

Домашнее задание №2

Липатов Данила МСМТ243

(пункты 1,2,4,5)

После подключения к НРС был перенесен исходный скрипт (скрипт писался в VSC .cpp, но компилировался и запускался с .cu, локально так же компилировался и запускался с .cu)
Был подключен модуль для CUDA - module load nvidia_sdk/nvhpc/23.5

Далее файл компилировался :

```
nvcc hw_1.cu -o hw_1.out
```

И запускался на узле А следующим образом

```
srun -n 1 --gpus=1 --constraint="type_a" hw_1.out
```

Реализация для каждого из методов лежит в соответствующем файле.

Аналогично запуску выше были запущены и другие реализации с pinned, unified, shared, cuda-stream.

Разница между global, pinned, unified memory следующая:

Global:

```
double *h_A = (double*)malloc(bytes);
double *h_B = (double*)malloc(bytes);
double *h_C = (double*)malloc(bytes);
double *d_A, *d_B, *d_C;

cudaMalloc(&d_A, bytes);
cudaMalloc(&d_B, bytes);
cudaMalloc(&d_C, bytes);

cudaMemcpy(d_A, h_A, bytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, bytes, cudaMemcpyHostToDevice);
```

Pinned:

```
double *h_A, *h_B, *h_C;
cudaMallocHost((void **) &h_A, size);
cudaMallocHost((void **) &h_B, size);
cudaMallocHost((void **) &h_C, size);

double *d_A, *d_B, *d_C;
cudaMalloc((void **) &d_A, size);
cudaMalloc((void **) &d_B, size);
cudaMalloc((void **) &d_C, size);

cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
```

Unified:

```
double *h_A, *h_B, *h_C;
cudaMallocManaged(&h_A, size);
```

```

cudaMallocManaged(&h_B, size);
cudaMallocManaged(&h_C, size);

```

По времени вычисления они одинаковые (+-).

Для shared memory используется pinned memory + переписанная функция умножения матриц как раз для разделения памяти, что позволяет значительно ускорить процесс умножения и получить следующий график:

