Федеральное государственное бюджетное образовательное учреждения высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра ПМиК

Расчётно-графическое задание

По дисциплине: «Программирование графических процессоров»

Выполнил:
студент III курса
ИВТ, гр. ИП-713
Михеев Н.А.

Проверил:
Ассистент кафедры ПМиК
Нужнов А.В.

Новосибирск, 2020 г.

Оглавление

1. Постановка задачи

Провести сравнительный анализ производительности программ, реализующих алгоритмы линейной алгебры с использованием библиотек Thrust, cuBLAS и «сырого» CUDA C кода.

Дополнительно: сравнить так же с реализацией CUDA на языке Python – PyCUDA.

2. Описание работы

Для проведения анализа были разработаны программы вычислений линейной алгебры SAXPY на двух языках программирования – C и Python, сняты необходимые метрики.

Для сравнения на языке C были реализованы три вида алгоритма: с использованием «сырого» CUDA C кода, Thrust – библиотека, основанная на использовании шаблонов языка C++, а так же cuBLAS - реализация интерфейса программирования приложений для создания библиотек, выполняющих основные операции линейной алгебры BLAS (Basic Linear Algebra Subprograms) для CUDA. На языке Python был реализован «сырой» CUDA код с использованием библиотеки PyCUDA, которая по сути реализуют подготовленный код ядра на языке C.

3. Результаты тестирования

Тестировались программы на компьютере со следующими характеристиками: CPU – Intel Core i7 – 8550u, GPU – Nvidia GeForce MX150 с 2 GB памяти, ОС – Linux Manjaro – 20.0.1

Версия CUDA compiler – 10.2, версия Python – 3.8, библиотека PyCUDA – 2019.1.2

Все вычисления производились на 32 битных числах с плавающей точкой, так как графические ускорители специализированы на таких вычислениях. Так же все вычисления производились по 100 раз и бралось среднее значение времени вычисления.

Размер векторов	C CUDA	PyCUDA	Thrust	cuBlas
1024	0,0083	0,0331	0,0099	0,0098
2048	0,0086	0,0342	0,0102	0,0099
4096	0,0091	0,0377	0,01	0,01
8192	0,0102	0,0378	0,0102	0,0104
16384	0,0127	0,0405	0,0109	0,0118
32768	0,0193	0,0435	0,0123	0,0149
65536	0,029	0,0531	0,0165	0,0204
131072	0,0508	0,0747	0,047	0,0431
262144	0,0969	0,132	0,0844	0,0802
524288	0,1783	0,223	0,159	0,1556
1048576	0,342	0,4043	0,3082	0,3062

Рис.1 – Полученные данные.

Из данных видно, что Thrust и cuBlas оказались самыми быстрыми реализациями с очень похожими результатами, далее идет сырой C CUDA код и за ним следует PyCUDA. Необходимо отметить, что на сравнительно небольших размерах векторов PyCUDA сильно уступает в скорости всем остальным реализациям, но с увеличением размера векторов, примерно с 131072, начинает наверстывать в скорости и ближе к концу практически сравнивается с сырой C CUDA реализацией.

