Выполнила: Белоусова Е., ИП-911

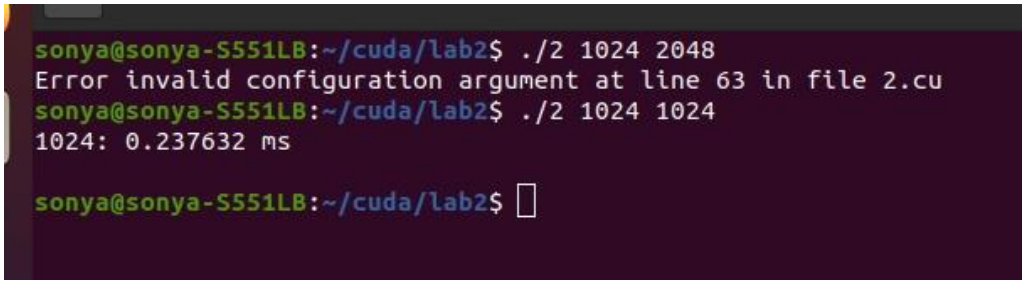
Задача

Задание: выполнить задание лабораторной 1, используя Events и nvprof.

Цель: научиться обрабатывать ошибки и профилировать код при выполнении программы на GPU.

Описание работы программы

Ошибки обрабатываем с помощью макроса CUDA_CHECK_RETURN.



Измерим время работы программы с помощью Events – объектов событий. Для этого объявим переменные типа события начала и окончания выполнения ядра. Инициализируем их с помощью cudaEventCreate(). Далее привяжем события start (начало выполнения ядра) и stop (окончание выполнения ядра). С помощью cudaEventSynchronize() будем ожидать окончания выполнения ядра, синхронизацию по событию stop. Получим время, прошедшее между событиями start и stop с помощью функции cudaEventElapsedTime. Уничтожить события необходимо с помощью cudaEventDestroy.

Результаты измерения:

size/threads	1	2	4	8	16	32	64	128	256	512	1024	CPU
1024	0,275456	0,230848	0,240224	0,223168	0,220096	0,26256	0,061888	0,215232	0,220352	0,242848	0,281888	0,00628
2048	0,269664	0,036768	0,032256	0,2552	0,21968	0,049728	0,01776	0,05632	0,054336	0,058496	0,051968	0,02204
4096	0,294304	0,060512	0,040736	0,225568	0,224768	0,0184	0,018272	0,022304	0,020192	0,02096	0,016928	0,03678
8192	0,391936	0,107264	0,067232	0,247232	0,224032	0,024512	0,022432	0,022784	0,01984	0,025248	0,020832	0,07336
16384	0,564672	0,195424	0,113792	0,423776	0,244544	0,031776	0,02816	0,620704	0,031456	0,870912	0,02752	0,14724
32768	0,916992	0,37664	0,203424	0,306848	0,26688	0,05184	0,046688	0,037664	0,04336	0,040896	0,044064	0,29417
65536	1,674304	0,975232	0,600576	0,418624	0,323456	0,086816	0,0752	0,070912	0,073472	0,072704	0,079168	0,48947
131072	3,096736	2,230304	1,043936	2,058272	0,524928	0,19968	0,338944	0,326432	0,330784	0,38176	0,330144	1,00539
262144	5,876064	4,147008	1,694304	1,9912	0,45744	0,352672	0,482848	0,468064	0,316032	1,261824	0,318464	1,69093
524288	12,111488	8,27568	3,360512	2,117824	1,121536	0,801184	0,590784	0,58864	2,5488	1,051296	0,742496	3,23727
1048576	24,458303	12,575904	7,870592	3,54656	2,631072	1,396352	1,348768	1,280928	1,276032	1,278048	1,276864	7,24478
2097152	50,19101	27,428864	13,360096	7,346432	4,48688	2,531328	2,390464	2,357248	2,356864	2,352864	2,345664	13,631
4194304	95,373215	48,316193	25,441216	12,747072	6,724416	4,844512	4,510688	4,574272	4,573472	4,556736	4,486144	26,164
8388608	191,982941	90,937027	47,857346	25,253183	12,904544	10,626976	8,864	8,83152	8,85712	8,7888	8,765696	52,4967
						мс.						