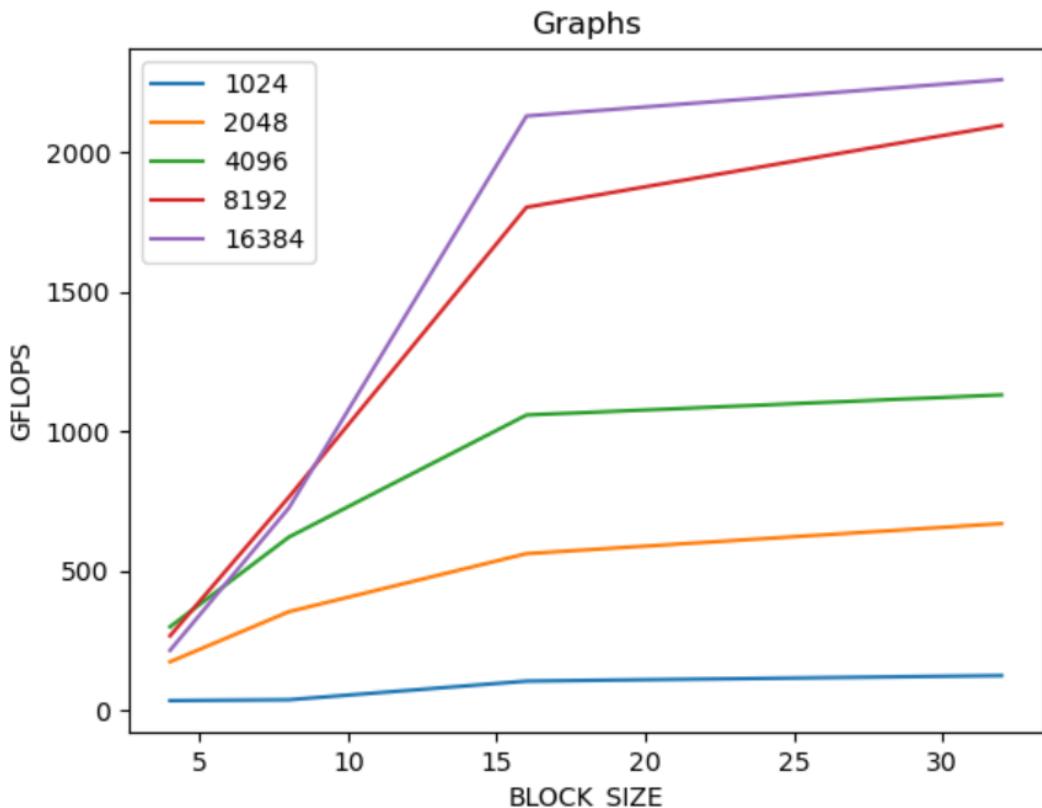


Рис. 1.1 Зависимость GFLOPS от BLOCK_SIZE

Для CUDA stream так же был произведен анализ, для выяснения оптимального количества потоков (1/2/4) на графике ниже, и произведено сравнение с cuBLAS, CPU вычислениями

Matrix C:								
180.0000	122.0000	158.0000	241.0000	153.0000	260.0000	277.0000	195.0000	
124.0000	85.0000	95.0000	172.0000	90.0000	174.0000	172.0000	117.0000	
178.0000	162.0000	184.0000	262.0000	131.0000	276.0000	242.0000	180.0000	
226.0000	117.0000	151.0000	265.0000	83.0000	219.0000	197.0000	165.0000	
136.0000	106.0000	185.0000	190.0000	144.0000	231.0000	224.0000	180.0000	
88.0000	147.0000	176.0000	175.0000	103.0000	196.0000	171.0000	157.0000	
134.0000	189.0000	141.0000	193.0000	116.0000	196.0000	194.0000	179.0000	
133.0000	74.0000	131.0000	159.0000	101.0000	198.0000	138.0000	133.0000	
Matrix h_c:								
180.0000	122.0000	158.0000	241.0000	153.0000	260.0000	277.0000	195.0000	
124.0000	85.0000	95.0000	172.0000	90.0000	174.0000	172.0000	117.0000	
178.0000	162.0000	184.0000	262.0000	131.0000	276.0000	242.0000	180.0000	
226.0000	117.0000	151.0000	265.0000	83.0000	219.0000	197.0000	165.0000	
136.0000	106.0000	185.0000	190.0000	144.0000	231.0000	224.0000	180.0000	
88.0000	147.0000	176.0000	175.0000	103.0000	196.0000	171.0000	157.0000	
134.0000	189.0000	141.0000	193.0000	116.0000	196.0000	194.0000	179.0000	
133.0000	74.0000	131.0000	159.0000	101.0000	198.0000	138.0000	133.0000	
Matrix C_ref:								
180.0000	122.0000	158.0000	241.0000	153.0000	260.0000	277.0000	195.0000	
124.0000	85.0000	95.0000	172.0000	90.0000	174.0000	172.0000	117.0000	
178.0000	162.0000	184.0000	262.0000	131.0000	276.0000	242.0000	180.0000	
226.0000	117.0000	151.0000	265.0000	83.0000	219.0000	197.0000	165.0000	
136.0000	106.0000	185.0000	190.0000	144.0000	231.0000	224.0000	180.0000	
88.0000	147.0000	176.0000	175.0000	103.0000	196.0000	171.0000	157.0000	
134.0000	189.0000	141.0000	193.0000	116.0000	196.0000	194.0000	179.0000	
133.0000	74.0000	131.0000	159.0000	101.0000	198.0000	138.0000	133.0000	

