Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University)
Faculty of Informational Technologies and Programming

REPORT

about laboratory work № 1

« Parallel Clustering Methods»

Student
Nerinovsky Arseny **_M41331_**
(Surname, initials) Group

Saint-Petersburg, 2018

Report

June 29, 2019

1

, CUDA.

2

, , . . :

.

$$A(m \times n) = \begin{pmatrix} a_{0,0} & \cdots & a_{0,n} \\ \vdots & \ddots & \vdots \\ a_{m,0} & \cdots & a_{m,n} \end{pmatrix}, m \neq n.$$

.

$$d_{i,j} = \sum_{k=0}^{n} (a_{i,k} - a_{j,k})^2.$$

- CUDA . .

3

CUDA C, GPU: 1. , ; 2. , , ; 3. GPU; 4. , GPU. 5. .
, . CUDA 3 : - **host** - **global** - **device**
, , GPU. API CUDA - .

4

[1]:
```
%cat mm.cu
```

```
#include <cuda_runtime.h>

#include <iostream>
#include <memory>
#include <string>

#include <cuda.h>
#include <stdio.h>




#ifndef BLOCK_SIZE
# define BLOCK_SIZE 16
#endif

#ifndef _M
# define _M 10000
#endif

#ifndef _N
# define _N 10000
#endif

#if !defined(CUDA) && !defined(CPU) && !defined(CHECK)
# define CUDA
#endif

#define gpuErrchk(ans) { gpuAssert((ans), __FILE__, __LINE__); }
inline void gpuAssert(cudaError_t code, const char *file, int line, bool
abort=true)
{
   if (code != cudaSuccess)
   {
      fprintf(stderr,"gpuAssert: %s %s %d\n", cudaGetErrorString(code), file,
line);
      if (abort) exit(code);
   }
}

__global__ void mx_dist(float *m_in, float *m_out, int m, int n)
{
    int i = blockIdx.y * blockDim.y + threadIdx.y;
    int j = blockIdx.x * blockDim.x + threadIdx.x;
```

```
        float s = 0, sum = 0;

    if( i < m && j < m) {

        for(int k = 0; k < n; ++k) {
                s = m_in[i*m + k] - m_in[j*m + k];
                sum += s*s;
        }

        // printf("--> %d %d %f %f\n", j, i, m_in[j*n], sum);
        m_out[i*m + j] = sum;
    }
}

void mx_dist_cpu(float *m_in, float *m_out, int m, int n)
{
        float s, sum;

        for(int i = 0; i < m; ++i)
                for(int j = 0; j < m; ++j) {
                        sum = 0;
                        for(int k = 0; k < n; ++k) {
                                s = m_in[i*m + k] - m_in[j*m + k];
                                sum += s*s;
                        }
                        m_out[i*m + j] = sum;
                }
}

void init_mx(float *A, size_t m, size_t n)
{
        for(int i = 0; i < m; ++i) {
                for(int j = 0; j < n; ++j) {
                        float t = sin(i*m + j) * 10 + 1;
                        A[i*m + j] = t;
                }
        }
}
void print_mx(float *A, size_t m, size_t n)
{
        for(int i = 0; i < m; ++i) {
                for(int j = 0; j < n; ++j) {
                        printf("%d %d %f\n", i, j, A[i*m + j]);
                }
        }
}

void cmp_mx(float *A, float *B, size_t m, size_t n)
```

```
{
        for(int i = 0; i < m; ++i) {
                for(int j = 0; j < n; ++j) {
                        if( abs(A[i*m + j] - B[i*m + j]) > 0.01) {
                                printf("not equal %f %f\n", A[i*m + j], B[i*m +
j]);
                                return;
                        } else {
                                printf("Equal\n");
                        }
                }
        }
}



float *run_cuda(float *A, size_t m, size_t n)
{
        cudaError_t e;

        float *A_d;
        float *B, *B_d;

        B = (float*) malloc(m*m*sizeof(float));


        e = cudaMalloc(&A_d, m*n*sizeof(float));
        gpuErrchk(e);
        e = cudaMalloc(&B_d, m*m*sizeof(float));
        gpuErrchk(e);


        e = cudaMemcpy(A_d, A, m*n*sizeof(float),
                                cudaMemcpyHostToDevice);
        gpuErrchk(e);


    unsigned int grid_rows = (m + BLOCK_SIZE - 1) / BLOCK_SIZE;
    unsigned int grid_cols = (n + BLOCK_SIZE - 1) / BLOCK_SIZE;

    dim3 dimGrid(grid_cols, grid_rows);
    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);

        mx_dist<<<dimGrid, dimBlock>>>(A_d, B_d, m, n);


        e = cudaMemcpy(B, B_d, m*m*sizeof(float),
                                cudaMemcpyDeviceToHost);
```

```
        gpuErrchk(e);


        cudaFree(A_d);
        cudaFree(B_d);


        return B;
}


float *run_cpu(float *A, size_t m, size_t n)
{

        float *B;
        B = (float*) malloc(m*m*sizeof(float));

        mx_dist_cpu(A, B, m, n);

        return B;
}

int main()
{

        int m = _M, n = _N;
        float *A;
        A = (float*) malloc(m*n*sizeof(float));
        init_mx(A, m, n);

#if defined(CUDA) | defined(CHECK)
        float *gpu = run_cuda(A, m, n);
#endif

#if defined(CPU) | defined(CHECK)
        float *cpu = run_cpu(A, m, n);
#endif

#if defined(CHECK)
        cmp_mx(gpu, cpu, m, m);
#endif
        //for(int _j = 0; _j < size; ++_j) printf("%f ", h_vec[2][_j]);
        // printf("\n");


    return 0;
```

