

Выполнила: Белоусова Е., ИП-911

Задание

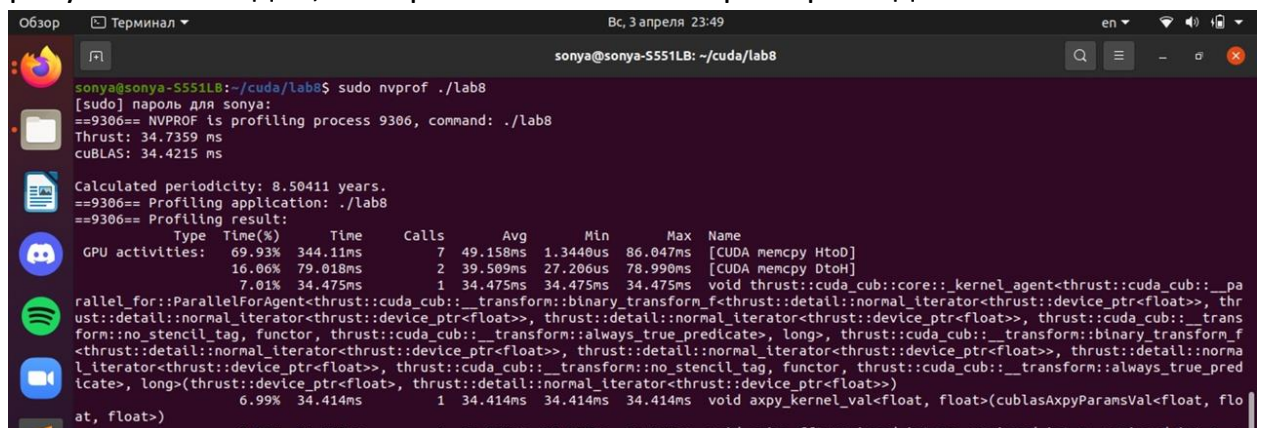
- сравните производительность программ, реализующих `saхру` на основе библиотек `thrust` и `cuBLAS`.

- выявите периодичность солнечной активности, опираясь на данные наблюдений о числе Вольфа, представленные по адресу:

http://www.gaoran.ru/database/csa/daily_wolf_r.html Цель: научиться пользоваться прикладными библиотеками CUDA.

Описание работы программы

Сравним производительность программ, реализующих `saхру` на основе библиотек `thrust` и `cuBLAS`. Размер исходного массива данных 2^{23} . По результатам видно, что время выполнения примерно одинаковое.



```
sonya@sonya-S551LB: ~/cuda/lab8
[sudo] пароль для sonya:
==9306== NVPROF is profiling process 9306, command: ./lab8
Thrust: 34.7359 ms
cuBLAS: 34.4215 ms

Calculated periodicity: 8.50411 years.
==9306== Profiling application: ./lab8
==9306== Profiling result:
Type Time(%) Time Calls Avg Min Max Name
GPU activities: 69.93% 344.11ms 7 49.158ms 1.3440us 86.047ms [CUDA memcpy HtoD]
16.06% 79.018ms 2 39.509ms 27.206us 78.990ms [CUDA memcpy DtoH]
7.01% 34.475ms 1 34.475ms 34.475ms 34.475ms void thrust::cuda_cub::core::kernel_agent<thrust::cuda_cub::pa
rallel_for::ParallelForAgent<thrust::cuda_cub::__transform::binary_transform_f<thrust::detail::normal_iterator<thrust::device_ptr<float>>, thr
ust::detail::normal_iterator<thrust::device_ptr<float>>, thrust::detail::normal_iterator<thrust::device_ptr<float>>, thrust::cuda_cub::__trans
form::no_stencil_tag, functor, thrust::cuda_cub::__transform::always_true_predicate>, long>, thrust::cuda_cub::__transform::binary_transform_f
<thrust::detail::normal_iterator<thrust::device_ptr<float>>, thrust::detail::normal_iterator<thrust::device_ptr<float>>, thrust::detail::norma
l_iterator<thrust::device_ptr<float>>, thrust::cuda_cub::__transform::no_stencil_tag, functor, thrust::cuda_cub::__transform::always_true_pred
icate>, long>(thrust::device_ptr<float>, thrust::detail::normal_iterator<thrust::device_ptr<float>>)
6.99% 34.414ms 1 34.414ms 34.414ms void axpy_kernel_val<float, float>(cublasAxpParamsVal<float, flo
at, float>)
```

Выявим периодичность солнечной активности, опираясь на данные наблюдений о числе Вольфа. Коэффициенты Фурье сами по себе трудно интерпретировать. Более значимой мерой коэффициентов является их величина в квадрате, которая является мерой мощности. Поскольку половина коэффициентов повторяется по величине, вам нужно вычислить мощность только для одной половины коэффициентов. Постройте график спектра мощности в зависимости от частоты, измеряемой в циклах в год.

