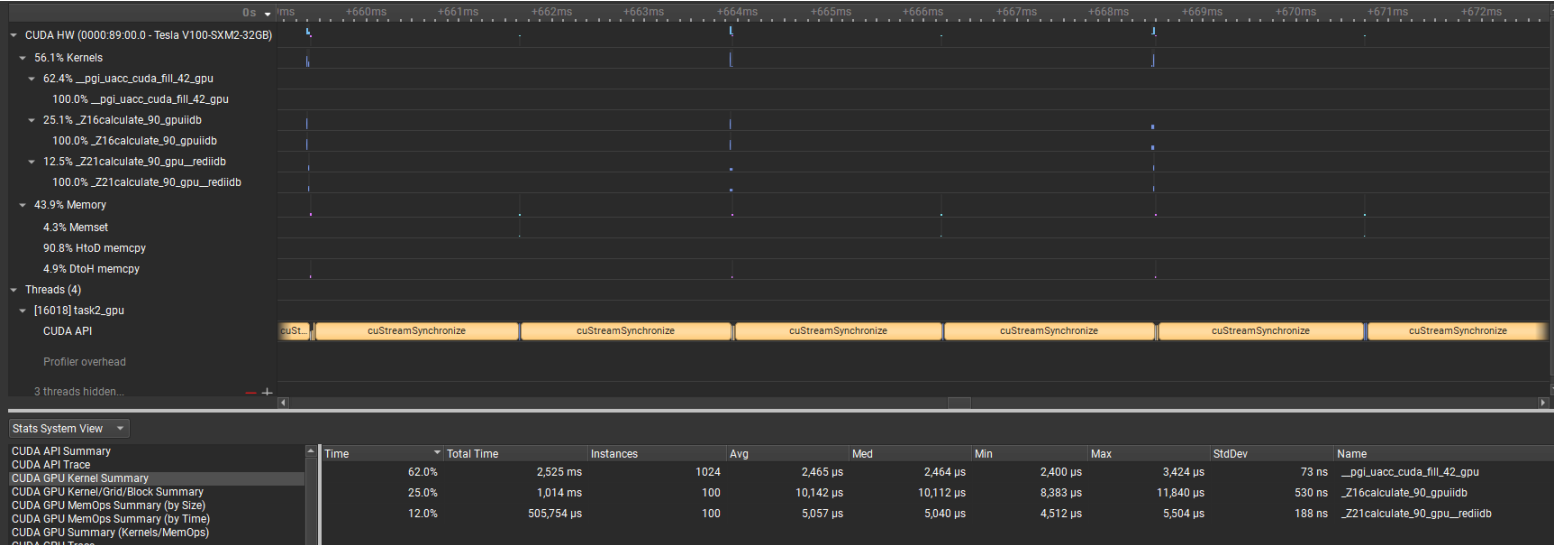
10	11.43	12.86	14.29	15.71	17.14	18.57	20
11.43	12.86	14.29	15.71	17.14	18.57	20	21.43
12.86	14.29	15.71	17.14	18.57	20	21.43	22.86
14.29	15.71	17.14	18.57	20	21.43	22.86	24.29
15.71	17.14	18.57	20	21.43	22.86	24.29	25.71
17.14	18.57	20	21.43	22.86	24.29	25.71	27.14
18.57	20	21.43	22.86	24.29	25.71	27.14	28.57
20	21.43	22.86	24.29	25.71	27.14	28.57	30

Как можно заметить значения лежат в отрезке [10, 30], значит, имеет смысл изначально инициализировать массив значениями из этого диапазона. Также стоит отметить, что модой в матрице является число 20 (а ещё и средним арифметическим границ отрезка). Значит начальная инициализация массива 20-ю существенно приблизит ответ к правильному, ещё до запуска основного алгоритма.