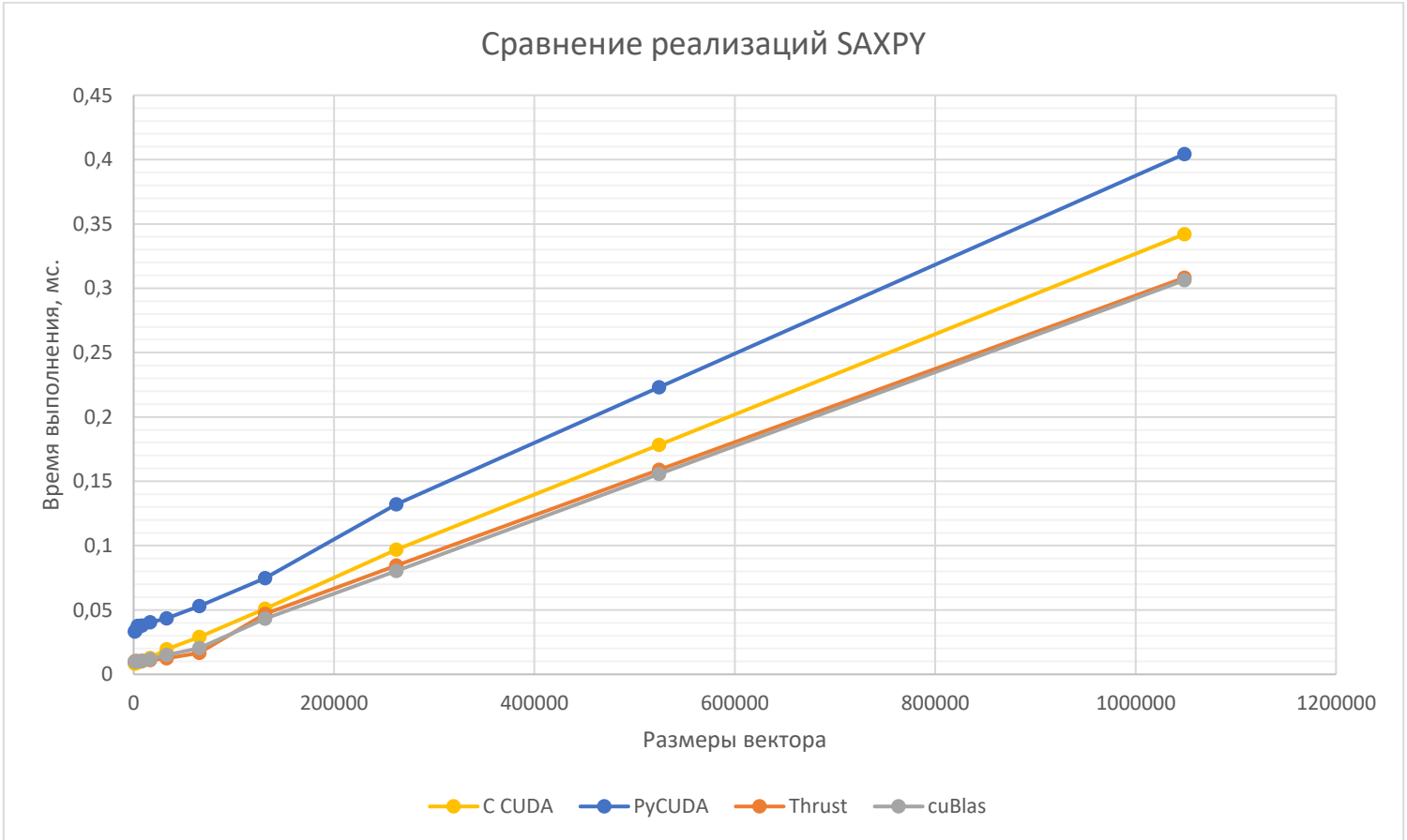

Рис.2 – график времени для алгоритмов.

На графике наблюдается значительное увеличение времени выполнения всех алгоритмов, при размере векторов более 65536. Еще до этой «точки» реализация Thrust на немного, но все же оказывается быстрее, чем cuBlas, но после преодоления 65536 элементов скорости выполнения алгоритмов сравниваются. PyCUDA на протяжении всего графика оказывается медленнее, но все-таки с увеличением размера векторов начинается сближаться с остальными реализациями.