Максимальная активность солнечных пятен наблюдается реже одного раза в год. Для представления циклической активности, которую легче интерпретировать, постройте график мощности в зависимости от периода, измеряемого в годах за цикл. График показывает, что активность солнечных пятен достигает пика примерно раз в 11 лет.

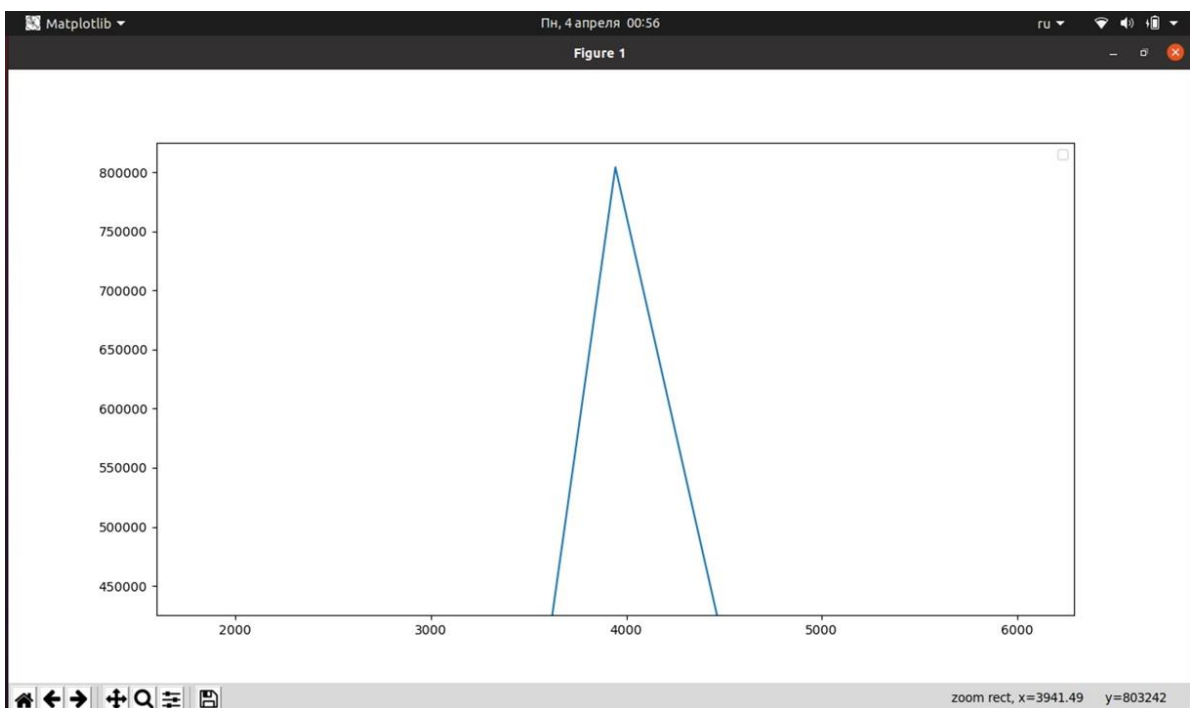
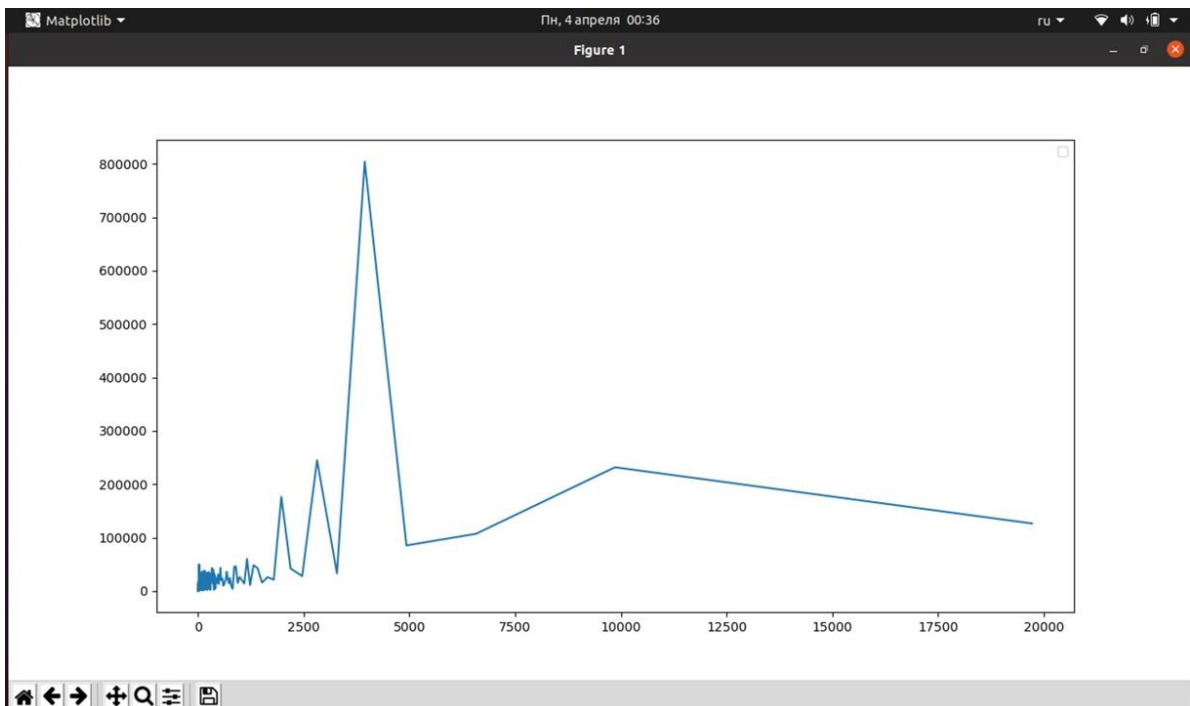
<https://linuxgazette.net/115/andreasen.html>

