

Код рабочей штуки с возведением в квадрат на OpenCL  
  
#include <CL/cl.h>

#include <stdio.h>

#include <time.h>

#define ARRAY\_SIZE 10000

const char\* kernelSource =

"\_\_kernel void square(\_\_global float\* input, \_\_global float\* output) {\n"

" int id = get\_global\_id(0);\n"

" output[id] = input[id] \* input[id];\n"

"}\n";

void calculateOnCPU(float\* input, float\* output, int size) {

for (int i = 0; i < size; ++i) {

output[i] = input[i] \* input[i];

}

}

int main() {

cl\_platform\_id platform;

clGetPlatformIDs(1, &platform, NULL);

cl\_device\_id device;

clGetDeviceIDs(platform, CL\_DEVICE\_TYPE\_GPU, 1, &device, NULL);

cl\_context context = clCreateContext(NULL, 1, &device, NULL, NULL, NULL);

cl\_command\_queue queue = clCreateCommandQueueWithProperties(context, device, NULL, NULL);

// Создание массивов данных

float inputData[ARRAY\_SIZE];

float outputDataCPU[ARRAY\_SIZE];

float outputDataGPU[ARRAY\_SIZE];

for (int i = 0; i < ARRAY\_SIZE; ++i) {

inputData[i] = i;

}

// Измерение времени выполнения на CPU

clock\_t cpuStartTime = clock();

calculateOnCPU(inputData, outputDataCPU, ARRAY\_SIZE);

clock\_t cpuEndTime = clock();

printf("Time on CPU: %f seconds\n", ((double)(cpuEndTime - cpuStartTime)) / CLOCKS\_PER\_SEC);

cl\_mem inputBuffer = clCreateBuffer(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR,

sizeof(float) \* ARRAY\_SIZE, inputData, NULL);

cl\_mem outputBuffer = clCreateBuffer(context, CL\_MEM\_WRITE\_ONLY,

sizeof(float) \* ARRAY\_SIZE, NULL, NULL);

// Загрузка и сборка ядра

cl\_program program = clCreateProgramWithSource(context, 1, &kernelSource, NULL, NULL);

clBuildProgram(program, 1, &device, NULL, NULL, NULL);

cl\_kernel kernel = clCreateKernel(program, "square", NULL);

// Установка аргументов ядра

clSetKernelArg(kernel, 0, sizeof(cl\_mem), &inputBuffer);

clSetKernelArg(kernel, 1, sizeof(cl\_mem), &outputBuffer);

// Запуск вычислений на GPU

size\_t globalSize = ARRAY\_SIZE;

clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &globalSize, NULL, 0, NULL, NULL);

// Измерение времени выполнения на GPU

clock\_t gpuStartTime = clock();

clEnqueueReadBuffer(queue, outputBuffer, CL\_TRUE, 0,

sizeof(float) \* ARRAY\_SIZE, outputDataGPU, 0, NULL, NULL);

clock\_t gpuEndTime = clock();

printf("Time on GPU: %f seconds\n", ((double)(gpuEndTime - gpuStartTime)) / CLOCKS\_PER\_SEC);

// Вывод результатов (первые 10 элементов)

printf("Results (CPU): ");

for (int i = 0; i < 10; ++i) {

printf("%f ", outputDataCPU[i]);

}

printf("\n");

printf("Results (GPU): ");

for (int i = 0; i < 10; ++i) {

printf("%f ", outputDataGPU[i]);

}

printf("\n");

// Очистка ресурсов

clReleaseMemObject(inputBuffer);

clReleaseMemObject(outputBuffer);

clReleaseKernel(kernel);

clReleaseProgram(program);

clReleaseCommandQueue(queue);

clReleaseContext(context);

return 0;

}

Рабочий код сложения и перемножения матриц

#include <CL/cl.h>

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define MATRIX\_SIZE 200

const char\* kernelSource =

"\_\_kernel void matrixMultiply(\_\_global int\* A, \_\_global int\* B, \_\_global int\* C, int width) {\n"

" int row = get\_global\_id(0);\n"

" int col = get\_global\_id(1);\n"

" int sum = 0;\n"

" for (int k = 0; k < width; ++k) {\n"

" sum += A[row \* width + k] \* B[k \* width + col];\n"

" }\n"

" C[row \* width + col] = sum;\n"