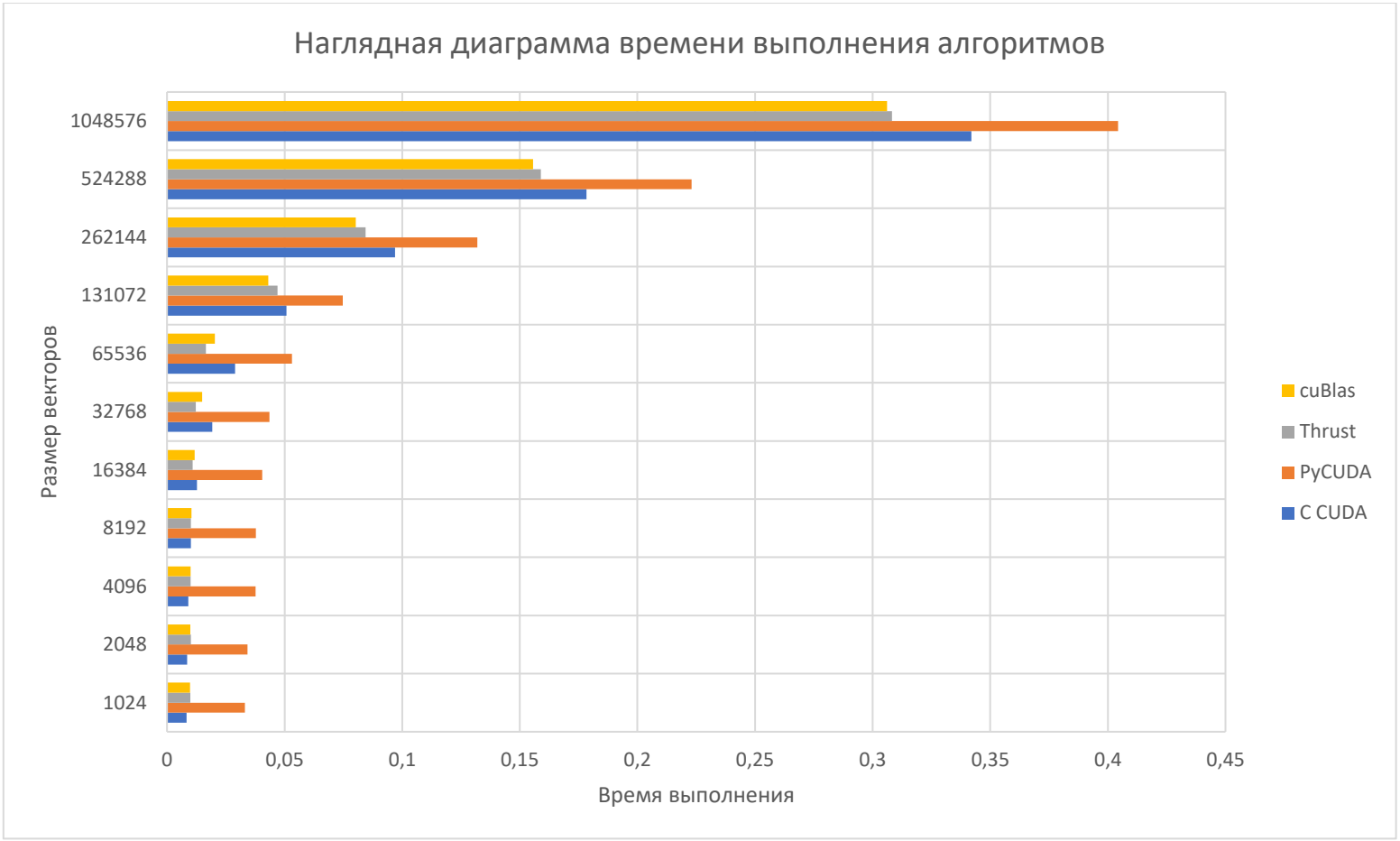

Рис.3 – более наглядная диаграмма времени выполнения алгоритмов линейной алгебры.

На наглядной диаграмме заметен паритет на малых размерах векторов реализаций на языке C. И заметны все различия в скорости, отмеченные выше.

