**Отчет по первому заданию**

[**Тероия Параллелизма**](https://classroom.google.com/u/1/c/NTg0Nzg0MTE5Mzgy)

**“Знакомство с OpenACC”**

Над отчетом работал:  
Емельянов Алексей Алексеевич 21932

Новосибирск 2023

Оглавление

[**Цель работы** 3](#_Toc127725656)

[**Работа** 4](#_Toc127725657)

[**Код программы** 6](#_Toc127725658)

[**Результат работы** 8](#_Toc127725659)

## **Цель работы**

Заполнить на графическом процессоре массив типа float/double значениями синуса (один период на всю длину массива). Размер массива - 10^7 элементов. Для заполненного массива на графическом процессоре посчитать сумму всех элементов массива. Сравнить со значением, вычисленном на центральном процессоре. Сравнить разультат для массивов  float и double.  
  
При сборке программы для исполнения на GPU использовать компилятор pgcc/pgc++ с ключами  
`-acc -Minfo=accel`. Необходимо  
разобраться с выводом о распараллеливании кода.  
  
Произвести профилирование программы установив переменную окружения PGI\_ACC\_TIME=1. Необходимо  
понять сколько времени тратится на вычисление и сколько времени на передачу  
данных.   
  
Результаты работы, какие знания о работе программы на GPU были получены представить в виде отчета.  
  
Возможные дополнительные вопросы:  
  
\* как выполняется параллельное суммирование,   
  
\* почему в программе два цикла, а ядер (по выводу профилировщика) три  
  
\* почему сумма всех элементов не равна нулю

## **Работа**

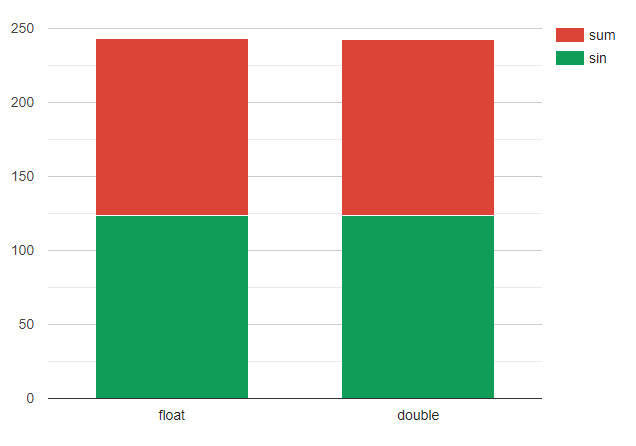
Компилятор для создания кода для видеокарты: pgc++ с флагами –fast –acc

Компилятор для создания кода для процессора: pgc++ с флагами –fast –Mconcur=all / -Minfo=accel

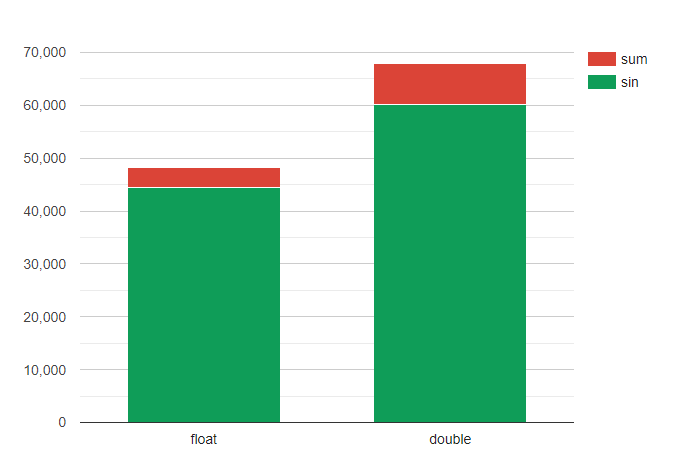
Просмотр данных по времени nvprof файл

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Тип данных** | **Общее время выполнения** | **Время выполнения цикла на сумирвоание** | **Время выполнения цикла на заполнение** | **результаты** |
| **gpu** | **Float** | **3499801 us** | **119.36 us** | **123.90 us** | **-0.024231** |
| **gpu** | **Double** | **3344809 us** | **119,01 us** | **123.71 us** | **-8.2423e-12** |
| **cpu one core** | **Float** | **48201 us** | **3651 us** | **44550 us** | **-0.024231** |
| **cpu one core** | **Double** | **67892 us** | **7818 us** | **60074 us** | **-8.2423e-12** |
| **cpu multi core** | **Float** | **23556 us** | **4824 us** | **18732 us** | **-0.024231** |
| **cpu multi core** | **Double** | **29544 us** | **10836 us** | **18708 us** | **-8.2423e-12** |