Рис. 1.2 Сравнение cuBLAS, cuda-stream, CPU

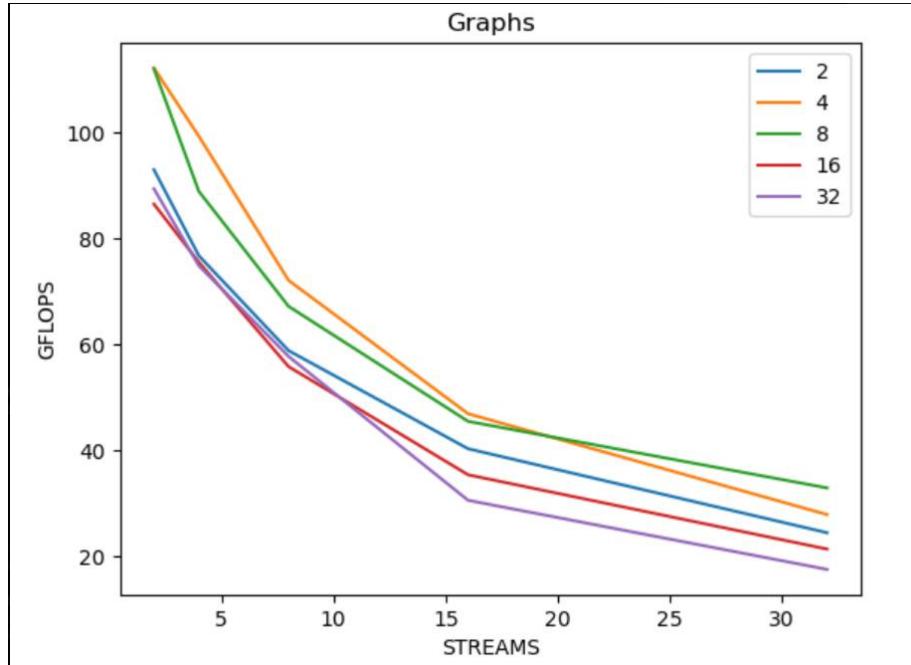


Рис. 1.3 матрица 1024

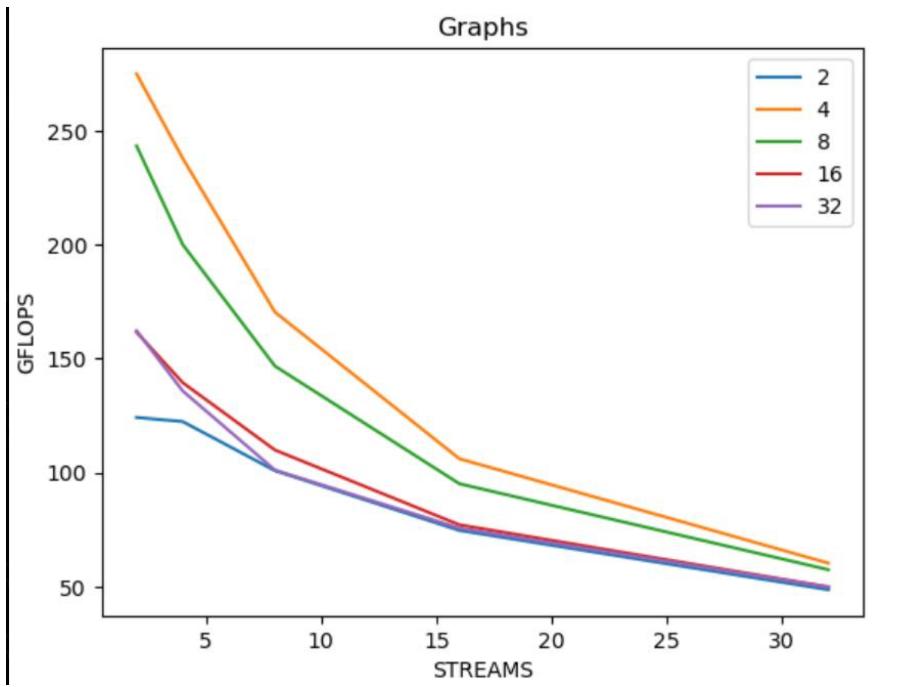


Рис. 1.4 матрица 2048

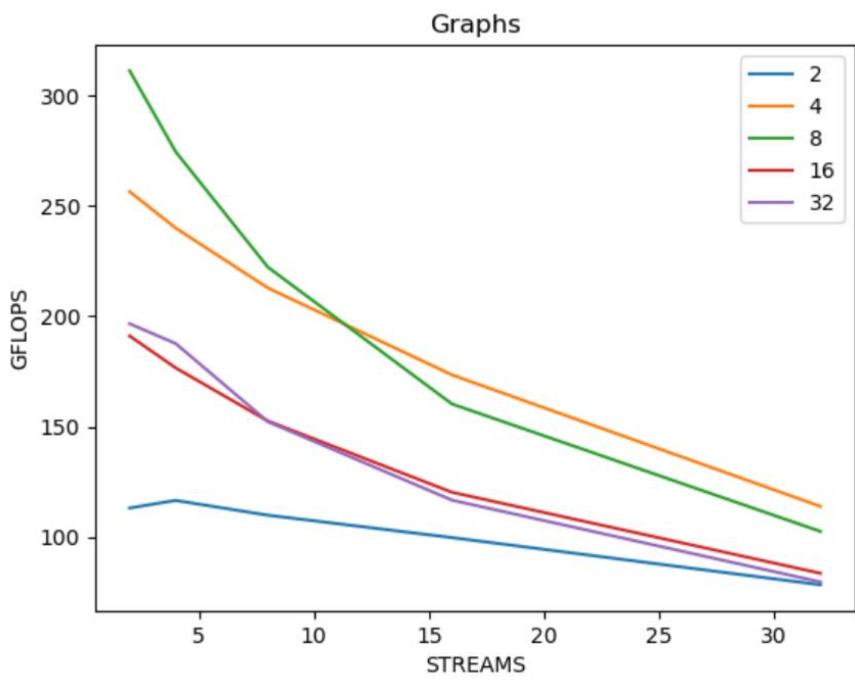


Рис. 1.5 матрица 4096

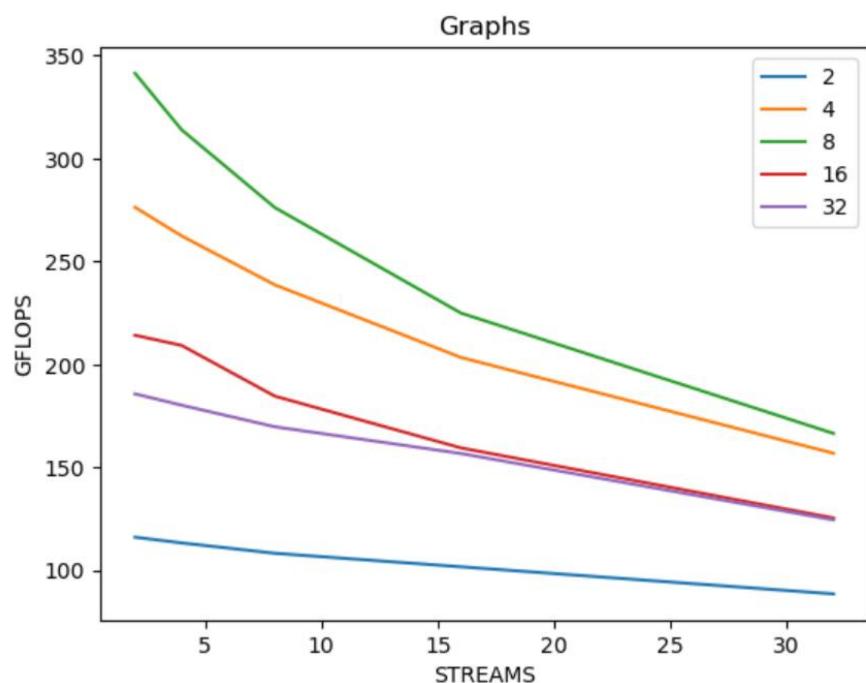


Рис. 1.6 матрица 8192