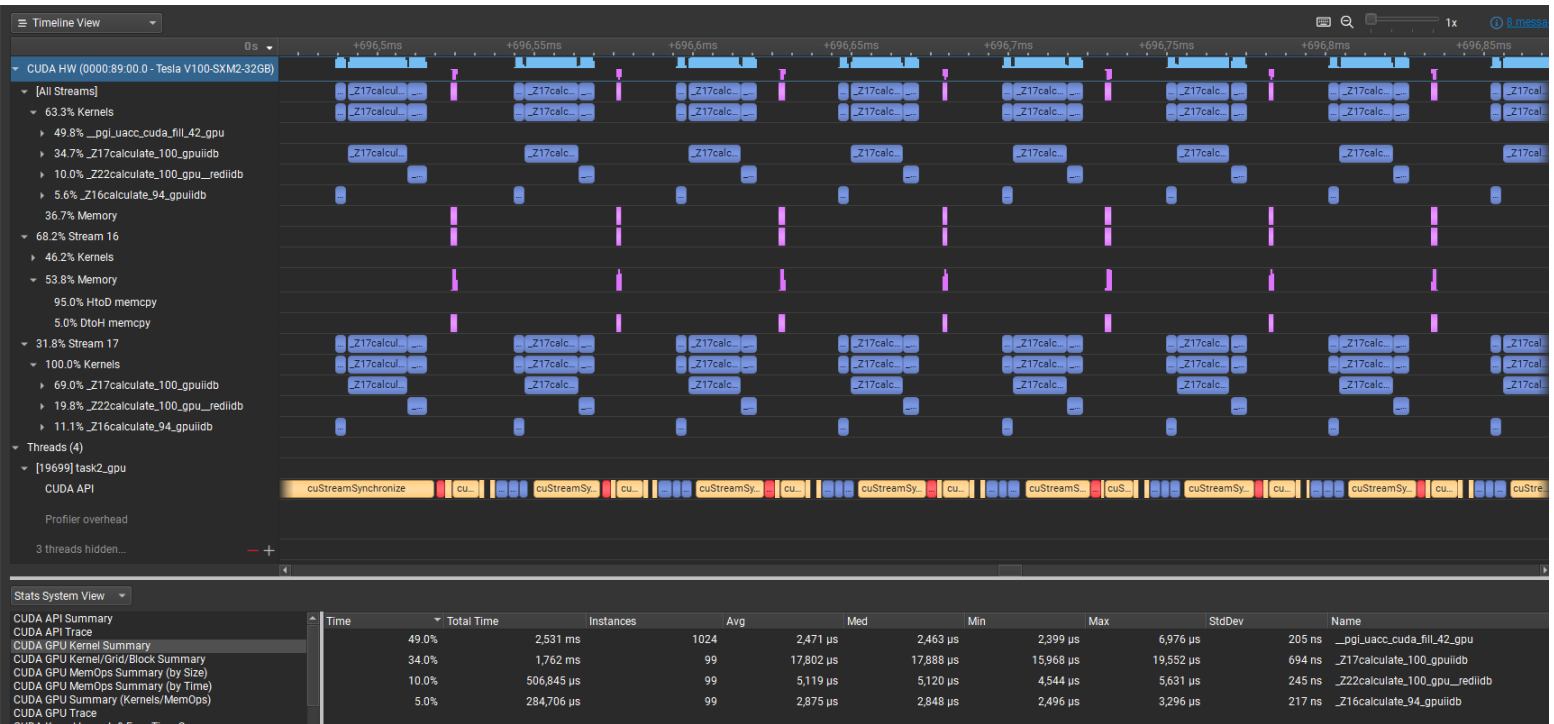
Сравнение поведения программы на сетке 128x128.

Инициализация	Время выполнения, с	Точность	Количество операций
Нет	1.1	1e-06	11073
Да	0.4	9.995e-07	30074

Как видно в таблице, изначальная инициализация дает хороший прирост производительности.



