Отчёт по заданию "Знакомство с OpenACC"

Александр Грищенко

1 Компилятор и программа

Так как код программы написан на языке C++, использовались компиляторы g++ для исполнения на CPU и pgc++ для исполнения на GPU. На сервере программа имеет название "task_1" с суффиксами "cpu", "cpu_multicore", "gpu_double", "gpu_float", для запуска на CPU в одноядерном режиме, CPU в многоядерном режиме, GPU с массивом типа double, GPU с массивом типа double соответственно.

2 Вывод программы

Программа считает сумму элементов массива. По свойству синуса эта сумма должна быть равна 0. значит, можно использовать полученное значение как абсолютную погрешность.

2.1 CPU (1 ядро)

```
Sum = -6.7691627509593251e-10
The elapsed time is:
main                    186 ms
        fill_array      168 ms
        sum_array       13537 us
```

Общее время работы - 186 мкс.
Суммарное время циклов - 182 мкс

2.2 CPU (multicore)

```
Sum = 2.4738255888223648e-10
The elapsed time is:
main                    155 ms
```

```
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
OpenACC (excl):   94.27%  21.067ms         1  21.067ms  21.067ms  21.067ms  acc_compute_construct@iostream:10
                   5.73%  1.2794ms         1  1.2794ms  1.2794ms  1.2794ms  acc_compute_construct@iostream:24
```

Общее время работы - 155 мкс.
Суммарное время циклов - 22 мкс

2.3 GPU (float)

```
Sum = 0
The elapsed time is:
main                    511 ms
```

```
          Type  Time(%)      Time     Calls       Avg       Min       Max  Name
GPU activities:   35.49%  122.50us         1  122.50us  122.50us  122.50us  fill_array_10_gpu(float*, int)
                  34.88%  120.42us         1  120.42us  120.42us  120.42us  sum_array_24_gpu(float*, int)
                  27.83%  96.063us         1  96.063us  96.063us  96.063us  sum_array_24_gpu__red(float*, int)
```

Общее время работы - 511 мкс.
Суммарное время циклов - 339 мкс

2.4 GPU (double)

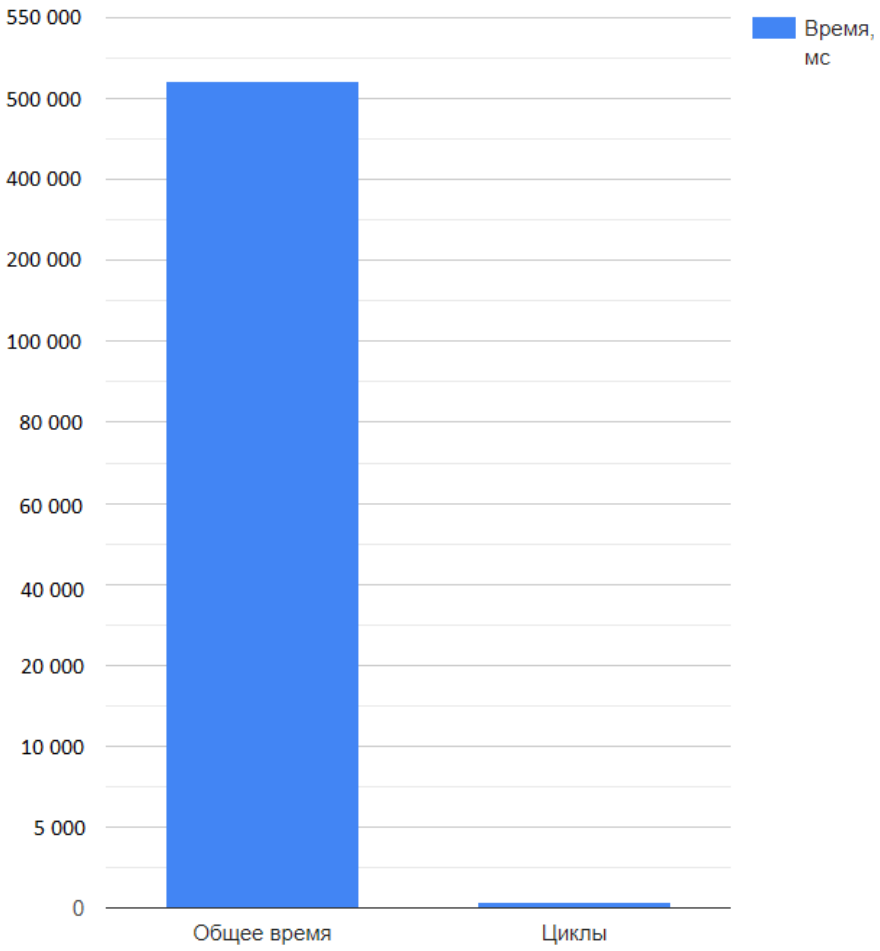
```
Sum = -3.1263880373444408e-12
The elapsed time is:
main                    521 ms
```

```
          Type  Time(%)      Time     Calls       Avg       Min       Max  Name
GPU activities:   36.37%  131.46us         1  131.46us  131.46us  131.46us  sum_array_24_gpu(double*, int)
                  34.48%  124.61us         1  124.61us  124.61us  124.61us  fill_array_10_gpu(double*, int)
                  27.39%  99.007us         1  99.007us  99.007us  99.007us  sum_array_24_gpu__red(double*, int)
```