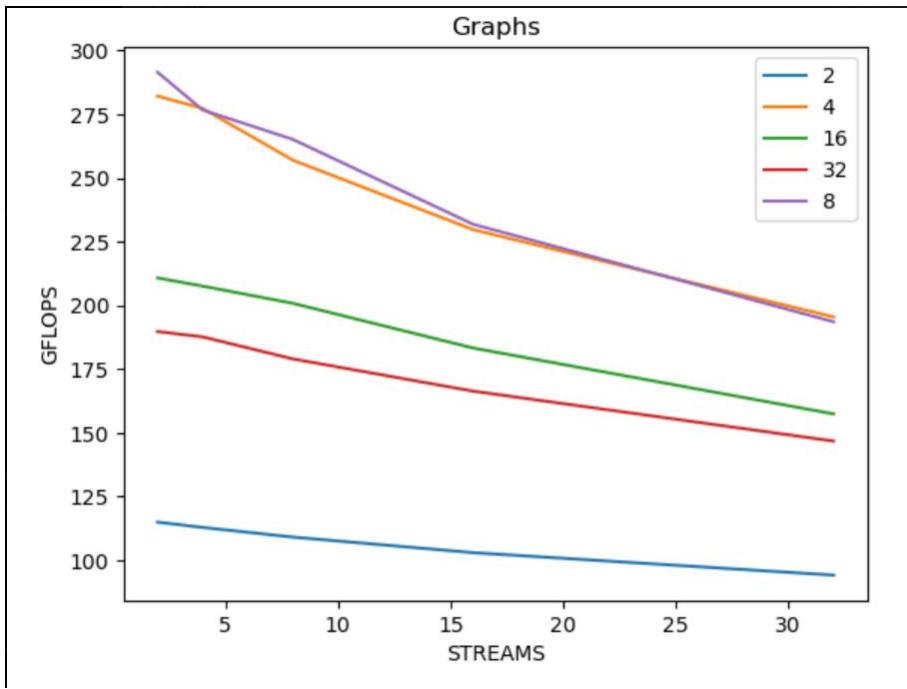


Рис. 1.7 матрица 16384

Потоки были кратны 2, если выбирается некратное 2-ум кол-во потоков, то оно округлялось сверху до кратного (так же чтобы был кратен кол-ву блоков), иначе матрица «обрезалась»

График для cuBLAS приведен ниже, его пиковая производительность в 8к GFLOPS соответствует максимальным мощностям ресурсов, так же было выяснено, что максимальная матрица равна (примерно) 35000, так как после 32768 возможно было еще добавить размеры, хотя матрица больше 40000 уже не проходила.

```
nvcc hw_1_cublas.cu -lcublas -o hw_1_cublas.out
```

График производительности:

