Федеральное агентство связи(Россвязь)

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики». (СибГУТИ)

Кафедра прикладной математики и кибернетики (ПМиК)

Отчет по лабораторной работе №4

по дисциплине «Програамирование Графических Процессоров»

Выполнил: студент группы ИП-712

Алексеев С.В.

Проверил:

Нужнов А.В.

Новосибирск – 2020 г.

**Содержание**

[Цель работы 3](#_Toc494217527)

[Выполнение работы 3](#_Toc494217528)

[Вывод 3](#_Toc494217529)

[Приложение 4](#_Toc494217530)

# **Цель работы**

Требуется написать программу транспонирования матриц, реализующую следующие алгоритмы:

* без использования разделяемой памяти (0);
* наивный алгоритм с использованием разделяемой памяти (1);
* алгоритм с разрешением конфликта банков разделяемой памяти (2).

Необходимо провести профилирование программы с использованием nvprof и nvpp, сравнить время выполнения ядер, реализующих разные алгоритмы, и оценить эффективность использования разделяемой памяти.

# **Выполнение работы**

Работать будем на предоставленном сервере.

Все три алгоритма подробно разобраны в материалах лекции 4.

Результаты работы программы (код см. в приложении А) следующие:

|  |  |  |  |
| --- | --- | --- | --- |
| Размер матрицы | Алгоритм “0”, мкс | Алгоритм “1”, мкс | Алгоритм “2”, мкс |
| 32 | 2.816 | 2.688 | 2.24 |
| 64 | 2.784 | 2.656 | 2.144 |
| 128 | 7.744 | 6.624 | 4.5760 |
| 256 | 22.241 | 19.745 | 12.64 |
| 512 | 86.115 | 78.659 | 50.402 |
| 1024 | 337.84 | 302.19 | 192.9 |
| 2048 | 1339.5 | 1204.9 | 766.94 |
| 4096 | 5357.3 | 4809.5 | 3053.4 |
| 8192 | 21438 | 19.386 | 12357 |

Таблица 1. Результаты работы программы.

# **Вывод**

Эксперименты проводились на видеокарте на сервере, предоставленном преподавателем.

Общая тенденция очевидна –

Прирост производительности при использовании наивного алгоритма с использованием разделяемой памяти в среднем около 8%. Если Использовать алгоритм с разрешением конфликта банков, то прирост производительности около 40%.

# **Приложение**

Приложение А. Листинг программы.

#include <cuda.h>

#include <stdio.h>

#define SH\_DIM 32

\_\_global\_\_ void gInitializeStorage(float\* storage\_d, int N)

{

int i = threadIdx.x + blockIdx.x \* blockDim.x;

int j = threadIdx.y + blockIdx.y \* blockDim.y;

// цикл позволяет выбирать произвольное значение N, меньше

// количества потоков в блоке, и больше общего количества потоков

for (int iy = j; iy < N; iy += gridDim.y \* blockDim.y)

{

for (int ix = i; ix < N; ix += gridDim.x \* blockDim.x)

{

storage\_d[ix + iy \* N] = ix + iy \* N;

}

}

}

\_\_global\_\_ void gTranspose0(float\* storage\_d, float\* storage\_d\_t, int N)

{

int i = threadIdx.x + blockIdx.x \* blockDim.x;

int j = threadIdx.y + blockIdx.y \* blockDim.y;

for (int iy = j; iy < N; iy += gridDim.y \* blockDim.y)

{

for (int ix = i; ix < N; ix += gridDim.x \* blockDim.x)

{

storage\_d\_t[iy + ix \* N] = storage\_d[ix + iy \* N];

}

}

\_\_syncthreads();

}

\_\_global\_\_ void gTranspose1(float\* storage\_d, float\* storage\_d\_t, int N)

{

\_\_shared\_\_ float buffer[SH\_DIM][SH\_DIM];

int i = threadIdx.x + blockIdx.x \* blockDim.x;

int j = threadIdx.y + blockIdx.y \* blockDim.y;

for (int iy = j; iy < N; iy += gridDim.y \* blockDim.y)

{

for (int ix = i; ix < N; ix += gridDim.x \* blockDim.x)