5

[2]:
```python
import subprocess
import os

def compile(*defs, **defskw):
    args = [f"-D{k}" for k in defs] + [f"-D{k}={v}" for k, v in defskw.items()]
    _cmd = 'nvcc mm.cu -o mm'.split() + args
    # print(' '.join(_cmd))
    cmd = subprocess.run(_cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

    if(cmd.stdout): print('cmd.stdout', cmd.stdout)
    if(cmd.stderr): print('cmd.stderr', cmd.stderr)

def run(env=None):
    cmd = subprocess.run('./mm', stdout=subprocess.PIPE, stderr=subprocess.
 ↪PIPE, env=env)
    return cmd.stdout.decode('utf8')
```

, CPU GPU

[3]:
```python
compile('CPU', _N=10_00, _M=50_0)
print("Execution time on CPU", end="\n\t")
%timeit run()
print()

compile('CUDA', _N=10_00, _M=50_0)
print("Execution time on GPU", end="\n\t")
%timeit run()
print()
```

```
Execution time on CPU
        1.2 s ś 371 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)

Execution time on GPU
        19.5 ms ś 394 ţs per loop (mean ś std. dev. of 7 runs, 100 loops each)
```

GPU.

, .

[4]:
```python
for bs in [8, 16, 32, 64, 128, 256, 512, 1024, 2048]:
    compile('CUDA', _N=10_00, _M=50_0, BLOCK_SIZE=bs)
    print(f"Execution time with block size {bs}", end="\n\t")
    %timeit run()
    print()
```

```
Execution time with block size 8
        29.7 ms ś 698 ţs per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

```
Execution time with block size 16
        21.6 ms ś 2.95 ms per loop (mean ś std. dev. of 7 runs, 100 loops each)

Execution time with block size 32
        28.4 ms ś 1.82 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)

Execution time with block size 64
        21.2 ms ś 3.38 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)

Execution time with block size 128
        21.5 ms ś 2.84 ms per loop (mean ś std. dev. of 7 runs, 100 loops each)

Execution time with block size 256
        21.8 ms ś 2.64 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)

Execution time with block size 512
        24.2 ms ś 1.79 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)

Execution time with block size 1024
        22.2 ms ś 940 ţs per loop (mean ś std. dev. of 7 runs, 10 loops each)

Execution time with block size 2048
        22.6 ms ś 3.02 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

6

, CUDA. . CUDA GPU.