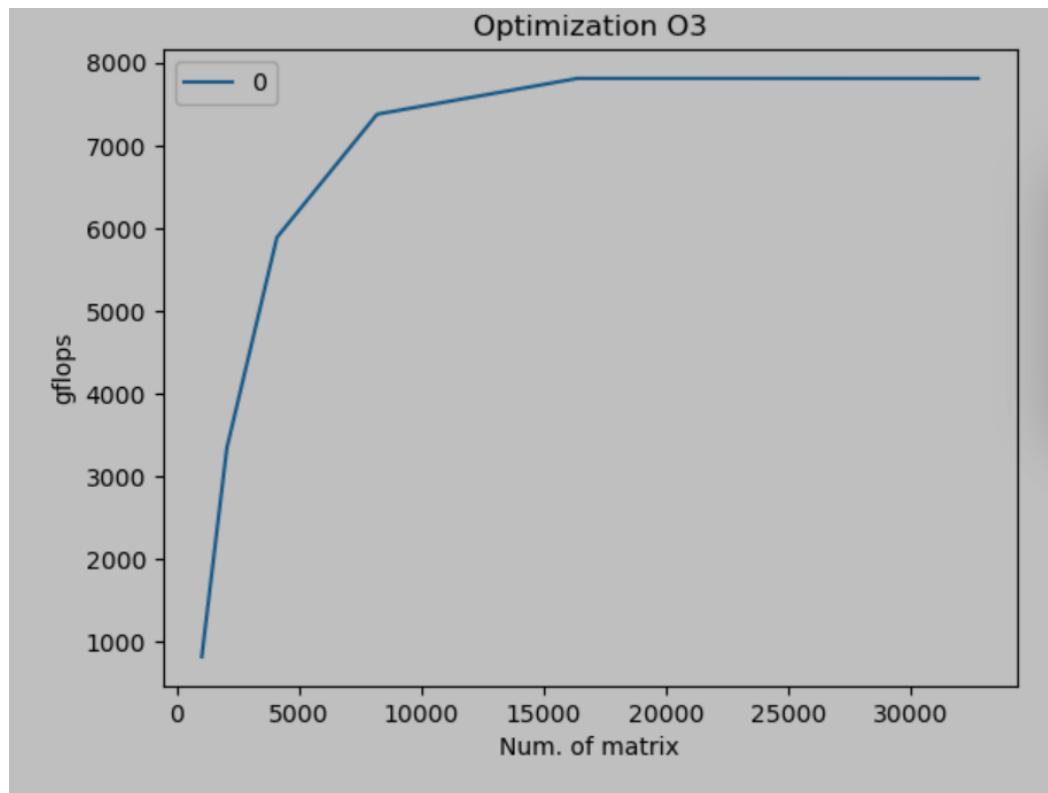


Рис. 1.8 GFLOPS от матрицы

Эмпирически было получено, что пиковая производительность приходится на большие матрицы, размер может превышать 32768×32768 , но уже при размерах 65530×65530 не получается просто так начать расчет.

Для реализации openMP на GPU использовался скрипт из предыдущей лабораторной с pragma:

```
#pragma omp target data map(to: A[0:N*N], B[0:N*N]) map(from: C[0:N*N])
#pragma omp target teams num_teams(NUM_TEAMS) distribute parallel for collapse(2)
И просто рассматривался вариант с
#pragma omp target teams distribute parallel for collapse(2)
```

Результат расчета сходится с предыдущими:

180.0000	122.0000	158.0000	241.0000	153.0000	260.0000	277.0000	195.0000
124.0000	85.0000	95.0000	172.0000	90.0000	174.0000	172.0000	117.0000
178.0000	162.0000	184.0000	262.0000	131.0000	276.0000	242.0000	180.0000
226.0000	117.0000	151.0000	265.0000	83.0000	219.0000	197.0000	165.0000
136.0000	106.0000	185.0000	190.0000	144.0000	231.0000	224.0000	180.0000
88.0000	147.0000	176.0000	175.0000	103.0000	196.0000	171.0000	157.0000
134.0000	109.0000	141.0000	193.0000	116.0000	196.0000	194.0000	179.0000
133.0000	74.0000	131.0000	159.0000	101.0000	198.0000	138.0000	133.0000