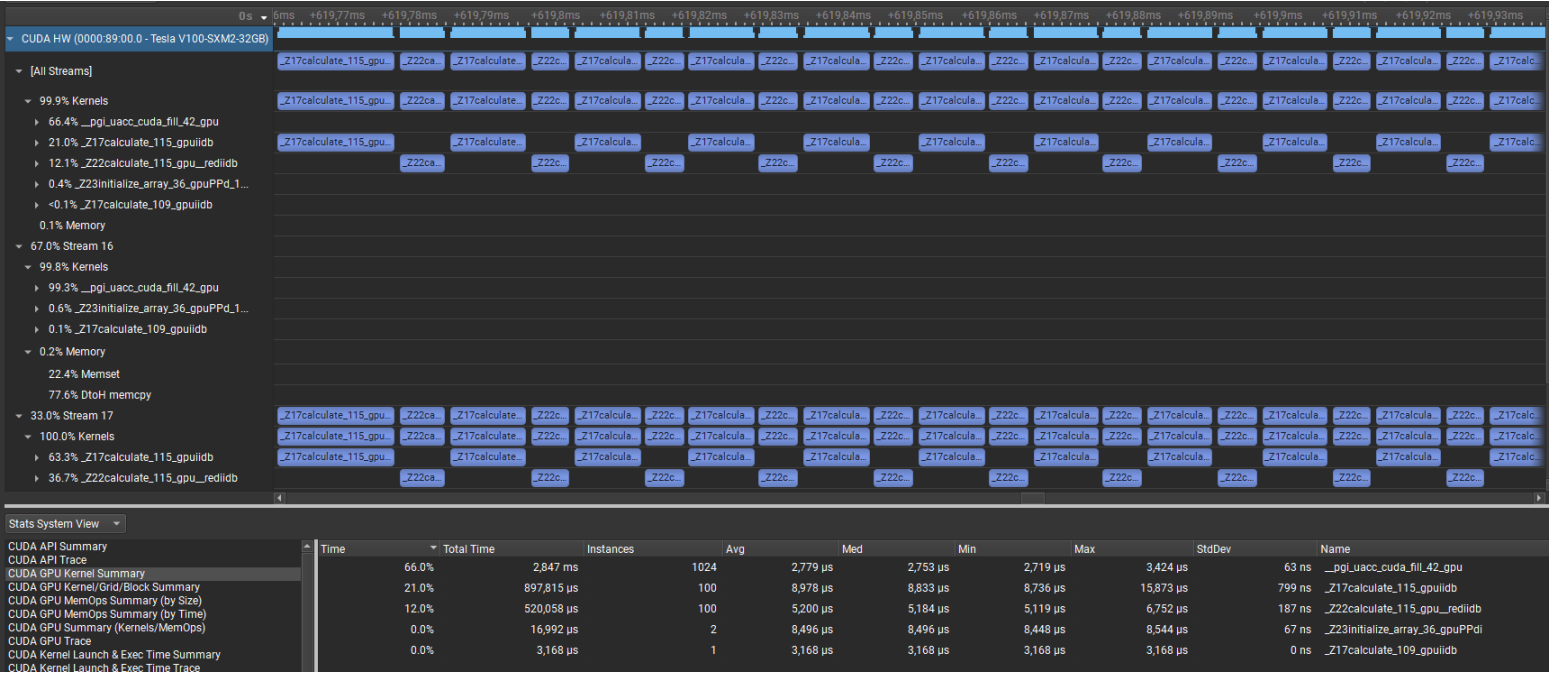
Этап 3



Этап 4

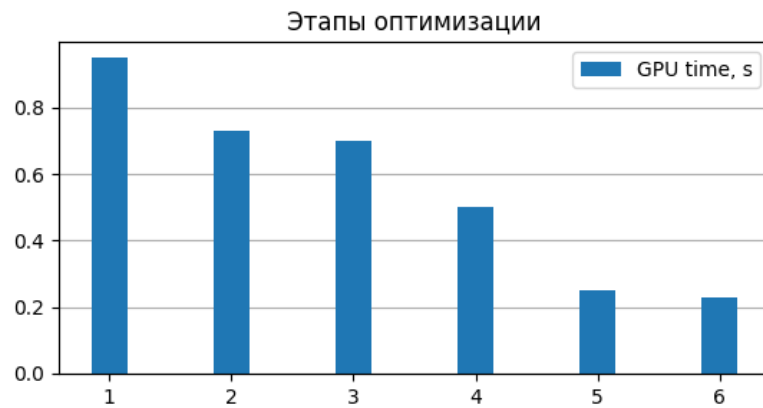


Этап 5



Этап 6

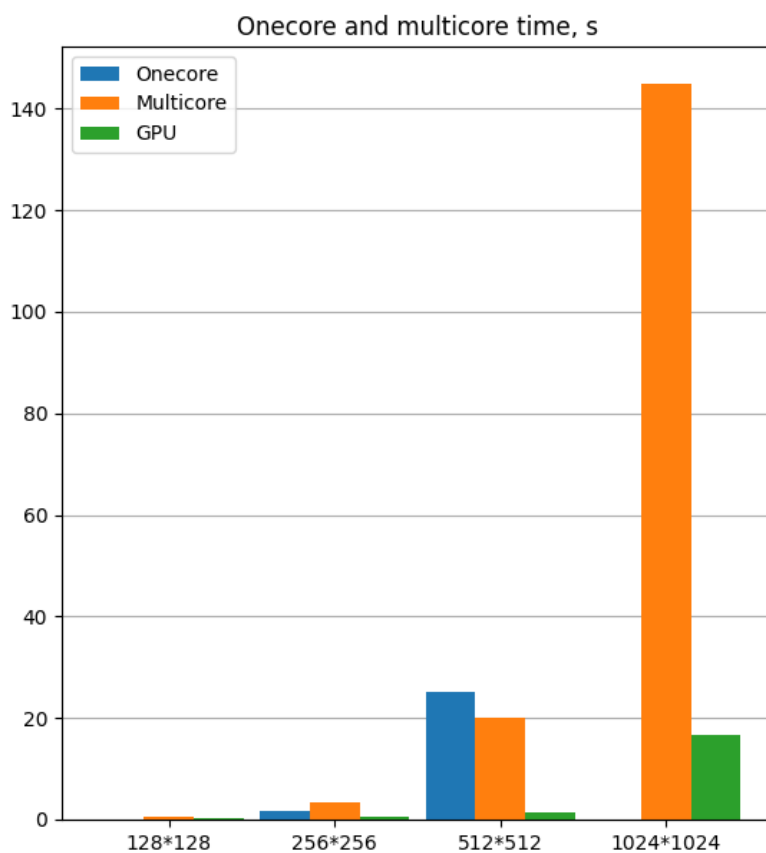
6.2 Диаграмма оптимизации (по горизонтали номер этапа; по вертикали время работы)



6.3 GPU – оптимизированный вариант

Размер сетки	Время выполнения, с	Точность	Количество опреаций
128*128	0.3	9.5e-07	11136
256*256	0.5	9.8e-07	37376
512*512	1.5	9.8e-07	120832
1024*1024	16.7	9.9e-07	365568

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



8 Вывод

При грамотном использовании ресурсов GPU можно достичь ускорения работы программы в десятки раз. GPU все еще уступает CPU в маленьких сетках (≤ 128), однако отставание не критично.

9 Приложение

9.1 Ссылка на GitHub

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

```
1  #include <chrono>
2  #include <cmath>
3  #include <iostream>
4
5  #ifdef _FLOAT
6      #define T float
7      #define MAX std::fmaxf
8      #define STOD std::stof
9  #else
10     #define T double
11     #define MAX std::fmax
12     #define STOD std::stod
13 #endif
14
15 // Вывести значения двумерного массива
16 void print_array(T **A, int size)
17 {
18     for (int i = 0; i < size; ++i)
19     {
20         for (int j = 0; j < size; ++j)
21         {
22             // Значение с GPU
23             #pragma acc kernels present(A)
24             printf("%.2f\t", A[i][j]);
25         }
26         std::cout << std::endl;
27     }
28     std::cout << std::endl;
29 }
30
```

```

31 // Инициализация матрицы, чтобы подготовить ее к основному алгоритму
32 void initialize_array(T **A, int size)
33 {
34     // Инициализируется матрица на GPU
35     #pragma acc parallel present(A)
36     {
37         // Заполнение углов матрицы значениями
38         A[0][0] = 10.0;
39         A[0][size - 1] = 20.0;
40         A[size - 1][size - 1] = 30.0;
