

**Задание:**

Реализовать транспонирование матрицы размерностью N\*K, где = 8\*2^12, число нитей взято threadsPerBlock = 128, использования разделяемой памяти, с разделяемой памятью без разрешения конфликта банков и с разрешением конфликта банков. Сравнить время выполнения соответствующих ядер на GPU. Для всех трёх случаев определить эффективность использования разделяемой памяти с помощью метрик nvprof или ncu.

**Цель:** приобретение навыков использования разделяемой памяти.

**Выполнение работы:**

Для выполнения работы была написана программа, реализующая транспонирование матрицы тремя методами:

* без использования shared памяти
* с использованием shared памяти и с возникновение конфликта банков
* с использование shared памяти и решением конфликта памяти

| #include <iostream> #include <cstdlib> #include "cuda\_runtime.h" #include "device\_launch\_parameters.h"  using namespace std;  #define CUDA\_NUM 32  \_\_global\_\_ void gBase\_Transposition(float \*matrix, float \*result, const int N, const int K) {  unsigned int k = threadIdx.x + blockIdx.x \* blockDim.x;  unsigned int n = threadIdx.y + blockIdx.y \* blockDim.y;  result[n + k \* N] = matrix[k + n \* K]; } \_\_global\_\_ void gShared\_Transposition\_Wrong(float \*matrix, float \*result, const int N, const int K) {  \_\_shared\_\_ float shared[CUDA\_NUM][CUDA\_NUM];  unsigned int k = threadIdx.x + blockIdx.x \* blockDim.x;  unsigned int n = threadIdx.y + blockIdx.y \* blockDim.y;   shared[threadIdx.y][threadIdx.x] = matrix[K + n \* N];  \_\_syncthreads();   k = threadIdx.x + blockIdx.y \* blockDim.x;  n = threadIdx.y + blockIdx.x \* blockDim.y;  result[k + n \* N] = shared[threadIdx.x][threadIdx.y]; } \_\_global\_\_ void gShared\_Transposition(float \*matrix, float \*result, const int N, const int K) {  \_\_shared\_\_ float shared[CUDA\_NUM][CUDA\_NUM + 1];  unsigned int k = threadIdx.x + blockIdx.x \* blockDim.x;  unsigned int n = threadIdx.y + blockIdx.y \* blockDim.y;   shared[threadIdx.y][threadIdx.x] = matrix[K + n \* N];  \_\_syncthreads();   k = threadIdx.x + blockIdx.y \* blockDim.x;  n = threadIdx.y + blockIdx.x \* blockDim.y;  result[k + n \* N] = shared[threadIdx.x][threadIdx.y]; } void MatrixShow(const int N, const int K, const float \*Matrix) {  cout << endl;  for (long long i = 0; i < K; ++i) {  for (long long j = 0; j < N; ++j) {  cout << Matrix[j + i \* N] << " ";  }  cout << endl;  }  cout << endl; }  int main() {  const int num = 1 << 12;  int N = 8 \* num, K = 8 \* num, threadsPerBlock = 128;  float \*GPU\_pre\_matrix, \*local\_pre\_matrix, \*GPU\_after\_matrix, \*local\_after\_matrix, elapsedTime;  cudaEvent\_t start, stop;  cudaEventCreate(&start);  cudaEventCreate(&stop);   /\* простое транспонирование \*/   cudaMalloc((void \*\*) &GPU\_pre\_matrix, N \* K \* sizeof(float));  cudaMalloc((void \*\*) &GPU\_after\_matrix, N \* K \* sizeof(float));   local\_pre\_matrix = (float \*) calloc(N \* K, sizeof(float));  local\_after\_matrix = (float \*) calloc(N \* K, sizeof(float));   for (int i = 0; i < N; ++i) {  for (int j = 0; j < K; ++j) {  local\_pre\_matrix[j + i \* K] = j + i \* K + 1;  }  }   cudaMemcpy(GPU\_pre\_matrix, local\_pre\_matrix, K \* N \* sizeof(float), cudaMemcpyHostToDevice);   cudaEventRecord(start, nullptr);  gBase\_Transposition <<< dim3(K / threadsPerBlock,N / threadsPerBlock),  dim3(threadsPerBlock, threadsPerBlock) >>>  (GPU\_pre\_matrix, GPU\_after\_matrix, N, K);  cudaDeviceSynchronize();  cudaEventRecord(stop, nullptr);  cudaEventSynchronize(stop);   cudaMemcpy(local\_after\_matrix, GPU\_after\_matrix, K \* N \* sizeof(float), cudaMemcpyDeviceToHost);  cudaEventElapsedTime(&elapsedTime, start, stop);   cout<<"1st method Matrix: "<<endl;   cout << "gBase\_Transposition:\n\t"  << elapsedTime  << endl;   cudaFree(GPU\_after\_matrix);  free(local\_after\_matrix);   /\* транспонирование без решения проблемы конфликта банков \*/   cudaMalloc((void \*\*) &GPU\_after\_matrix, N \* K \* sizeof(float));  local\_after\_matrix = (float \*) calloc(N \* K, sizeof(float));   cudaEventRecord(start, nullptr);  gShared\_Transposition\_Wrong <<< dim3(K / threadsPerBlock, N / threadsPerBlock),  dim3(threadsPerBlock,threadsPerBlock) >>>  (GPU\_pre\_matrix, GPU\_after\_matrix, N, K);   cudaDeviceSynchronize();  cudaEventRecord(stop, nullptr);  cudaEventSynchronize(stop);   cudaMemcpy(local\_after\_matrix, GPU\_after\_matrix, K \* N \* sizeof(float), cudaMemcpyDeviceToHost);  cudaEventElapsedTime(&elapsedTime, start, stop);   cout<<"2st method Matrix: "<<endl;  cout << "gShared\_Transposition\_Wrong:\n\t"  << elapsedTime  << endl;   cudaFree(GPU\_after\_matrix);  free(local\_after\_matrix);   /\* транспонирование с решением проблемы конфликта банков \*/   cudaMalloc((void \*\*) &GPU\_after\_matrix, N \* K \* sizeof(float));  local\_after\_matrix = (float \*) calloc(N \* K, sizeof(float));   cudaEventRecord(start, nullptr);  gShared\_Transposition <<< dim3(K / threadsPerBlock, N / threadsPerBlock),  dim3(threadsPerBlock,threadsPerBlock) >>>  (GPU\_pre\_matrix, GPU\_after\_matrix, N, K);   cudaDeviceSynchronize();  cudaEventRecord(stop, nullptr);  cudaEventSynchronize(stop);   cudaMemcpy(local\_after\_matrix, GPU\_after\_matrix, K \* N \* sizeof(float), cudaMemcpyDeviceToHost);  cudaEventElapsedTime(&elapsedTime, start, stop);   cout<<"3st method Matrix: "<<endl;   cout << "gShared\_Transposition:\n\t"  << elapsedTime  << endl;   cudaFree(GPU\_pre\_matrix);  cudaFree(GPU\_after\_matrix);  free(local\_pre\_matrix);  free(local\_after\_matrix);   return 0; } |
| --- |