Рис. 1.9 Расчет матрицы

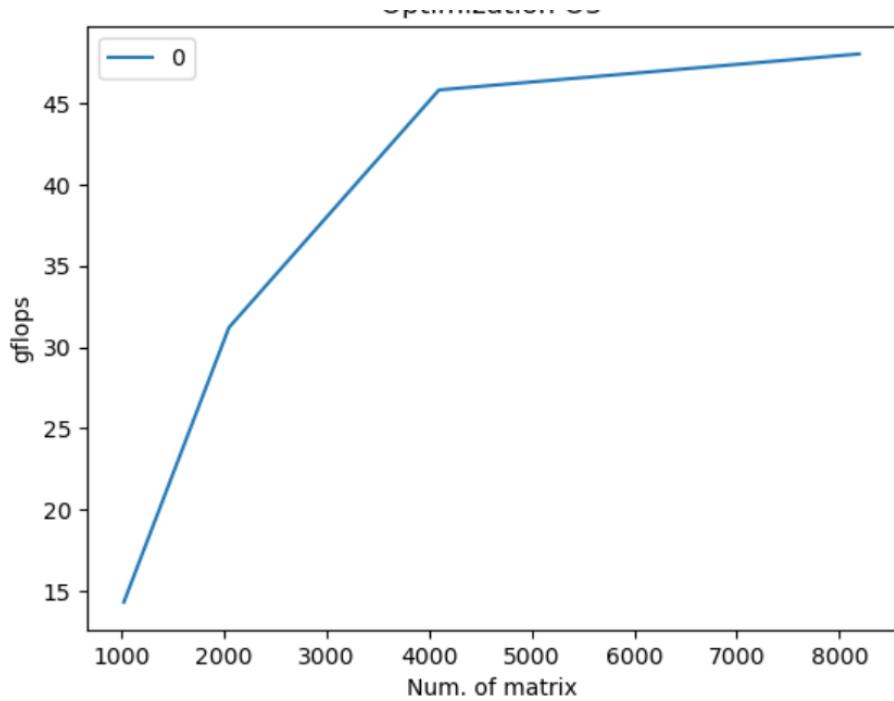


Рис. 1.10 GFLOPS от матрицы

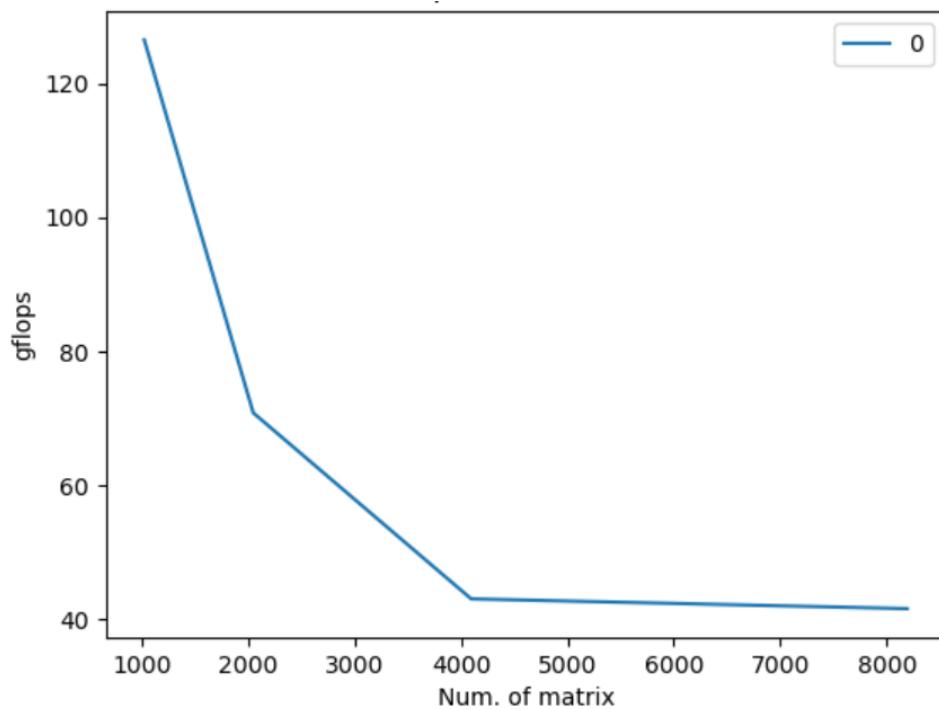


Рис. 1.11 GFLOPS от матрицы