<https://arxiv.org/vc/arxiv/papers/1310/1310.6876v1.pdf>

<https://www.mathworks.com/help/matlab/math/using-fft.html>

```
sonya@sonya-S551LB: /cuda/lab8$ ./lab8
Thrust: 35.0519 ms
cuBLAS: 34.6435 ms

Calculated periodicity: 10.8066 years.
sonya@sonya-S551LB:~/cuda/lab8$ python3 1.py
```



Листинг

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <cuFFT.h>
#include <stdio.h>
```

```
#include <malloc.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <cctype>
#include <list>
#include <stdlib.h>
#include <ctime>

#include <thrust/device_vector.h>
#include <thrust/fill.h>
#include <thrust/host_vector.h>
#include <thrust/sequence.h>
#include <thrust/transform.h>

#include <cublas.h>
#include <cublas_v2.h>

#define NX 64
#define BATCH 1
#define pi 3.141592
#define SZ (1<<25)
#define ALPHA 3.0f

using namespace std;
```

```

struct functor {
    const float koef;
    functor(float _koef) : koef(_koef) {}
    __host__ __device__ float operator()(float x, float y) { return koef * x + y; }
};

void saxpy(float _koef, thrust::device_vector<float> &x, thrust::device_vector<float> &y)
{
    functor func(_koef);
    thrust::transform(x.begin(), x.end(), y.begin(), y.begin(), func);
}

int main()
{
    srand(time(NULL));
    cudaEvent_t start, stop;
    float *x = new float[SZ];
    float *y = new float[SZ];
    thrust::host_vector<float> h1(SZ);
    thrust::host_vector<float> h2(SZ);
    float *dev_x;
    float *dev_y;

    cublasHandle_t handle;
    cublasCreate(&handle);

    float time;
    float alpha = ALPHA;

    cudaMalloc(&dev_x, SZ*sizeof(float));

```

```

cudaMalloc(&dev_y, SZ*sizeof(float));

cudaEventCreate(&start);
cudaEventCreate(&stop);

for(int i=0;i<SZ;i++){
    h1[i]=rand()%1024;
    h2[i]=rand()%1024;
}

thrust::device_vector<float> g1 = h1;
thrust::device_vector<float> g2 = h2;

cudaEventRecord(start, 0);

saxpy(alpha, g1, g2);

cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&time, start, stop);

cout << "Thrust: " << time << " ms\n";