Скриншоты работы программ:

```
/home/lolimpo/PycharmProjects/venv/bin/python
Using device: GeForce MX150
N: 1048576
Number of test tries: 100
PyCUDA avg. time: 0.4379916191101074

Process finished with exit code 0
```

Рис.4 – результат работы программы на PyCUDA.

```
/home/lolimpo/Projects/CUDA_Course/Course_Work/cmake-build-debug/Course_Work
Number of test tries: 100
Native CUDA avg. time: 0.366713ms.
cuBlas avg. time: 0.319212ms.
Thrust avg. time: 0.315246ms.

Process finished with exit code 0
```

Рис.4 – результат работы программы на языке C.

4. Вывод

Результаты тестирования показали, что использование специализированных библиотек для реализации задач линейной алгебры дают значительный прирост в производительности при большом количестве данных и также облегчает разработку в целом. «Сырой» CUDA код показал себя как и ожидалось – достаточно быстро и стабильно. Использование PyCUDA не совсем оправданно для «чистых» вычислений из-за видимой просадки в производительности, но так же как и библиотеки Thrust и cuBlas не сложен в разработке, хоть и для работы все же требуется написать код ядра на C, но больше не требуется сложных конструкций для его запуска как в «сыром» C. Выглядит это все как чистый питон за исключением локальной вставки C-кода.

5. Приложение: листинг программ

Код программы на языке Python версии 3.8:

```python
import time

import numpy as np
import pycuda.autoinit
from pycuda.compiler import SourceModule
import pycuda.driver as cuda

print(f"Using device: {cuda.Context.get_device().name()}")


def kernel_function():
    saxpy_kernel = SourceModule("""
    __global__ void saxpy(int n, float a, float *x, float *y)
    {
        int index = blockIdx.x * blockDim.x + threadIdx.x;
        int stride = blockDim.x * gridDim.x;
        for(auto i = index; i < n; i += stride)
            y[index] += a * x[index];
    }
    """)
```

```
    return saxpy_kernel.get_function("saxpy")


if __name__ == '__main__':
    testTry = 100
    pycuda_time = 0
    N = 2 ** 20
    print(f"N: {N}")
    for i in range(testTry):
        XVAL = np.float32(10 ** np.random.rand())
        YVAL = np.float32(10 ** np.random.rand())
        alpha = np.float32(14)

        x = np.zeros((N), dtype=np.float32) + np.float32(XVAL)
        y = np.zeros((N), dtype=np.float32) + np.float32(YVAL)

        dev_x = cuda.mem_alloc(x.nbytes)
        dev_y = cuda.mem_alloc(y.nbytes)

        cuda.memcpy_htod(dev_x, x)
        cuda.memcpy_htod(dev_y, y)

        saxpy = kernel_function()

        start = time.time()
        saxpy(np.int32(N), alpha, dev_x, dev_y, block=(32, 1, 1), grid=(int(np.ce
il(N/32)), 1, 1))
        cuda.Context.synchronize()
        elapsed = time.time() - start
        pycuda_time += elapsed * 1000
        #print(f"PyCUDA SAXPY: {elapsed * 100} ms.")

    print(f"Number of test tries: {testTry}")
    print(f"PyCUDA avg. time: {pycuda_time / testTry}")
```