Листинг 1 – main.cu

Команда компиляции и результат работы программы:

| D:\Projects\CUDA\_CMake\cmake-build-debug\LR05\_GPU.exe 1st method Matrix:  gBase\_Transposition:  0.035904  2st method Matrix:  gShared\_Transposition\_Wrong:  0.015456  3st method Matrix:  gShared\_Transposition:  0.010208 |
| --- |

Результат nsys profile:

| Time (%) Total Time (ns) Num Calls Avg (ns) Med (ns) Min (ns) Max (ns) StdDev (ns) Name  -------- --------------- --------- ------------- ------------- ----------- ----------- ------------ ----------------------  54,0 3 059 585 608 39 78 450 913,0 100 129 119,0 225 584 154 999 628 42 116 780,0 poll  44,0 2 500 442 326 5 500 088 465,0 500 079 638,0 500 068 711 500 142 513 30 598,0 pthread\_cond\_timedwait  1,0 70 727 109 452 156 475,0 8 440,0 1 088 8 197 867 485 009,0 ioctl  0,0 864 678 25 34 587,0 6 079,0 3 341 558 250 110 154,0 mmap64  0,0 612 116 9 68 012,0 79 945,0 9 944 115 837 34 543,0 sem\_timedwait  0,0 146 898 43 3 416,0 3 075,0 1 218 7 657 1 382,0 open64  0,0 125 941 31 4 062,0 2 251,0 1 023 21 602 4 797,0 fopen  0,0 98 791 19 5 199,0 2 715,0 1 093 31 494 6 935,0 mmap  0,0 73 270 3 24 423,0 23 037,0 17 773 32 460 7 441,0 pthread\_create  0,0 64 339 28 2 297,0 2 271,0 1 143 7 318 1 257,0 munmap  0,0 29 923 1 29 923,0 29 923,0 29 923 29 923 0,0 fgets  0,0 18 817 2 9 408,0 9 408,0 5 945 12 872 4 898,0 pipe2  0,0 16 016 5 3 203,0 3 770,0 1 207 4 070 1 161,0 open  0,0 12 865 2 6 432,0 6 432,0 5 101 7 764 1 883,0 fread  0,0 12 345 7 1 763,0 1 570,0 1 013 4 018 1 051,0 fclose  0,0 12 047 1 12 047,0 12 047,0 12 047 12 047 0,0 connect  0,0 10 922 2 5 461,0 5 461,0 4 775 6 147 970,0 socket  0,0 8 144 2 4 072,0 4 072,0 2 519 5 625 2 196,0 fwrite  0,0 7 057 5 1 411,0 1 417,0 1 108 1 796 297,0 write  0,0 5 601 3 1 867,0 1 866,0 1 429 2 306 438,0 read  0,0 5 561 2 2 780,0 2 780,0 1 395 4 166 1 959,0 putc  0,0 3 212 1 3 212,0 3 212,0 3 212 3 212 0,0 listen  0,0 2 557 2 1 278,0 1 278,0 1 057 1 500 313,0 pthread\_cond\_broadcast  0,0 1 746 1 1 746,0 1 746,0 1 746 1 746 0,0 bind  [5/8] Executing 'cuda\_api\_sum' stats report  Time (%) Total Time (ns) Num Calls Avg (ns) Med (ns) Min (ns) Max (ns) StdDev (ns) Name  -------- --------------- --------- ------------ ------------ -------- ---------- ------------ ----------------------  98,0 78 365 928 2 39 182 964,0 39 182 964,0 443 78 365 485 55 412 452,0 cudaEventCreate  0,0 580 438 3 193 479,0 17 721,0 12 771 549 946 308 719,0 cudaLaunchKernel  0,0 427 512 4 106 878,0 104 053,0 77 510 141 896 29 859,0 cudaMalloc  0,0 38 307 6 6 384,0 4 116,0 2 371 19 192 6 425,0 cudaEventRecord  0,0 19 130 4 4 782,0 2 460,0 660 13 550 5 968,0 cudaMemcpy  0,0 8 104 3 2 701,0 2 456,0 2 178 3 470 680,0 cudaEventSynchronize  0,0 7 482 3 2 494,0 1 892,0 1 845 3 745 1 083,0 cudaDeviceSynchronize  0,0 1 752 4 438,0 341,0 137 933 346,0 cudaFree  0,0 985 1 985,0 985,0 985 985 0,0 cuModuleGetLoadingMode |
| --- |

По результатам работы программы можно сделать вывод, что использование shared памяти положительно влияет на скорость выполнения задачи, но в случае не решенной проблемы конфликта банков, а именно когда происходит запись в одну и туже ячейку идет потеря производительности. С использование shared памяти +1 эта проблема исчезает.

**Вывод:**   
В ходе выполнения работы, была исследована и применена работа с глобальной памятью графического процессора (GPU) с использованием технологии CUDA. В ходе работы мы ознакомились с работой с shared памятью и разрешением конфликта банков памяти. в прошлый раз данные были некорректны из-за проблем работы CUDA из под WSL, переход на Ubuntu решил эту проблему