for (int i = 0; i < SZ; i++)
{
    x[i] = h1[i];
    y[i] = h2[i];
}

cublasInit();

```

```

cublasSetVector(SZ, sizeof(x[0]), x, 1, dev_x, 1);
cublasSetVector(SZ, sizeof(y[0]), y, 1, dev_y, 1);

cudaEventRecord(start, 0);

cublasSaxpy(handle, SZ, &alpha, dev_x, 1, dev_y, 1);

cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&time, start, stop);

cublasGetVector(SZ, sizeof(y[0]), dev_y, 1, y, 1);
cublasShutdown();

cout << "cuBLAS: " << time << " ms\n\n";

```

```

string line;
string buffer;
cufftHandle plan;
cufftComplex *cpu_data;
cufftComplex *gpu_data;
vector<string> numline;
vector<float> DATA;
vector<float> freq;
vector<float> power;
ifstream in;
cufftComplex *data_h;
int j=0;
for(int i=1938;i<=1991;i++){

```

```

string path = string("data/") + to_string(i) + string(".dat");
in.open(path);
if(!in.is_open()){
    cout << "Unknown Error" << endl;
    return -1;
}
while(getline(in,line)){
    buffer="";
    line+=" ";
    for(int k=0;k<line.size();k++){
        if(line[k]!=' '){
            buffer+=line[k];
        }else{
            if(buffer!="")numline.push_back(buffer);
            buffer="";
        }
    }
    if (numline.size() != 0)
    {
        if(numline[2]=="999"){
            numline[2] = to_string(DATA.back());
        }
        DATA.push_back(stoi(numline[2]));
        numline.clear();
    }
    j++;
}

in.close();
}

int N = DATA.size();

```

```

cudaMalloc((void**)&gpu_data, sizeof(cufftComplex) * N * BATCH);
data_h = (cufftComplex*)calloc(N, sizeof(cufftComplex));
cpu_data = new cufftComplex[N * BATCH];
for (int i = 0; i < N * BATCH; i++)
{
    cpu_data[i].x = DATA[i];
    cpu_data[i].y = 0.0f;
}

cudaMemcpy(gpu_data, cpu_data, sizeof(cufftComplex) * N * BATCH,
cudaMemcpyHostToDevice);

if (cufftPlan1d(&plan, N * BATCH, CUFFT_C2C, BATCH) != CUFFT_SUCCESS)
{
    cerr << "ERROR cufftPlan1d" << endl;
    return -1;
}

if (cufftExecC2C(plan, gpu_data, gpu_data, CUFFT_FORWARD) != CUFFT_SUCCESS)
{
    cerr << "ERROR cufftPlan1d" << endl;
    return -1;
}

if (cudaDeviceSynchronize() != cudaSuccess)
{
    cerr << "ERROR cufftPlan1d" << endl;
    return -1;
}

cudaMemcpy(data_h, gpu_data, N * sizeof(cufftComplex), cudaMemcpyDeviceToHost);

ofstream of("x.txt");
ofstream of1("y.txt");
power.resize(N/2+1);
for(int i=1;i<=N/2;i++){
    power[i]=sqrt(data_h[i].x*data_h[i].x+data_h[i].y*data_h[i].y);
    of<<i<<endl;
}

```



```
        of1<<power[i]<<endl;
    }
```

```
float max_freq = 0.5;
```

```
freq.resize(N/2+1);
ofstream of4("y1.txt");
for(int i=1;i<=N/2;i++){
    freq[i]=1/(float(i)/float(N/2)*max_freq);
    of4<<freq[i]<<endl;
}
```

```
int maxind = 1;
for (int i = 1; i <= N/2; i++){
    if(power[i]>power[maxind])maxind=i;
}
```

```
cout << "Calculated periodicity: " << freq[maxind]/365 << " years." << endl;
```

```
cufftDestroy(plan);
cudaFree(gpu_data);
free(data_h);
free(cpu_data);
}
```

```
import matplotlib.pyplot as plt
import numpy as np
```

```
ax=plt.subplot(111)
fn = open("x.txt")
n = fn.readlines()
```

```
n = [int(i) for i in n]
```

```
ft = open("y.txt")
```

```
T = ft.readlines()
```

```
T = [float (i) for i in T]
```

```
ft1 = open("y1.txt")
```

```
T1 = ft1.readlines()
```

```
T1 = [float (i) for i in T1]
```

```
#ax.plot(n, T)
```

```
ax.plot(T1, T)
```

```
ax.legend()
```

```
plt.show()
```