МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

**спецкурс «Параллельные и распределенные вычисления»**

**ОТЧЕТ**

**Лабораторная работа № 2**

**«Использование различных типов памяти»**

Выполнил: Королев П.А.

Группа: М80-107М-21

Преподаватель: Семенов С. А.

Москва, 2021

**Содержание**

[1. Постановка задачи 2](#_gjdgxs)

[2. Описание решения 2](#_30j0zll)

[3. Аппаратное обеспечение и ПО 2](#_1fob9te)

[4. Основные моменты кода 2](#_3znysh7)

[5. Результат работы программы 2](#_2et92p0)

[6. Сравнение скорости выполнения на CPU и GPU 2](#_tyjcwt)

[7. Выводы 3](#_3dy6vkm)

[8. Приложения 3](#_1t3h5sf)

# **1. Постановка задачи**

Вариант 3 - Разложение матрицы LU.

# **2. Описание решения**

Будет посчитана результирующая матрица через произведение двух матриц, A=LU, где L— нижняя треугольная матрица, а

U — верхняя треугольная матрица.

# **3. Аппаратное обеспечение и ПО**

Процессор Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz

Оперативная память 8,00 ГБ

Тип системы 64-разрядная операционная система, процессор x64

Ядер : 4

Виртуализация: Включена

Tesla K80, 11441 MiB

Количество потоковых процессоров:

2496из 10752 (GA102)

Частота ядра:

562 МГциз 2233 (Playstation 5 GPU)

Частота в режиме Boost:

824 МГциз 2903 (Radeon Pro W6600)

Количество транзисторов:

7,100 млниз 14400 (GeForce GTX 1080 SLI (мобильная))

Программное обеспечение:

Google Collab

# **4. Основные моменты кода**

\_\_global\_\_ void matrixLU(float \*lu\_matrix, const int N, bool \* flag)

{

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

if(id >=N) return;

for(int j = id; j < N; j+=blockDim.x\*gridDim.x){

if(tex1Dfetch(matrix\_t, j) == 0) \*flag = false;

\*(lu\_matrix+j) = (float) tex1Dfetch(matrix\_t, j);

}

\_\_syncthreads();

if(!\*flag) return;

for(int r = 1; r < N; r++){

int temp\_id = id;

while(temp\_id < r){

temp\_id+=blockDim.x\*gridDim.x;

}

if(temp\_id >= N) return ;

\_\_syncthreads();

//L colume r-1 -compute

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

{

//l[i][r-1]

\*(lu\_matrix+i\*N + r-1) = (float) tex1Dfetch(matrix\_t, i\*N+r-1);

for(int k = 0; k < r-1; k++)

{

\*(lu\_matrix+(i)\*N+r-1) -= (\*(lu\_matrix+i\*N+k)) \* (\*(lu\_matrix+k\*N + r-1));

}

\*(lu\_matrix+(i)\*N+r-1) /= \*(lu\_matrix+(r-1)\*N+r-1);

}

\_\_syncthreads();

//U row r - compute

for(int j = temp\_id; j < N; j+=blockDim.x\*gridDim.x)

{

//u[r][j]

\*(lu\_matrix+r\*N+j) = (float) tex1Dfetch(matrix\_t, r\*N+j);

for(int k = 0; k <r; k++){

\*(lu\_matrix+r\*N+j) -= (\*(lu\_matrix+r\*N+k)) \* (\*(lu\_matrix+k\*N + j));

}

if(tex1Dfetch(matrix\_t, r\*N+j) == 0) \*flag = false;

}

\_\_syncthreads();

if(!\*flag) return;

}

}

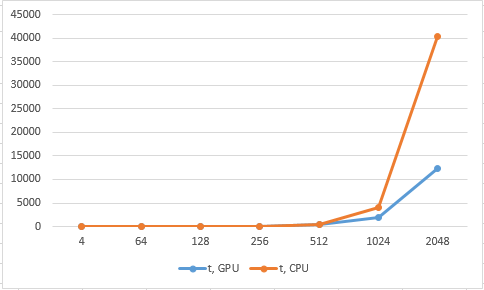
# **5. Результат работы программы**

| Для GPU | Для CPU |
| --- | --- |
| N = 2048 | N = 2048 |
| t = 12345ms | t = 40301ms |

# 

# **6. Сравнение скорости выполнения на CPU и GPU**

| **N** | **GPU**  **время выполнения, мс** | **CPU**  **время выполнения, мс** | **tCPU/tGPU** |
| --- | --- | --- | --- |
| 4 | 0 | 0 | 0 |
| 64 | 4 | 0 | 0 |
| 128 | 17 | 6 | 0,36 |
| 256 | 69 | 33 | 0,48 |
| 512 | 356 | 431 | 1,21 |
| 1024 | 1840 | 4051 | 2,21 |
| 2048 | 12345 | 40301 | 3,27 |



# **7. Выводы**

В Лабораторной работе №2 проведен анализ работы различных программ по решению задачи разложения матрицы LU.

При сравнении реализации задачи на CPU и GPU, было выявлено, что скорость выполнения программы на GPU сильно выше скорости выполнения программы на CPU, начиная с матрицы 1024х1024.

