**Отчёт**

Программа Знакомство с OpenACC

Компилятор PGCC

Подготовил Солопов Илья

Группа 21932

[Время выполнения циклов 2](#_Toc127657672)

[Время выполнения циклов на GPU 2](#_Toc127657673)

[Время выполнения циклов на CPU 2](#_Toc127657674)

[Время работы программ 3](#_Toc127657675)

[Общее время выполнения на GPU 3](#_Toc127657676)

[Общее время выполнения на CPU 3](#_Toc127657677)

[Точность расчётов 4](#_Toc127657678)

[Диаграммы 5](#_Toc127657679)

[Коды программы 6](#_Toc127657680)

[Код программы на GPU 6](#_Toc127657681)

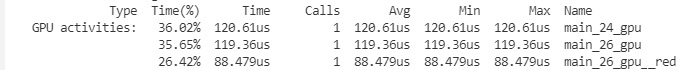
[Код программы на CPU 7](#_Toc127657682)

[Выбор оптимального решения 8](#_Toc127657683)

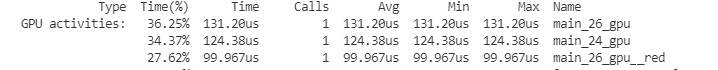
Время выполнения циклов

Время выполнения циклов на GPU

С использованием типа данных float были получены следующие показатели.



С использованием типа данных double были получены следующие показатели



Время выполнения циклов на CPU

При многопоточном исполнении программы с использованием типа данных float были получены следующие показатели.



При многопоточном исполнении программы с использованием типа данных double были получены следующие показатели.



Утилита nvprof не поддерживает сбор информации о работе CPU в однопоточном режиме. Из данных, представленных выше, можно заметить, что большую часть времени выполнения программы занимает исполнение циклов. Исходя из этого, время для кода, выполняющегося на одном потоке CPU, можно принять близким ко времени работы всей программы, которое представлено в следующем пункте.

Время работы программ

Общее время выполнения на GPU

Общее время исполнения программы на графическом процессоре с использованием типа данных float составило 3426.560767 миллисекунд, а с использованием double 3434.754238 миллисекунд.

Общее время выполнения на CPU

|  |  |  |
| --- | --- | --- |
| **Многопоточность** | **Затраченное время, *мс* (float)** | **Затраченное время, *мс* (double)** |
| Нет | 276.890000 | 267.774000 |
| Да | 25.218737 | 27.805645 |

Точность расчётов

|  |  |  |
| --- | --- | --- |
| **Процессор** | **Результат вычислений (float)** | **Результат вычислений (double)** |
| GPU | -0.0357971191406250000000000 | 0.0000250675636124242373626 |
| CPU | -0.0140503961592912673950195 | 0.0000250055376995731890814 |

Диаграммы

Коды программы

Код программы на GPU

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define N 10000000

#define PI 3.14159

//поддержка double

#define LF\_SUP

#ifdef LF\_SUP

#define TYPE double

#else

#define TYPE float

#endif

int main(){

    TYPE \*arr = (TYPE\*)malloc(sizeof(TYPE) \* N), sum =0.0,

    tmp=(PI\*2/N);

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

    #pragma acc kernels

    for (int i = 0; i < N; ++i){

        arr[i] = sin(tmp \* i);

    }

    #pragma acc parallel loop reduction(+:sum)

    for (int i = 0; i < N; ++i) {

        sum += arr[i];

    }

    #pragma acc exit data copyout(sum)

    printf("%-32.25lf\n", sum);

    free(arr);

    return 0;

}

Код программы на CPU

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#define N 10000000

#define PI 3.14159

//поддержка double

#define LF\_SUP

#ifdef LF\_SUP

#define TYPE double

#else

#define TYPE float

#endif

int main(){

    struct timespec start, end;

    clock\_gettime(CLOCK\_REALTIME, &start);

    TYPE \*arr = (TYPE\*)malloc(sizeof(TYPE) \* N), sum =0.0,

    tmp=(PI\*2/N);

    for (int i = 0; i < N; ++i){

        arr[i] = sin(tmp \* i);

    }

    for (int i = 0; i < N; ++i) {

        sum += arr[i];

    }

    printf("%-32.25lf\n", sum);

    free(arr);

    clock\_gettime(CLOCK\_REALTIME, &end);

    double time = ((end.tv\_sec - start.tv\_sec) + (end.tv\_nsec - start.tv\_nsec) / 1000000000.0)\*1000;

    printf("%lf ms\n", time);

    return 0;

}

Выбор оптимального решения

Полученные результаты говорят нам о том, что время исполнения циклов на графическом процессоре ничтожно мало по сравнению со временем, которое тратится на загрузку данных и аллокацию памяти. В то же время, центральный процессор в любом из режимов работает быстрее (на 1 и 2 порядка в однопотоке и многопотоке соответственно) графического. Исходя из полученных данных, оптимальным решением будет являться использование CPU в многопоточном режиме.