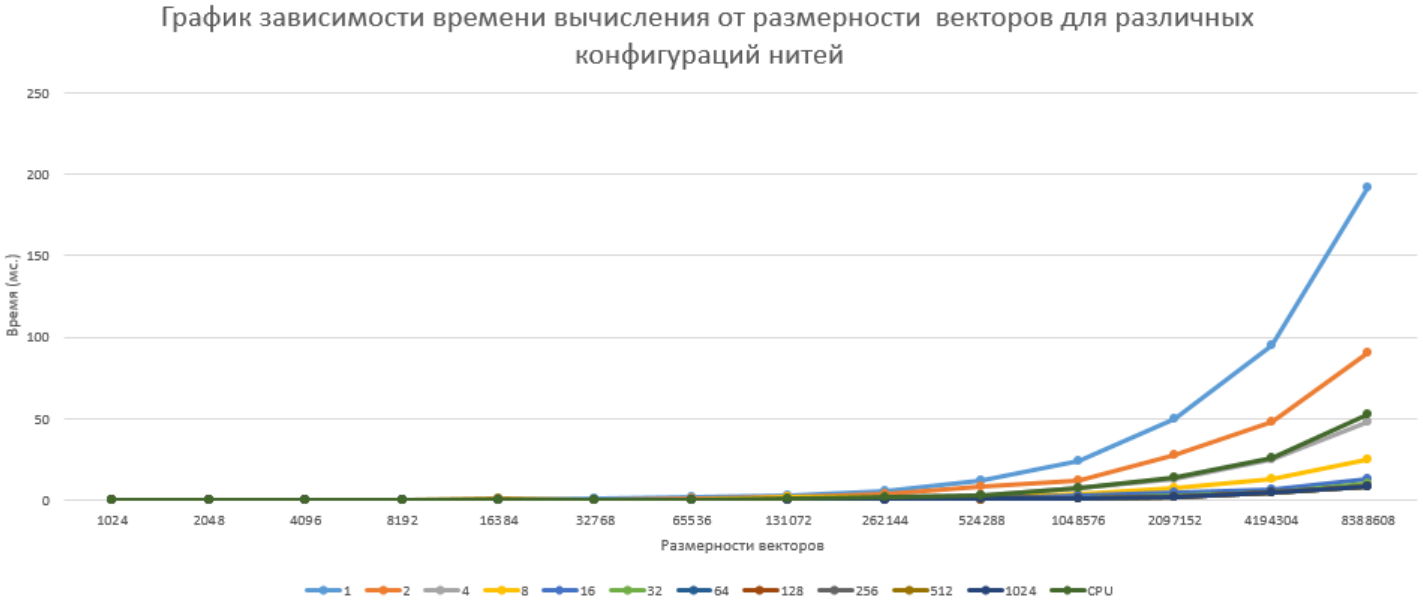
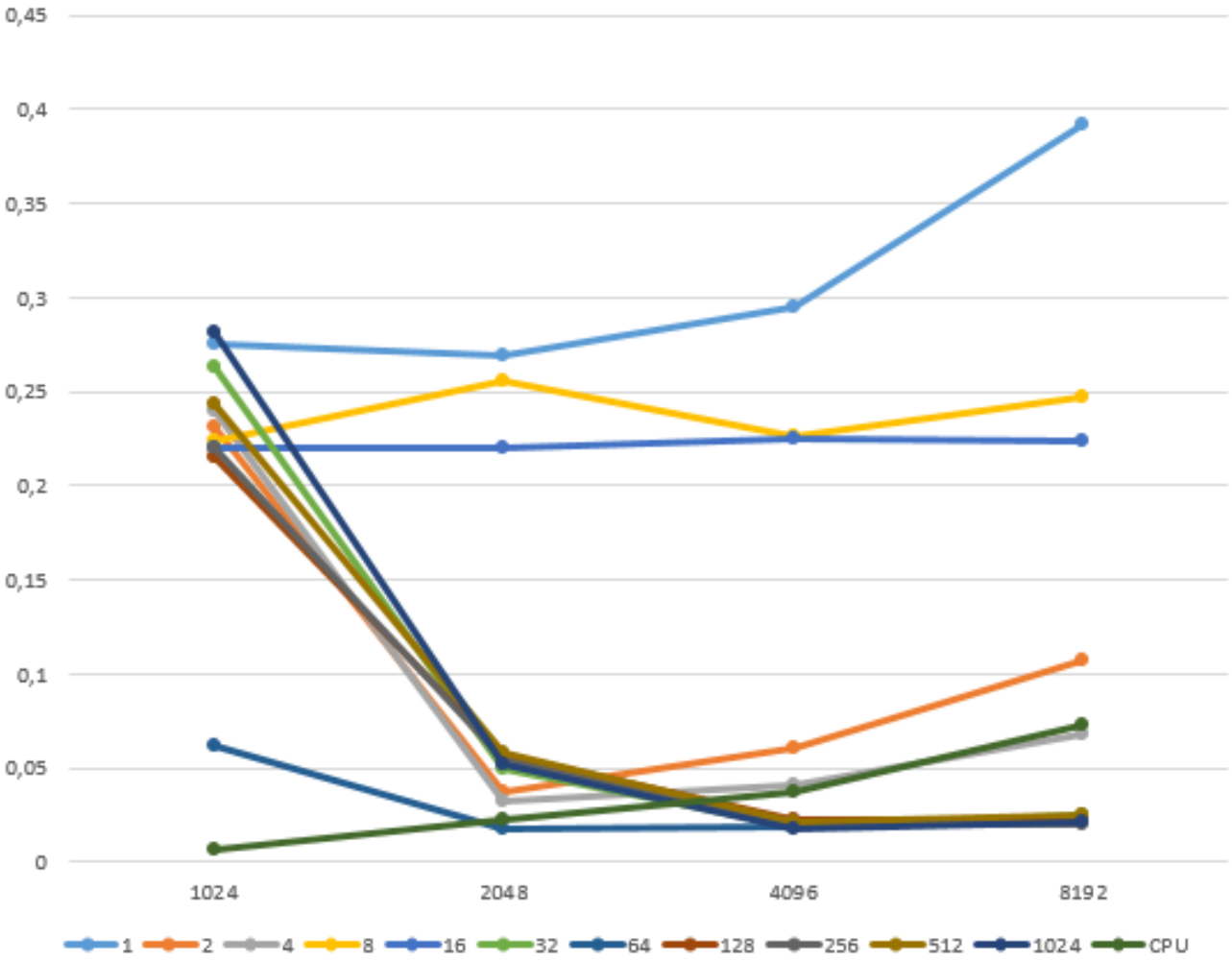