**8. Приложения**

Ссылка на Github: <https://github.com/PitKoro/PaDC/blob/main/lab2/main.cu>

Код на CPU

int cpu\_decomposition(int \*matrix, int N, float \*lu\_matrix)

{

clock\_t start = clock();

for(int j = 0; j < N; j++){

if((\*(matrix+j)) == 0) return -1;

\*(lu\_matrix+j) = (float) \*(matrix+j);

}

for(int r = 1; r < N; r++){

//L colume r-1 -compute

for(int i = r; i < N; i++)

{

//l[i][r-1]

\*(lu\_matrix+i\*N + r-1) = (float) \*(matrix+i\*N+r-1);

for(int k = 0; k < r-1; k++)

{

\*(lu\_matrix+(i)\*N+r-1) -= (\*(lu\_matrix+i\*N+k)) \* (\*(lu\_matrix+k\*N + r-1));

}

\*(lu\_matrix+(i)\*N+r-1) /= \*(lu\_matrix+(r-1)\*N+r-1);

}

//U row r - compute

for(int j = r; j < N; j++)

{

//u[r][j]

\*(lu\_matrix+r\*N+j) = (float) \*(matrix+r\*N+j);

for(int k = 0; k <r; k++){

\*(lu\_matrix+r\*N+j) -= (\*(lu\_matrix+r\*N+k)) \* (\*(lu\_matrix+k\*N + j));

}

if((\*(matrix+(r)\*N+j)) == 0)

return -1;

}

}

return (int)(clock() - start)/ (CLOCKS\_PER\_SEC / 1000);

}

Код на GPU

\_\_global\_\_ void matrixLU(float \*lu\_matrix, const int N, bool \* flag)

{

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

if(id >=N) return;

for(int j = id; j < N; j+=blockDim.x\*gridDim.x){

if(tex1Dfetch(matrix\_t, j) == 0) \*flag = false;

\*(lu\_matrix+j) = (float) tex1Dfetch(matrix\_t, j);

}

\_\_syncthreads();

if(!\*flag) return;

for(int r = 1; r < N; r++){

int temp\_id = id;

while(temp\_id < r){

temp\_id+=blockDim.x\*gridDim.x;

}

if(temp\_id >= N) return ;

\_\_syncthreads();

//L colume r-1 -compute

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

{

//l[i][r-1]

\*(lu\_matrix+i\*N + r-1) = (float) tex1Dfetch(matrix\_t, i\*N+r-1);

for(int k = 0; k < r-1; k++)

{

\*(lu\_matrix+(i)\*N+r-1) -= (\*(lu\_matrix+i\*N+k)) \* (\*(lu\_matrix+k\*N + r-1));

}

\*(lu\_matrix+(i)\*N+r-1) /= \*(lu\_matrix+(r-1)\*N+r-1);

}

\_\_syncthreads();

//U row r - compute

for(int j = temp\_id; j < N; j+=blockDim.x\*gridDim.x)

{

//u[r][j]

\*(lu\_matrix+r\*N+j) = (float) tex1Dfetch(matrix\_t, r\*N+j);

for(int k = 0; k <r; k++){

\*(lu\_matrix+r\*N+j) -= (\*(lu\_matrix+r\*N+k)) \* (\*(lu\_matrix+k\*N + j));

}

if(tex1Dfetch(matrix\_t, r\*N+j) == 0) \*flag = false;

}

\_\_syncthreads();

if(!\*flag) return;

}

}

int gpu\_decomposition(int \*matrix, int N, float \*lu\_matrix)

{

int \* dev\_matrix, threads;

float \* dev\_lu\_matrix, elapsed\_time;

cudaEvent\_t start, stop;

bool flag = true, \* dev\_flag;

cudaEventCreate(&start);

cudaEventCreate(&stop);

cudaMalloc(&dev\_flag, sizeof(bool));

cudaMalloc(&dev\_matrix, sizeof(int) \* N \* N);

cudaMalloc(&dev\_lu\_matrix, sizeof(float) \* N \* N);

cudaMemcpy(dev\_matrix, matrix, sizeof(int) \* N \* N, cudaMemcpyHostToDevice);

cudaMemcpy(dev\_flag, &flag, sizeof(bool), cudaMemcpyHostToDevice);

cudaBindTexture(NULL, matrix\_t, dev\_matrix, cudaCreateChannelDesc<int>(), sizeof(int) \* N \* N);

if(N < 250){

threads = N;

}

else{

threads = 250;

}

cudaEventRecord(start, 0);

matrixLU<<<1,threads>>>(dev\_lu\_matrix, N, dev\_flag);

cudaEventRecord(stop, 0);

cudaEventSynchronize(stop);

cudaEventElapsedTime(&elapsed\_time, start, stop);

cudaEventDestroy(start);

cudaEventDestroy(stop);

cudaMemcpy(&flag, dev\_flag, sizeof(bool), cudaMemcpyDeviceToHost);

if(flag){

cudaMemcpy(lu\_matrix, dev\_lu\_matrix, sizeof(float) \* N \* N, cudaMemcpyDeviceToHost);

}

cudaFree(dev\_matrix);

cudaFree(dev\_lu\_matrix);

if(!flag) return -1;

return round(elapsed\_time);

}