41         A[size - 1][0] = 20.0;
42
43         // Заполнение периметра матрицы
44         T step = 10.0 / (size - 1);
45         #pragma acc loop independent
46         for (int i = 1; i < size - 1; ++i)
47         {
48             T addend = step * i;
49             A[0][i] = A[0][0] + addend; // horizontal
50             A[size - 1][i] = A[size - 1][0] + addend; // horizontal
51             A[i][0] = A[0][0] + addend; // vertical
52             A[i][size - 1] = A[0][size - 1] + addend; // vertical
53         }
54
55         // Заполнение 20-ю, чтобы сократить количество операций в основном алгоритме
56         #pragma acc loop independent collapse(2)
57         for (int i = 1; i < size - 1; ++i)
58             for (int j = 1; j < size - 1; ++j)
59                 A[i][j] = 20.0;
60     }
61 }
62
63 // Очистка динамической памяти, выделенной под двумерную матрицу
64 void delete_2d_array(T **A, int size)
65 {
66     for (int i = 0; i < size; i++)
67         delete[] A[i];
68     delete[] A;
69 }
70
71 // Основной алгоритм
72 void calculate(int net_size = 128, int iter_max = 1e6, T accuracy = 1e-6, bool res = false)
73 {
74     // Создание 2-х двумерных матриц, одна будет считаться на основе другой
75     T **Anew = new T *[net_size],
76     **A = new T *[net_size];
77     for (int i = 0; i < net_size; i++)
78     {
79         A[i] = new T[net_size];
80         Anew[i] = new T[net_size];
81     }
82
83     #pragma acc enter data create(A[:net_size][:net_size], Anew[:net_size][:net_size])
84
85     // Инициализация матриц
86     initialize_array(A, net_size);
87     initialize_array(Anew, net_size);
88
89     // Текущая ошибка
90     T error = 0;
91     // Счетчик итераций
92     int iter;
93     // Указатель для swap
94     T **temp;
95     // Флаг обновления ошибки на хосте для обработки условием цикла
96     bool update_flag = true;
97

