**Выполнила: Белоусова Е., ИП-911**

**Задача**

**Задание:**

• написать программу транспонирования матриц, реализующую алгоритм без использования разделяемой памяти, наивный алгоритм с использованием разделяемой памяти и алгоритм с разрешением конфликта банков разделяемой памяти;

• провести профилирование программы с использованием nvprof или nvpp - сравнить время выполнения ядер, реализующих разные алгоритмы, и оценить эффективность использования разделяемой памяти (лекция 4).

**Цель:** научиться использовать разделяемую память.

**Описание работы программы**

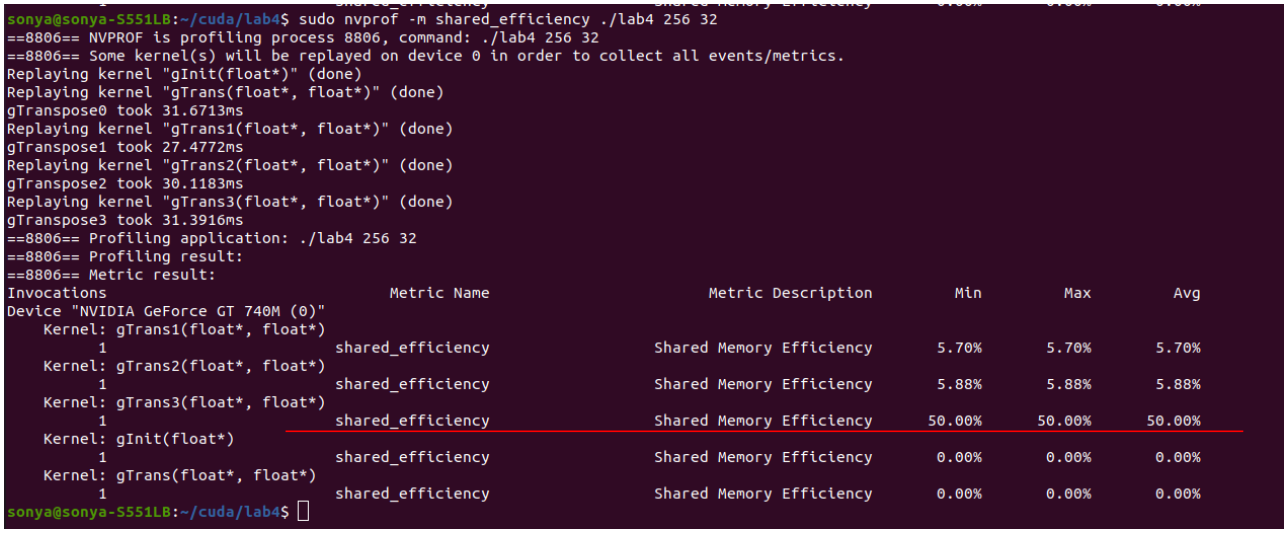
Написана программа транспонирования матриц, реализующая алгоритм без использования разделяемой памяти, наивный алгоритм с использованием разделяемой памяти и алгоритм с разрешением конфликта банков разделяемой памяти.