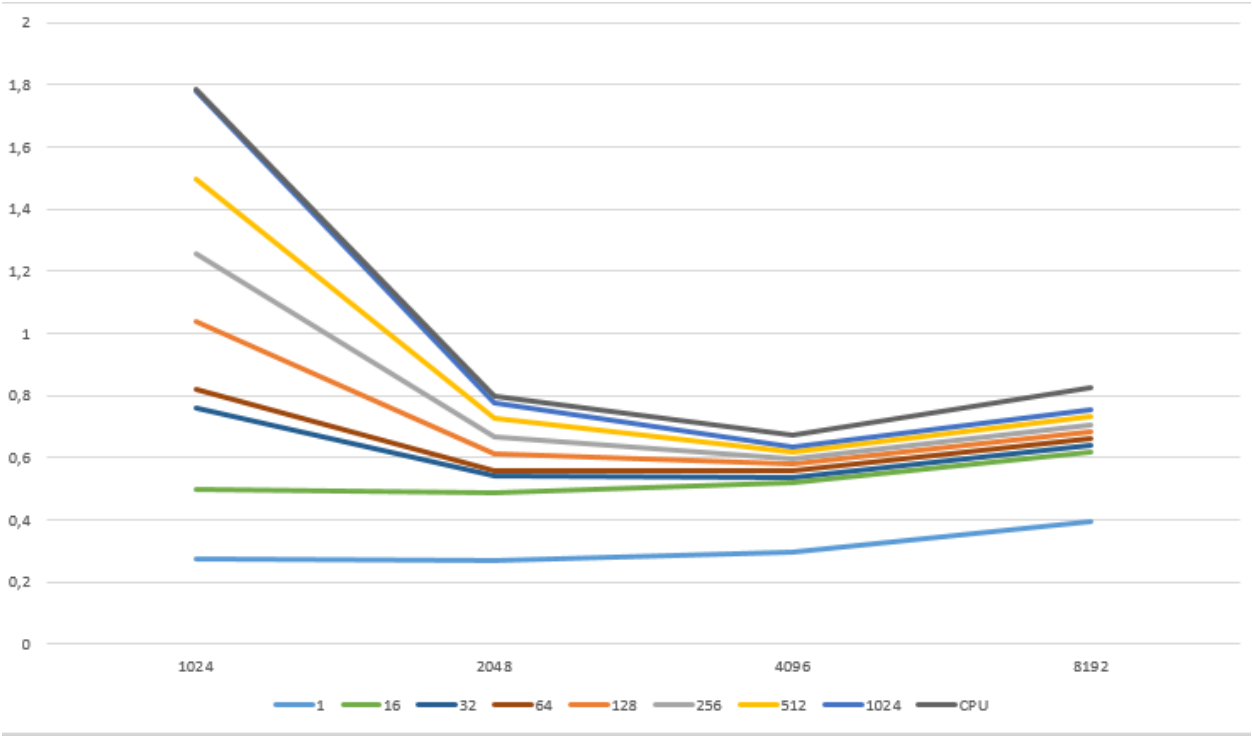