```

```

97
98 #pragma acc data copy(error)
99 {
100     for (iter = 0; iter < iter_max; ++iter)
101     {
102         // Сокращение количества обращений к CPU. Больше сетка - реже стоит сверять значения.
103         update_flag = !(iter % net_size);
104
105         if (update_flag)
106         {
107             // загрузка ошибки на GPU
108             #pragma acc kernels
109             error = 0;
110         }
111
112         // Распараллелить циклы с редукцией и запустить асинхронные ядра
113         #pragma acc kernels loop independent collapse(2) reduction(max : error) present(A, Anew) async(1)
114         for (int i = 1; i < net_size - 1; i++)
115             for (int j = 1; j < net_size - 1; j++)
116             {
117                 Anew[i][j] = (A[i + 1][j] + A[i - 1][j] + A[i][j - 1] + A[i][j + 1]) * 0.25;
118                 // Пересчитать ошибку
119                 if (update_flag)
120                     error = std::max(error, std::abs(Anew[i][j] - A[i][j]));
121             }
122
123         // swap(A, Anew)
124         temp = A, A = Anew, Anew = temp;
125         // Проверить ошибку
126         if (update_flag)
127         {
128             // Синхронизация и обновление ошибки на хосте
129             #pragma acc update host(error) wait(1)
130             // Если ошибка не превышает точность, прекратить выполнение цикла
131             if (error <= accuracy)
132                 break;
133         }
134     }
135 }
136 #pragma acc update host(error)
137 std::cout.precision(2);
138 if (res)
139     print_array(A, net_size);
140 std::cout << "iter=" << iter << ",\terror=" << error << std::endl;
141
142 #pragma acc exit data delete (A[:net_size][:net_size], Anew[:net_size][:net_size])
143 delete_2d_array(A, net_size);
144 delete_2d_array(Anew, net_size);
145 }
146
147 int main(int argc, char *argv[])
148 {
149     auto begin_main = std::chrono::steady_clock::now();
150     int net_size = 128, iter_max = (int)1e6;
151     T accuracy = 1e-6;
152     bool res = false;
153     for (int arg = 1; arg < argc; arg++)
154     {
155         std::string str = argv[arg];
156         if (!str.compare("-res"))
157             res = true;
158         else
159         {
160             arg++;
161             if (!str.compare("-a"))
162                 accuracy = stod(argv[arg]);
163             else if (!str.compare("-i"))
164                 iter_max = (int)std::stoi(argv[arg]);
165             else if (!str.compare("-s"))
166                 net_size = std::stoi(argv[arg]);
167             else
168             {
169                 std::cout << "Wrong args!";
170                 return -1;
171             }
172         }
173     }
174     calculate(net_size, iter_max, accuracy, res);
175     auto end_main = std::chrono::steady_clock::now();
176     int time_spent_main = std::chrono::duration_cast<std::chrono::milliseconds>(end_main - begin_main).count();
177     std::cout << "The elapsed time is:\nmain\t\t" << time_spent_main << " ms\n";
178     return 0;
179 }
180

```