Сравнение времени выполнения ядер:

```
sonya@sonya-S551LB:~/cuda/lab4$ sudo nvprof ./lab4 256 32
==8577== NVPROF is profiling process 8577, command: ./lab4 256 32
gTranspose0 took 0.06528ms
gTranspose1 took 0.0464ms
gTranspose2 took 0.046656ms
gTranspose3 took 0.034592ms
==8577== Profiling application: ./lab4 256 32
==8577== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   62.87%  511.29us         3  170.43us  170.41us  170.44us  [CUDA memcpy HtoD]
                   19.03%  154.76us         1  154.76us  154.76us  154.76us  [CUDA memcpy DtoH]
                    5.44%  44.226us         1  44.226us  44.226us  44.226us  gTrans(float*, float*)
                    4.38%  35.586us         1  35.586us  35.586us  35.586us  gTrans1(float*, float*)
                    4.15%  33.762us         1  33.762us  33.762us  33.762us  gTrans2(float*, float*)
                    2.73%  22.209us         1  22.209us  22.209us  22.209us  gTrans3(float*, float*)
                    1.40%  11.392us         1  11.392us  11.392us  11.392us  gInit(float*)
      API calls:   98.62%  194.69ms         2  97.343ms  2.6490us  194.68ms  cudaEventCreate
                    0.30%  596.09us         4  149.02us  45.598us  192.93us  cudaEventSynchronize
                    0.22%  442.23us         4  110.56us  57.067us  250.52us  cudaMemcpy
                    0.22%  433.11us         5  86.622us  3.3210us  266.91us  cudaMalloc
                    0.16%  324.95us        97  3.3500us     318ns  138.21us  cuDeviceGetAttribute
                    0.16%  312.81us         5  62.561us  6.8720us  140.66us  cudaFree
                    0.12%  234.56us         5  46.911us  16.424us  152.61us  cudaLaunchKernel
                    0.11%  220.11us         1  220.11us  220.11us  220.11us  cuDeviceTotalMem
                    0.03%  65.680us         1  65.680us  65.680us  65.680us  cuDeviceGetName
                    0.02%  40.657us         8  5.0820us  3.0730us  7.3210us  cudaEventRecord
                    0.01%  12.186us         1  12.186us  12.186us  12.186us  cuDeviceGetPCIBusId
                    0.01%  10.781us         4  2.6950us  2.0700us  3.3910us  cudaEventElapsedTime
                    0.01%  10.325us         1  10.325us  10.325us  10.325us  cudaDeviceSynchronize
                    0.00%  4.9840us         2  2.4920us  1.5570us  3.4270us  cudaEventDestroy
                    0.00%  4.7940us         3  1.5980us     374ns  2.5890us  cuDeviceGetCount
                    0.00%  2.7330us         2  1.3660us     622ns  2.1110us  cuDeviceGet
                    0.00%     644ns         1     644ns     644ns     644ns  cuDeviceGetUuid
                    0.00%     615ns         1     615ns     615ns     615ns  cudaGetLastError
sonya@sonya-S551LB:~/cuda/lab4$
```

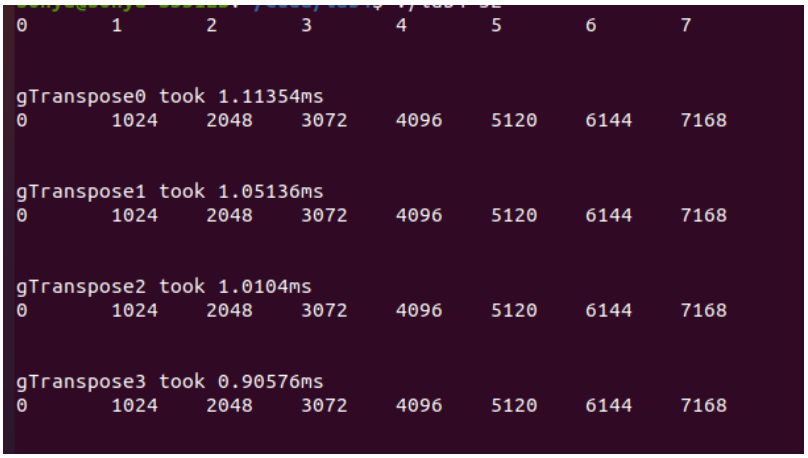
Функции gTranpose1 и gTranpose2 используют разделяемую память (выделяют динамически и статически соответственно). В нее считывается блок матрицы по строкам и записывается по столбам (происходит транспонирование). Далее из буффера считываем по строкам и записываем по строкам. Присутствует конфликт банков (более одной нити варпа может обратиться к одному и тому же банку). В gTranpose3 этот конфликт разрешается путем выделения дополнительной памяти (при считывании по столбцам обращение к разным банкам).

Сравнение оценки эффективности использования разделяемой памяти:



```
sonya@sonya-S551LB:~/cuda/lab4$ sudo nvprof -m shared_efficiency ./lab4 256 32
==8806== NVPROF is profiling process 8806, command: ./lab4 256 32
==8806== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Replaying kernel "gInit(float*)" (done)
Replaying kernel "gTrans(float*, float*)" (done)
gTranspose0 took 31.6713ms
Replaying kernel "gTrans1(float*, float*)" (done)
gTranspose1 took 27.4772ms
Replaying kernel "gTrans2(float*, float*)" (done)
gTranspose2 took 30.1183ms
Replaying kernel "gTrans3(float*, float*)" (done)
gTranspose3 took 31.3916ms
==8806== Profiling application: ./lab4 256 32
==8806== Profiling result:
==8806== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GT 740M (0)"
    Kernel: gTrans1(float*, float*)
          1                        shared_efficiency                  Shared Memory Efficiency       5.70%       5.70%       5.70%
    Kernel: gTrans2(float*, float*)
          1                        shared_efficiency                  Shared Memory Efficiency       5.88%       5.88%       5.88%
    Kernel: gTrans3(float*, float*)
          1                        shared_efficiency                  Shared Memory Efficiency      50.00%      50.00%      50.00%
    Kernel: gInit(float*)
          1                        shared_efficiency                  Shared Memory Efficiency       0.00%       0.00%       0.00%
    Kernel: gTrans(float*, float*)
          1                        shared_efficiency                  Shared Memory Efficiency       0.00%       0.00%       0.00%
sonya@sonya-S551LB:~/cuda/lab4$
```