Код программы на языке C:

```
#include <iostream>
#include <random>
#include <cublas_v2.h>
#include <thrust/device_vector.h>

__global__
void saxpy(int n, float a, float *x, float *y)
{
    unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    unsigned int stride = blockDim.x * gridDim.x;

    for (auto i = index; i < n; i += stride) {
        y[i] += a * x[i];
    }
```

```
}

struct saxpy_functor
{
  const float a_;
  saxpy_functor(float a) : a_(a) {}
  __host__ __device__
  float operator() (const float &x, const float &y) const {
      return a_ * x + y;
  }
};

int main ()
{
    cudaEvent_t start, finish;
    cudaEventCreate(&start);
    cudaEventCreate(&finish);
    float elapsedTime;
    float cudaTime = 0, cublasTime = 0, thrustTime = 0;
    const int N = (1 << 20);
    int blockSize = 32;
    int numBlocks = N / blockSize;

    float *host_x, *host_y;
    float *dev_x, *dev_y;

    int testTry = 100;
    for(int i = 0; i < testTry; i++)
    {
        std::random_device rd;
        std::uniform_int_distribution<int> uid(0, 10000);
        const float XVAL = uid(rd);
        const float YVAL = uid(rd);
        const float alpha = 14.0f;

//Native CUDA Method
        host_x = new float[N];
        host_y = new float[N];
        for (int i = 0; i < N; i++)
        {
            host_x[i] = XVAL;
            host_y[i] = YVAL;
        }

        cudaMalloc ((void **) &dev_x, N * sizeof (float));
        cudaMalloc ((void **) &dev_y, N * sizeof (float));
        cudaMemcpy (dev_x, host_x, N * sizeof (float), cudaMemcpyHostToDevice);
        cudaMemcpy (dev_y, host_y, N * sizeof (float), cudaMemcpyHostToDevice);
        cudaDeviceSynchronize ();

        cudaEventRecord (start);
```

```
        saxpy<<<numBlocks, blockSize>>> (N, alpha, dev_x, dev_y);
        cudaDeviceSynchronize ();
        cudaEventRecord (finish);
        cudaEventSynchronize (finish);
        cudaEventElapsedTime (&elapsedTime, start, finish);
        cudaTime += elapsedTime;
        //std::cout << "Native CUDA:" << elapsedTime << "ms.\n";

        cudaFree (dev_x);
        cudaFree (dev_y);
        delete[] host_x;
        delete[] host_y;
//cuBLAS Method
        cublasHandle_t handle;
        cublasCreate (&handle);

        host_x = new float[N];
        host_y = new float[N];
        for (int i = 0; i < N; i++)
        {
            host_x[i] = XVAL;
            host_y[i] = YVAL;
        }

        cudaMalloc ((void **) &dev_x, N * sizeof (float));
        cudaMalloc ((void **) &dev_y, N * sizeof (float));
        cublasSetVector (N, sizeof (host_x[0]), host_x, 1, dev_x, 1);
        cublasSetVector (N, sizeof (host_y[0]), host_x, 1, dev_y, 1);
        cudaDeviceSynchronize ();

        cudaEventRecord (start);
        cublasSaxpy (handle, N, &alpha, dev_x, 1, dev_y, 1);
        cudaDeviceSynchronize ();
        cudaEventRecord (finish);
        cudaEventSynchronize (finish);
        cudaEventElapsedTime (&elapsedTime, start, finish);
        cublasTime += elapsedTime;
        //std::cout << "CUBLAS SAXPY: " << elapsedTime << "ms.\n";

        cudaFree (dev_x);
        cudaFree (dev_y);
        delete[] host_x;
        delete[] host_y;
//Thrust Method
        thrust::device_vector<float> X (N, XVAL);
        thrust::device_vector<float> Y (N, YVAL);

        cudaEventRecord (start);
        thrust::transform (X.begin (), X.end (), Y.begin (), Y.begin (), saxpy_fu
nctor (alpha));
        cudaDeviceSynchronize ();
```

```
        cudaEventRecord (finish);
        cudaEventSynchronize (finish);
        cudaEventElapsedTime (&elapsedTime, start, finish);
        thrustTime += elapsedTime;
        //std::cout << "Thrust SAXPY: " << elapsedTime << "ms.\n";
    }
    std::cout << "Number of test tries: " << testTry << "\n";
    std::cout << "Native CUDA avg. time: " << cudaTime / (float)testTry << "ms.\n
";
    std::cout << "cuBlas avg. time: " << cublasTime / (float)testTry << "ms.\n";
    std::cout << "Thrust avg. time: " << thrustTime / (float)testTry << "ms.\n";
    return 0;
}
```