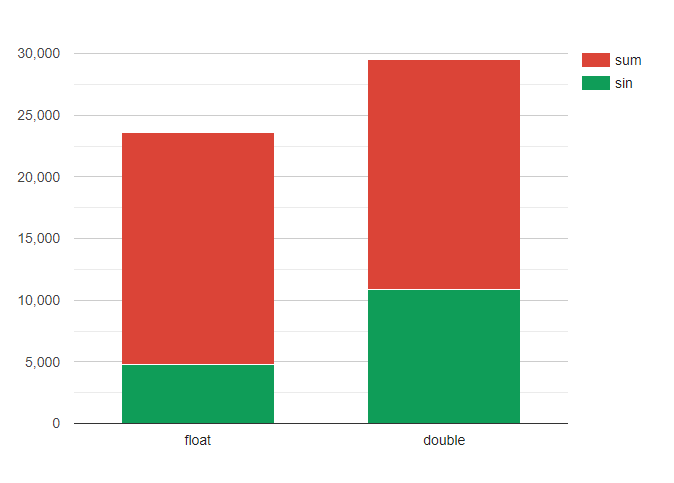
**График GPU**

****

**График CPU one**

****

**График CPU multi**

****

## **Код программы**

#include <iostream>

#include <cmath>

#include <chrono>

#include <iomanip>

#include <time.h>

const unsigned int N = 10000000;

float arr[N];

float pi = 3.1415926535;

float x;

int main()

{

    auto start = std::chrono::high\_resolution\_clock::now();

    float sum = 0;

    auto startCreate = std::chrono::high\_resolution\_clock::now();

#pragma acc enter data create(arr [0:N], sum)

#pragma acc parallel loop present(arr [0:N])

    for (unsigned int i = 0; i < N; i++)

    {

        arr[i] = sin(2 \* pi \* i / N);

    }

    auto elapsed = std::chrono::high\_resolution\_clock::now() - startCreate;

    long microseconds = std::chrono::duration\_cast<std::chrono::microseconds>(

                                 elapsed)

                                 .count();

    std::cout << microseconds << " create time\n";

    auto startSum = std::chrono::high\_resolution\_clock::now();

#pragma acc parallel loop present(arr [0:N], sum) reduction(+ \

                                                            : sum)

    for (unsigned int i = 0; i < N; i++)

    {

        sum += arr[i];

    }

#pragma acc exit data delete (arr [0:N])copyout(sum)

    elapsed = std::chrono::high\_resolution\_clock::now() - startSum;

    microseconds = std::chrono::duration\_cast<std::chrono::microseconds>(

                                 elapsed)

                                 .count();

    std::cout << microseconds << " sum time\n";

    elapsed = std::chrono::high\_resolution\_clock::now() - start;

    microseconds = std::chrono::duration\_cast<std::chrono::microseconds>(

                                 elapsed)

                                 .count();

    std::cout << microseconds << " microseconds\n";

    std::cout << sum << " sum\n";

    return 0;

}

#include <iostream>

#include <cmath>

#include <chrono>

#include <iomanip>

#include <time.h>

const unsigned int N = 10000000;

double arr[N];

double pi = 3.1415926535;

double x;

int main()

{

    auto start = std::chrono::high\_resolution\_clock::now();

    double sum = 0;

    auto startCreate = std::chrono::high\_resolution\_clock::now();

#pragma acc enter data create(arr [0:N], sum)

#pragma acc parallel loop present(arr [0:N])

    for (unsigned int i = 0; i < N; i++)

    {

        arr[i] = sin(2 \* pi \* i / N);

    }

    auto elapsed = std::chrono::high\_resolution\_clock::now() - startCreate;

    long microseconds = std::chrono::duration\_cast<std::chrono::microseconds>(

                            elapsed)

                            .count();

    std::cout << microseconds << " create time\n";

    auto startSum = std::chrono::high\_resolution\_clock::now();

#pragma acc parallel loop present(arr [0:N], sum) reduction(+ \

                                                            : sum)

    for (unsigned int i = 0; i < N; i++)

    {

        sum += arr[i];

    }

#pragma acc exit data delete (arr [0:N])copyout(sum)

    elapsed = std::chrono::high\_resolution\_clock::now() - startSum;

    microseconds = std::chrono::duration\_cast<std::chrono::microseconds>(

                       elapsed)

                       .count();

    std::cout << microseconds << " sum time\n";

    elapsed = std::chrono::high\_resolution\_clock::now() - start;

    microseconds = std::chrono::duration\_cast<std::chrono::microseconds>(

                       elapsed)

                       .count();

    std::cout << microseconds << " microseconds\n";

    std::cout << sum << " sum\n";

    return 0;

}

## **Результат работы**

**Показателе продемонстрировали следующий результат, если нам нужно работать с небольшим количеством данных, оптимизированный вариант будет на CPU.**