Министерство цифрового развития, связи
и массовых коммуникаций Российской Федерации

Сибирский государственный университет
телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

ЛАБОРАТОРНАЯ РАБОТА №5

По дисциплине: «Программирование графических процессоров»

Выполнили:

Студенты 3 курса группы ИП-111
Корнилов А.А.,
Попов М.И.,
Толкач А.А.

Проверил:

Профессор кафедры ПМиК
Малков Е.А.

Новосибирск, 2024

Задание:

Реализуйте транспонирование матрицы размерностью N*K без использования разделяемой памяти, с разделяемой памятью без разрешения конфликта банков и с разрешением конфликта банков. Сравните время выполнения соответствующих ядер на GPU. Для всех трёх случаев определите эффективность использования разделяемой памяти с помощью метрик nvprof или ncu.

Цель: приобретение навыков использования разделяемой памяти.

Выполнение работы:

Для выполнения работы была написана программа которая запускает на транспонирования 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;

    buffer[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] = buffer[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;

    buffer[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] = buffer[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 N = 8, K = 8, threadsPerBlock = 8;
    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));

    cout<<"Initial Matrix: "<<endl;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < K; ++j) {
            local_pre_matrix[j + i * K] = j + i * K + 1;
            cout << local_pre_matrix[j + i * K] << " ";
        }
        cout<< endl;
    }
    cout<< endl;

    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;
    MatrixShow(N, K, local_after_matrix);

    cout << "gBase_Transposition:\n\t"
         << elapsedTime
         << endl;

    cudaFree(GPU_after_matrix);
    free(local_after_matrix);

    /* траспонирование без решениея 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;
    MatrixShow(N, K, local_after_matrix);

    cout << "gShared_Transposition_Wrong:\n\t"
         << elapsedTime
         << endl;

    cudaFree(GPU_after_matrix);
    free(local_after_matrix);

    /* траспонирование 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;
    MatrixShow(N, K, local_after_matrix);

    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 – программа LR03_1.cu

Команда компиляции и результат работы программы:

```
C:\Windows\system32\wsl.exe --distribution Ubuntu --exec /bin/bash -c "cd
/mnt/d/Projects/CUDA_CMake/cmake-build-debug &&
/mnt/d/Projects/CUDA_CMake/cmake-build-debug/LR05_GPU"

Initial Matrix:
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
```

```
1st method Matrix:

1 9 17 25 33 41 49 57
2 10 18 26 34 42 50 58
3 11 19 27 35 43 51 59
4 12 20 28 36 44 52 60
5 13 21 29 37 45 53 61
6 14 22 30 38 46 54 62
7 15 23 31 39 47 55 63
8 16 24 32 40 48 56 64

gBase_Transposition:
        0.126976
2st method Matrix:

9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0

gShared_Transposition_Wrong:
        0.101056
3st method Matrix:

9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0
9 17 25 33 41 49 57 0

gShared_Transposition:
        0.079328
```

```
Process finished with exit code 0
```

Результат nvprof:

```
==34963== Profiling application: ./LR05_GPU
==34963== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   27.44%  3.4240us         3  1.1410us     992ns  1.2160us  [CUDA memcpy DtoH]
                   21.54%  2.6880us         1  2.6880us  2.6880us  2.6880us
gShared_Transposition_Wrong(float*, float*, int, int)
                   20.51%  2.5590us         1  2.5590us  2.5590us  2.5590us  gBase_Transposition(float*, float*,
int, int)
                   19.49%  2.4320us         1  2.4320us  2.4320us  2.4320us  gShared_Transposition(float*,
float*, int, int)
                   11.03%  1.3760us         1  1.3760us  1.3760us  1.3760us  [CUDA memcpy HtoD]
      API calls:   98.51%  765.15ms         2  382.57ms  1.3020us  765.15ms  cudaEventCreate
                    0.66%  5.1249ms         4  1.2812ms  10.419us  5.0814ms  cudaFree
                    0.59%  4.5617ms         1  4.5617ms  4.5617ms  4.5617ms  cuDeviceGetPCIBusId
                    0.11%  846.88us         4  211.72us  6.4010us  820.76us  cudaMalloc
                    0.04%  345.82us         4  86.454us  79.456us  95.065us  cudaMemcpy
                    0.03%  267.57us         3  89.190us  55.983us  139.77us  cudaEventSynchronize
                    0.02%  190.10us         3  63.366us  41.827us  80.408us  cudaDeviceSynchronize
                    0.02%  119.48us         3  39.826us  33.261us  51.274us  cudaLaunchKernel
                    0.01%  94.002us         6  15.667us  11.802us  22.641us  cudaEventRecord
                    0.00%  30.956us       101     306ns     170ns  3.1860us  cuDeviceGetAttribute
                    0.00%  7.4030us         3  2.4670us  2.1540us  2.7250us  cudaEventElapsedTime
                    0.00%  2.6450us         3     881ns     361ns  1.3320us  cuDeviceGetCount
                    0.00%  1.5430us         2     771ns     421ns  1.1220us  cuDeviceGet
                    0.00%  1.3220us         1  1.3220us  1.3220us  1.3220us  cuDeviceGetName
                    0.00%     591ns         1     591ns     591ns     591ns  cuDeviceTotalMem
                    0.00%     361ns         1     361ns     361ns     361ns  cuDeviceGetUuid
```

По результату работы программы можно сделать вывод, что использование shared памяти действительно делает программы быстрее, но в случае не решеной проблемы конфликта банков, а именно когда происходит запись в одну и туже ячейку идет потеря производительности. С использование shared памяти +1 эта проблема исчезает. К сожалению провести профилирование использования памяти не получилось из-за несовместимости оборудования:

```
==34963== Warning: Unified Memory Profiling is not supported on the current configuration because a pair of devices without peer-to-peer support is detected on this multi-GPU setup. When peer mappings are not available, system falls back to using zero-copy memory. It can cause kernels, which access unified memory, to run slower. More
```

details can be found at: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-managed-memory

Также могу отметить что если при сборке проекта через CMake, при компиляции указывать отдельно архитектуру (параметр CUDA_ARCHITECTURES) используемую nvcc, и если ставить версию выше чем 62 shared память не получиться использовать и в результате матрицы зануляются.