График зависимости времени вычисления от размерности векторов для различных конфигураций нитей
Время (мс.)
250
200
150
100
50
0
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
Размерности векторов
1
2
4
8
16
32
64
128
256
512
1024
CPU


0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
1024
2048
4096
8192
1
2
4
8
16
32
64
128
256
512
1024
CPU



Профилировка программы с помощью nvprof.

```
sonya@sonya-S551LB:~/cuda/lab2$ sudo nvprof ./2 1024 1024
==5105== NVPROF is profiling process 5105, command: ./2 1024 1024
1024: 0.144192 ms

==5105== Profiling application: ./2 1024 1024
==5105== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   58.79%  3.1040us         1  3.1040us  3.1040us  3.1040us  add(int*, int*, int*, int)
                   41.21%  2.1760us         1  2.1760us  2.1760us  2.1760us  [CUDA memcpy DtoH]
      API calls:   99.36%  197.85ms         2  98.927ms  1.7240us  197.85ms  cudaEventCreate
                    0.18%  352.65us        97  3.6350us     303ns  142.32us  cuDeviceGetAttribute
                    0.12%  243.93us         1  243.93us  243.93us  243.93us  cuDeviceTotalMem
                    0.11%  210.24us         3  70.079us  4.2280us  199.36us  cudaMalloc
                    0.10%  190.85us         3  63.618us  13.011us  135.18us  cudaFree
                    0.07%  135.74us         1  135.74us  135.74us  135.74us  cudaLaunchKernel
                    0.03%  58.593us         1  58.593us  58.593us  58.593us  cuDeviceGetName
                    0.02%  34.413us         1  34.413us  34.413us  34.413us  cudaMemcpy
                    0.01%  14.153us         2  7.0760us  4.8420us  9.3110us  cudaEventRecord
                    0.01%  10.864us         1  10.864us  10.864us  10.864us  cuDeviceGetPCIBusId
                    0.00%  5.8300us         1  5.8300us  5.8300us  5.8300us  cudaEventSynchronize
                    0.00%  3.8130us         3  1.2710us     344ns  2.3400us  cuDeviceGetCount
                    0.00%  3.5650us         2  1.7820us     922ns  2.6430us  cudaEventDestroy
                    0.00%  2.8800us         1  2.8800us  2.8800us  2.8800us  cudaEventElapsedTime
                    0.00%  2.7830us         2  1.3910us     458ns  2.3250us  cuDeviceGet
                    0.00%     601ns         1     601ns     601ns     601ns  cuDeviceGetUuid
                    0.00%     462ns         1     462ns     462ns     462ns  cudaGetLastError
```