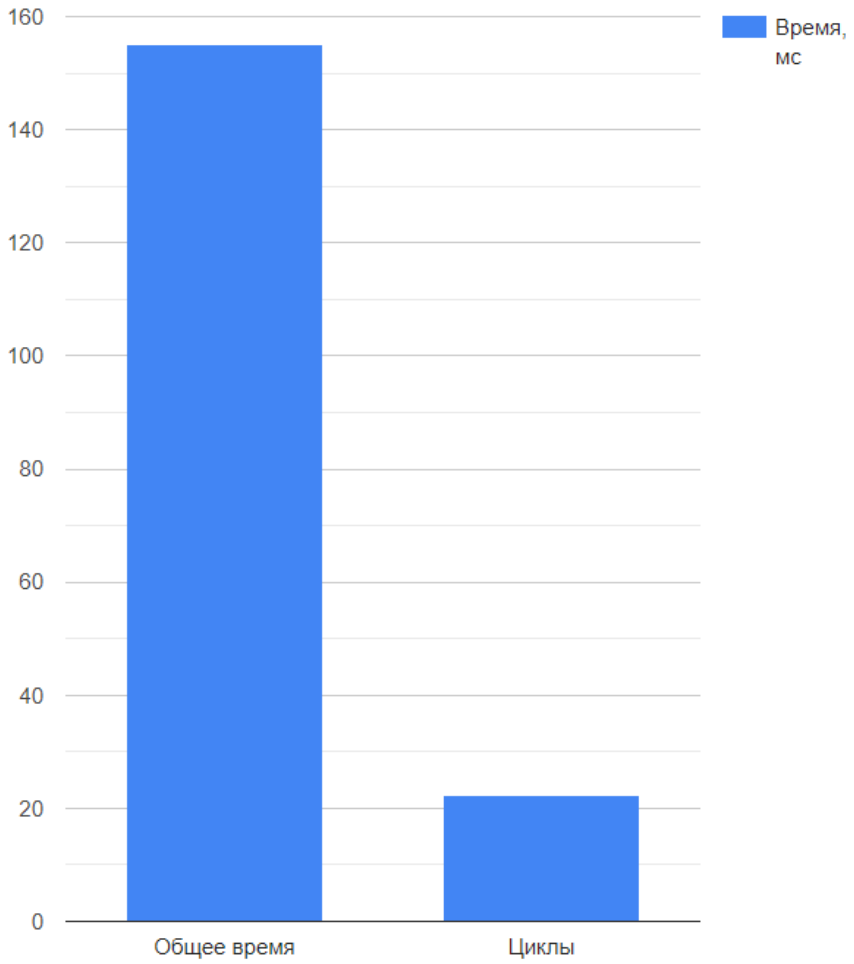
Общее время работы - 521 мкс.
Суммарное время циклов - 355 мкс

3 Диаграмма

Ниже на дигараммах представлено сравнение суммарного времени работы всех циклов и общего времени работы на GPU и CPU multicore



Сравнение суммарного времени работы всех циклов и общего времени работы (GPU)



Сравнение суммарного времени работы всех циклов и общего времени работы (CPU multicore)

4 Выводы

Несмотря на быстрое выполнение циклов на GPU, значительная часть времени работы программы ушла на передачу данных. Поэтому для этой задачи лучше всего подходит CPU.

5 Код программы

Код одинаков для CPU и GPU, поскольку компилятор g++ игнорирует непонятные ему директивы.

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <cmath>
#include <chrono>

void fill_array(double *arr, int size)
{
        double step = 2 * M_PI / size;
#pragma acc data copyin(step)
        {
#pragma acc parallel loop present(arr[:size])
                for (int i = 0; i < size; i++)
                {
                        double x = i * step;
                        arr[i] = sin(x);
                }
        }
}

double sum_array(double *arr, int size)
{
        double sum = 0;
#pragma acc data copy(sum)
        {
#pragma acc parallel loop reduction(+: sum)
                for (int i = 0; i < size; i++)
                {
                        sum += arr[i];
                }
        }
        return sum;
}

void print_array(double *arr, int size)
{
```

```
#pragma acc exit data copyout(arr[:size])
        std::cout.precision(2);
        for (int i = 0; i < size; i += 1e5)
        {
                std::cout << arr[i] << " ";
        }
        printf("\n");
}

int main()
{
        auto begin_main = std::chrono::steady_clock::now();
        const int size = 1e7;
        double *arr = new double[size];

        auto begin_fill_array = std::chrono::steady_clock::now();
#pragma acc enter data create(arr[:size])
        fill_array(arr, size);
        auto end_fill_array = std::chrono::steady_clock::now();

        auto begin_sum_array = std::chrono::steady_clock::now();
        double sum = sum_array(arr, size);
        auto end_sum_array = std::chrono::steady_clock::now();
        // print_array(arr, size); //to debug the contents of an array
        std::cout.precision(17);
        std::cout << "Sum = " << sum << std::endl;
        delete [] arr;
#pragma acc exit data delete(arr [:size])
        auto end_main = std::chrono::steady_clock::now();
        int time_spent_main = std::chrono::duration_cast<std::chrono::milliseconds>
              (end_main - begin_main).count();
        int time_spent_fill_array = std::chrono::duration_cast<std::chrono::milliseconds>
              (end_fill_array - begin_fill_array).count();
        int time_spent_sum_array = std::chrono::duration_cast<std::chrono::microseconds>
              (end_sum_array - begin_sum_array).count();
        std::cout << "The elapsed time is:\nmain\t\t" << time_spent_main << " ms\n"
                << "\tfill_array\t" << time_spent_fill_array << " ms\n"
                << "\tsum_array\t" << time_spent_sum_array << " us\n";
        return 0;
}
```