Проверка работоспособности программы (выводится первая строка матрицы размера 1024):



```
0       1       2       3       4       5       6       7

gTranspose0 took 1.11354ms
0       1024    2048    3072    4096    5120    6144    7168

gTranspose1 took 1.05136ms
0       1024    2048    3072    4096    5120    6144    7168

gTranspose2 took 1.0104ms
0       1024    2048    3072    4096    5120    6144    7168

gTranspose3 took 0.90576ms
0       1024    2048    3072    4096    5120    6144    7168
```

Оценка времени выполнения ядер:

| **Размер матрицы** | **32** | **64** | **128** | **256** | **512** | **1024** | **2048** |
|---|---|---|---|---|---|---|---|
| **gTranpose0** | 3,0720 us | 4,4160 us | 12,161 us | 44,226 us | 237,77 us | 886,88 us | 3, 5647 ms |
| **gTranpose1** | 2,5610 us | 3,0080 us | 9,6640 us | 35,586 us | 221,64 us | 782,91 us | 3,2395 ms |
| **gTranpose2** | 2,3680 us | 3,2640 us | 8,9290 us | 33,762 us | 203,98 us | 872,54 us | 5,4379 ms |
| **gTranpose3** | 1,9840 us | 2,5280 us | 5,8240 us | 22,209 us | 187,34 us | 755,93 us | 2,8888 ms |

Оценка эффективности использования разделяемой памяти:

| **Размер матрицы** | **32** | **64** | **128** | **256** | **512** | **1024** | **2048** |
|---|---|---|---|---|---|---|---|
| **gTranpose1** | 5,88% | 5,88% | 5,75% | 5,70% | 5,79% | 5,72% | 5,78% |
| **gTranpose2** | 5,88% | 5,88% | 5,88% | 5,88% | 5,88% | 5,88% | 5,88% |
| **gTranpose3** | 50% | 50% | 49,61% | 50% | 50% | 49,99% | 49,99% |

Не удалось получить больше 50%.