Листинг

//2.cu

```
#include <cuda.h>
#include <cuda_runtime.h>
#include <iomanip>
#include <iostream>
#include <malloc.h>
#include <stdio.h>

using namespace std;

#define CUDA_CHECK_RETURN(value)                                \
 {                                                              \
   cudaError_t _m_cudaStat = value;                             \
```

```
   if (_m_cudaStat != cudaSuccess) {                                \
     fprintf(stderr, "Error %s at line %d in file %s\n",            \
           cudaGetErrorString(_m_cudaStat), __LINE__, __FILE__);    \
     exit(1);                                                       \
   }                                                                \
  }

__global__ void add(int*a, int *b, int *c, int N) {
int i = threadIdx.x + blockIdx.x * blockDim.x;
  if(i >= N) return;
  c[i] = a[i] + b[i];
}

void init(int*a, int *b, int *c, int N)
{
   for(int k = 0; k < N; k++)
   {
     a[k] = k;
     b[k] = k;
     c[k] = 0;
   }
}

int main(int argc,char* argv[])
{
   float elapsedTime;
   int N = 0;
   int *dev_a, *dev_b, *dev_c, *c, *a, *b;

   if(argc == 3)
   {
       int N = atoi(argv[1]);
       int th = atoi(argv[2]);
       cudaEvent_t start, stop;
       cudaEventCreate(&start);
       cudaEventCreate(&stop);

       a = (int*)calloc(N, sizeof(int));
       b = (int*)calloc(N, sizeof(int));
       c = (int*)calloc(N, sizeof(int));
       init(a, b, c, N);
       CUDA_CHECK_RETURN(cudaMalloc((void **)&dev_a, N * sizeof(int)));
       CUDA_CHECK_RETURN(cudaMalloc((void **)&dev_b, N * sizeof(int)));
       CUDA_CHECK_RETURN(cudaMalloc((void **)&dev_c, N * sizeof(int)));

       cudaEventRecord(start, 0);
       add<<<N / th, th>>>(dev_a, dev_b, dev_c, N);
       cudaEventRecord(stop, 0);
       cudaEventSynchronize(stop);

       CUDA_CHECK_RETURN(cudaGetLastError());
       cudaEventElapsedTime(&elapsedTime, start, stop);
       CUDA_CHECK_RETURN(cudaMemcpy(c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost));
       fprintf(stderr, "%d: %.6f ms\n", N, elapsedTime);
       cudaEventDestroy(start);
       cudaEventDestroy(stop);
       free(c);
       cudaFree(dev_a);
       cudaFree(dev_b);
       cudaFree(dev_c);
       cout << endl;
   }
   else if(argc == 1){
     for (int k = 1 << 6; k <= 1 << 10; k = k << 1) {
       fprintf(stderr, "Threads per block(%i):\n", k);
       for (int j = 10; j <= 23; j++) {
```

```
        cudaEvent_t start, stop;
        cudaEventCreate(&start);
        cudaEventCreate(&stop);
        N = 1 << j;

        a = (int*)calloc(N, sizeof(int));
        b = (int*)calloc(N, sizeof(int));
        c = (int*)calloc(N, sizeof(int));
        init(a, b, c, N);
        CUDA_CHECK_RETURN(cudaMalloc((void **)&dev_a, N * sizeof(int)));
        CUDA_CHECK_RETURN(cudaMalloc((void **)&dev_b, N * sizeof(int)));
        CUDA_CHECK_RETURN(cudaMalloc((void **)&dev_c, N * sizeof(int)));

        cudaEventRecord(start, 0);
        add<<<N / k, k>>>(dev_a, dev_b, dev_c, N);
        cudaEventRecord(stop, 0);
        cudaEventSynchronize(stop);

        CUDA_CHECK_RETURN(cudaGetLastError());
        cudaEventElapsedTime(&elapsedTime, start, stop);
        CUDA_CHECK_RETURN(cudaMemcpy(c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost));
        fprintf(stderr, "%d: %.6f ms\n", N, elapsedTime);
        cudaEventDestroy(start);
        cudaEventDestroy(stop);
        free(c);
        cudaFree(dev_a);
        cudaFree(dev_b);
        cudaFree(dev_c);
       }
       cout << endl;
      }
     }
     return 0;
}
```