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                    177 ms
        fill_array      162 ms
        sum_array       12802 us
```

Общее время работы - 177 мс.
Суммарное время циклов - 175 мс

2.2 CPU (multicore)

```
Sum = 1.1641532182693481e-10
The elapsed time is:
main                    162 ms
```

```
          Type  Time(%)      Time     Calls       Avg       Min       Max  Name
OpenACC (excl):   97.53%  27.800ms         1  27.800ms  27.800ms  27.800ms  acc_compute_construct@iostream:20
                   2.47%  703.22us         1  703.22us  703.22us  703.22us  acc_compute_construct@iostream:34
```

Общее время работы - 162 мс.
Суммарное время циклов - 28 мс

2.3 GPU (float)

```
Sum = 0.02472686767578125
The elapsed time is:
main                    512 ms
```

```
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
GPU activities:    3.43%  115.68us         1  115.68us  115.68us  115.68us  sum_array_34_gpu(float*, int)
                   3.42%  115.30us         1  115.30us  115.30us  115.30us  fill_array_20_gpu(float*, int)
                   2.68%  90.463us         1  90.463us  90.463us  90.463us  sum_array_34_gpu__red(float*, int)
```

Общее время работы - 512 мс.
Суммарное время циклов - 322 мкс

2.4 GPU (double)

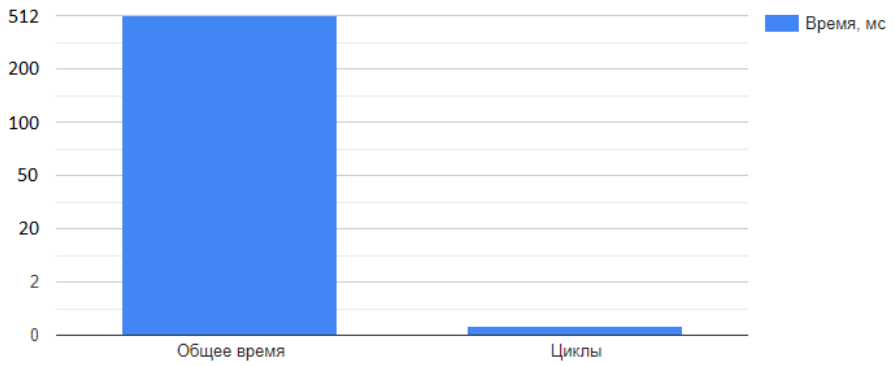
```
Sum = -3.1263880373444408e-12
The elapsed time is:
main                    536 ms
```

```
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
GPU activities:    1.00%  125.34us         1  125.34us  125.34us  125.34us  sum_array_34_gpu(double*, int)
                   1.00%  125.09us         1  125.09us  125.09us  125.09us  fill_array_20_gpu(double*, int)
                   0.79%  99.423us         1  99.423us  99.423us  99.423us  sum_array_34_gpu__red(double*, int)
```

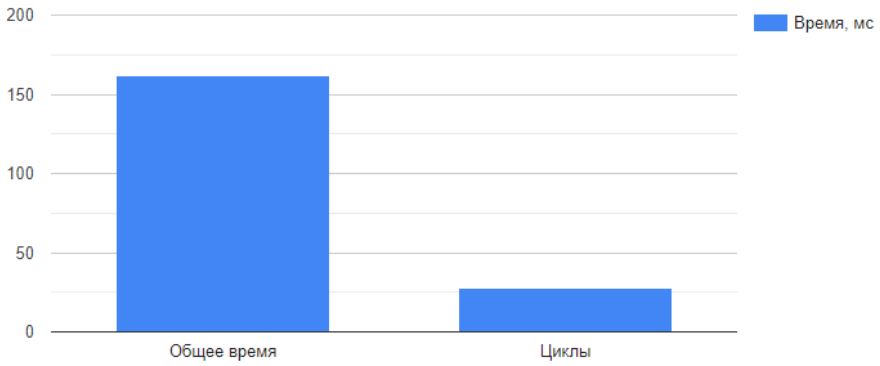
Общее время работы - 536 мс.
Суммарное время циклов - 350 мкс

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>

// #define _FLOAT

#ifdef _FLOAT
#define T float
#define SIN sinf
#else
#define T double
#define SIN sin
#endif

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

T sum_array(T *arr, int size)
{
        T sum = 0;
#pragma acc data copy(sum)
        {
#pragma acc parallel loop present(arr[:size]) reduction(+: sum)
```

```
                for (int i = 0; i < size; i++)
                        sum += arr[i];
        }
        return sum;
}

void print_array(T *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;
        T *arr = new T[size];

        auto begin_fill_array = std::chrono::steady_clock::now();
        fill_array(arr, size);
        auto end_fill_array = std::chrono::steady_clock::now();

#pragma acc enter data create(arr[:size])
        auto begin_sum_array = std::chrono::steady_clock::now();
        T 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\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;
}
```