### Листинг

```
#include <cuda.h>

#include <cuda_runtime.h>

#include <iomanip>

#include <iostream>

#include <malloc.h>

#include <stdio.h>


#define SH_DIM 32


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 gTrans3(float *matrix, float *matrixT) {
 __shared__ float buffer_s[SH_DIM][SH_DIM + 1];
 int i = threadIdx.x + blockIdx.x * blockDim.x;
 int j = threadIdx.y + blockIdx.y * blockDim.y;
 int N = blockDim.x * gridDim.x;

 buffer_s[threadIdx.y][threadIdx.x] = matrix[i + j * N];
 __syncthreads();

 i = threadIdx.x + blockIdx.y * blockDim.x;
 j = threadIdx.y + blockIdx.x * blockDim.y;
 matrixT[i + j * N] = buffer_s[threadIdx.x][threadIdx.y];
}

__global__ void gTrans2(float *matrix, float *matrixT) {
 __shared__ float buffer_s[SH_DIM][SH_DIM];
 int i = threadIdx.x + blockIdx.x * blockDim.x;
 int j = threadIdx.y + blockIdx.y * blockDim.y;
 int N = blockDim.x * gridDim.x;

 buffer_s[threadIdx.y][threadIdx.x] = matrix[i + j * N];
 __syncthreads();

 i = threadIdx.x + blockIdx.y * blockDim.x;
```

```
  j = threadIdx.y + blockIdx.x * blockDim.y;
  matrixT[i + j * N] = buffer_s[threadIdx.x][threadIdx.y];
}

__global__ void gTrans1(float *matrix, float *matrixT) {
  extern __shared__ float buffer[];
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  int j = threadIdx.y + blockIdx.y * blockDim.y;
  int N = blockDim.x * gridDim.x;

  buffer[threadIdx.y + threadIdx.x * blockDim.y] = matrix[i + j * N];
  __syncthreads();

  i = threadIdx.x + blockIdx.y * blockDim.x;
  j = threadIdx.y + blockIdx.x * blockDim.y;
  matrixT[i + j * N] = buffer[threadIdx.x + threadIdx.y * blockDim.x];
}

__global__ void gTrans(float *matrix, float *matrixT) {
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  int j = threadIdx.y + blockIdx.y * blockDim.y;
  int N = blockDim.x * gridDim.x;
  matrixT[j + i * N] = matrix[i + j * N];
}

__global__ void gInit(float *matrix) {
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  int j = threadIdx.y + blockIdx.y * blockDim.y;
  int N = blockDim.x * gridDim.x;
  matrix[i + j * N] = (float)(i + j * N);
}
```

```
void Output(float *a, int N) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++)
      fprintf(stdout, "%g\t", a[j + i * N]);
    fprintf(stdout, "\n");
  }
  fprintf(stdout, "\n\n\n");
}

int main(int argc, char *argv[]) {
  int N;
  int dimB;
  if (argc == 3) {
    N = atoi(argv[1]);
    dimB = atoi(argv[2]);
  } else {
    N = 1 << 10;
    dimB = 1 << 5;
  }
  float elapsedTime;
  int dimG = N / dimB;
  float *mtx, *mtxT,*mtxT1,*mtxT2,*mtxT3,*matrix, *matrixT;
  cudaEvent_t start, stop;
  cudaEventCreate(&start);
  cudaEventCreate(&stop);

  matrix = (float *)calloc(N * N,sizeof(float));
  matrixT = (float *)calloc(N * N, sizeof(float));
  CUDA_CHECK_RETURN(cudaMalloc((void **)&mtx, N * N * sizeof(float)));
  CUDA_CHECK_RETURN(cudaMalloc((void **)&mtxT, N * N * sizeof(float)));
  CUDA_CHECK_RETURN(cudaMalloc((void **)&mtxT1, N * N * sizeof(float)));
  CUDA_CHECK_RETURN(cudaMalloc((void **)&mtxT2, N * N * sizeof(float)));
```

```
CUDA_CHECK_RETURN(cudaMalloc((void **)&mtxT3, N * N * sizeof(float)));
gInit<<<dim3(dimG,dimG),dim3(dimB,dimB) >>>(mtx);
CUDA_CHECK_RETURN(cudaDeviceSynchronize());
CUDA_CHECK_RETURN(cudaMemcpy(matrix, mtx, N * N * sizeof(float), cudaMemcpyDeviceToHost));

cudaEventRecord(start, 0);
gTrans<<<dim3(dimG,dimG),dim3(dimB,dimB)>>>(mtx, mtxT);
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&elapsedTime, start,stop);
printf("gTranspose0 took %gms\n", elapsedTime);
CUDA_CHECK_RETURN(cudaMemcpy(mtx, matrix, N * N * sizeof(float), cudaMemcpyHostToDevice));

cudaEventRecord(start, 0);
gTrans1<<<dim3(dimG,dimG),dim3(dimB,dimB),dimB*dimB*sizeof(float)>>>(mtx, mtxT1);
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&elapsedTime, start,stop);
printf("gTranspose1 took %gms\n", elapsedTime);
CUDA_CHECK_RETURN(cudaMemcpy(mtx, matrix, N * N * sizeof(float), cudaMemcpyHostToDevice));

cudaEventRecord(start, 0);
gTrans2<<<dim3(dimG,dimG),dim3(dimB,dimB)>>>(mtx, mtxT2);
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&elapsedTime, start,stop);
printf("gTranspose2 took %gms\n", elapsedTime);
CUDA_CHECK_RETURN(cudaMemcpy(mtx, matrix, N * N * sizeof(float), cudaMemcpyHostToDevice));

cudaEventRecord(start, 0);
gTrans3<<<dim3(dimG,dimG),dim3(dimB,dimB)>>>(mtx, mtxT3);
cudaEventRecord(stop, 0);
```

```
  cudaEventSynchronize(stop);
  cudaEventElapsedTime(&elapsedTime, start,stop);
  printf("gTranspose3 took %gms\n", elapsedTime);
  CUDA_CHECK_RETURN(cudaGetLastError());
  cudaEventDestroy(start);
  cudaEventDestroy(stop);
  free(matrix);
  free(matrixT);
  cudaFree(mtx);
  cudaFree(mtxT);
  cudaFree(mtxT1);
  cudaFree(mtxT2);
  cudaFree(mtxT3);
  return 0;
}
```