{

buffer[threadIdx.y][threadIdx.x] = storage\_d[ix + iy \* N];

}

}

\_\_syncthreads();

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

j = threadIdx.y + blockIdx.x \* blockDim.y;

for (int iy = j; iy < N; iy += gridDim.y \* blockDim.y)

{

for (int ix = i; ix < N; ix += gridDim.x \* blockDim.x)

{

storage\_d\_t[ix + iy \* N] = buffer[threadIdx.x][threadIdx.y];

}

}

\_\_syncthreads();

}

\_\_global\_\_ void gTranspose2(float\* storage\_d, float\* storage\_d\_t, int N)

{

\_\_shared\_\_ float buffer[SH\_DIM][SH\_DIM + 1];

int i = threadIdx.x + blockIdx.x \* blockDim.x;

int j = threadIdx.y + blockIdx.y \* blockDim.y;

for (int iy = j; iy < N; iy += gridDim.y \* blockDim.y)

{

for (int ix = i; ix < N; ix += gridDim.x \* blockDim.x)

{

buffer[threadIdx.y][threadIdx.x] = storage\_d[ix + iy \* N];

}

}

\_\_syncthreads();

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

j = threadIdx.y + blockIdx.x \* blockDim.y;

for (int iy = j; iy < N; iy += gridDim.y \* blockDim.y)

{

for (int ix = i; ix < N; ix += gridDim.x \* blockDim.x)

{

storage\_d\_t[ix + iy \* N] = buffer[threadIdx.x][threadIdx.y];

}

}

\_\_syncthreads();

}

void Output(float\* a, int N)

{

for (int i = 0; i < N; i += N / 4)

{

for (int j = 0; j < N; j += N / 4)

{

printf("%10.0f\t", a[j + i \* N]);

}

printf("\n");

}

}

int main(int argc, char\* argv[])

{

if (argc < 2)

{

fprintf(stderr, "USAGE: matrix <dimension>\n");

return -1;

}

int N = atoi(argv[1]);

const int max\_size = 1024;

int size = N / 32 + (N % 32 > 0);

int dim\_of\_blocks = (size > max\_size) ? max\_size : size;

int dim\_of\_threads = 32;

float \*storage\_d, \*storage\_d\_t, \*storage\_h;

cudaMalloc((void\*\*)&storage\_d, N \* N \* sizeof(float));

cudaMalloc((void\*\*)&storage\_d\_t, N \* N \* sizeof(float));

storage\_h = (float\*)calloc(N \* N, sizeof(float));

gInitializeStorage<<<dim3(dim\_of\_blocks, dim\_of\_blocks), dim3(dim\_of\_threads, dim\_of\_threads)>>>(storage\_d, N);

cudaThreadSynchronize();

// cudaMemcpy(storage\_h, storage\_d, N \* N \* sizeof(float), cudaMemcpyDeviceToHost);

// Output(storage\_h, N);

gTranspose0<<<dim3(dim\_of\_blocks, dim\_of\_blocks), dim3(dim\_of\_threads, dim\_of\_threads)>>>(storage\_d, storage\_d\_t, N);

cudaThreadSynchronize();

// cudaMemcpy(storage\_h, storage\_d\_t, N\*N\*sizeof(float), cudaMemcpyDeviceToHost);

// Output(storage\_h, N);

gTranspose1<<<dim3(dim\_of\_blocks, dim\_of\_blocks), dim3(dim\_of\_threads, dim\_of\_threads)>>>(storage\_d, storage\_d\_t, N);

cudaThreadSynchronize();

// cudaMemcpy(storage\_h, storage\_d\_t, N\*N\*sizeof(float), cudaMemcpyDeviceToHost);

// Output(storage\_h, N);

gTranspose2<<<dim3(dim\_of\_blocks, dim\_of\_blocks), dim3(dim\_of\_threads, dim\_of\_threads)>>>(storage\_d, storage\_d\_t, N);

cudaThreadSynchronize();

// cudaMemcpy(storage\_h, storage\_d\_t, N \* N \* sizeof(float), cudaMemcpyDeviceToHost);

// Output(storage\_h, N);

cudaFree(storage\_d);

cudaFree(storage\_d\_t);

free(storage\_h);

return 0;

}