"}\n";

void printMatrix(const char\* name, int\* matrix, int rows, int cols) {

printf("%s:\n", name);

for (int i = 0; i < rows; ++i) {

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

printf("%d ", matrix[i \* cols + j]);

}

printf("\n");

}

printf("\n");

}

void matrixAdd(int\* A, int\* B, int\* C, int size) {

#pragma omp parallel for

for (int i = 0; i < size; ++i) {

C[i] = A[i] + B[i];

}

}

int main() {

cl\_platform\_id platform;

clGetPlatformIDs(1, &platform, NULL);

cl\_device\_id device;

clGetDeviceIDs(platform, CL\_DEVICE\_TYPE\_GPU, 1, &device, NULL);

cl\_context context = clCreateContext(NULL, 1, &device, NULL, NULL, NULL);

cl\_command\_queue queue = clCreateCommandQueueWithProperties(context, device, NULL, NULL);

int matrixA[MATRIX\_SIZE \* MATRIX\_SIZE];

int matrixB[MATRIX\_SIZE \* MATRIX\_SIZE];

int matrixCAdd[MATRIX\_SIZE \* MATRIX\_SIZE];

int matrixCMultiply[MATRIX\_SIZE \* MATRIX\_SIZE];

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; ++i) {

matrixA[i] = rand() % 10;

matrixB[i] = rand() % 10;

}

printMatrix("Matrix A", matrixA, MATRIX\_SIZE, MATRIX\_SIZE);

printMatrix("Matrix B", matrixB, MATRIX\_SIZE, MATRIX\_SIZE);

int matrixSize = MATRIX\_SIZE;

cl\_mem bufferA = clCreateBuffer(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR,

sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, matrixA, NULL);

cl\_mem bufferB = clCreateBuffer(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR,

sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, matrixB, NULL);

cl\_mem bufferCAdd = clCreateBuffer(context, CL\_MEM\_WRITE\_ONLY,

sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, NULL, NULL);

cl\_mem bufferCMultiply = clCreateBuffer(context, CL\_MEM\_WRITE\_ONLY,

sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, NULL, NULL);

cl\_mem bufferMatrixSize = clCreateBuffer(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR,

sizeof(int), &matrixSize, NULL);

cl\_program program = clCreateProgramWithSource(context, 1, &kernelSource, NULL, NULL);

clBuildProgram(program, 1, &device, NULL, NULL, NULL);

cl\_kernel kernelMultiply = clCreateKernel(program, "matrixMultiply", NULL);

// Выполнение сложения матриц на CPU с использованием OpenMP

matrixAdd(matrixA, matrixB, matrixCAdd, MATRIX\_SIZE \* MATRIX\_SIZE);

printMatrix("Matrix C (Addition - CPU)", matrixCAdd, MATRIX\_SIZE, MATRIX\_SIZE);

// Выполнение умножения матриц на GPU

clSetKernelArg(kernelMultiply, 0, sizeof(cl\_mem), &bufferA);

clSetKernelArg(kernelMultiply, 1, sizeof(cl\_mem), &bufferB);

clSetKernelArg(kernelMultiply, 2, sizeof(cl\_mem), &bufferCMultiply);

clSetKernelArg(kernelMultiply, 3, sizeof(int), &matrixSize);

size\_t globalSizeMultiply[2] = { MATRIX\_SIZE, MATRIX\_SIZE };

clEnqueueNDRangeKernel(queue, kernelMultiply, 2, NULL, globalSizeMultiply, NULL, 0, NULL, NULL);

clEnqueueReadBuffer(queue, bufferCMultiply, CL\_TRUE, 0,

sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, matrixCMultiply, 0, NULL, NULL);

printMatrix("Matrix C (Multiplication - GPU)", matrixCMultiply, MATRIX\_SIZE, MATRIX\_SIZE);

clReleaseMemObject(bufferA);

clReleaseMemObject(bufferB);

clReleaseMemObject(bufferCAdd);

clReleaseMemObject(bufferCMultiply);

clReleaseMemObject(bufferMatrixSize);

clReleaseKernel(kernelMultiply);

clReleaseProgram(program);

clReleaseCommandQueue(queue);

clReleaseContext(context);

return 0;

}

Нерабочий, но хоть какой-то вариант разложения на гпу, цпу

#include <iostream>

#include <vector>

#include <CL/cl.h>

const char\* kernelSource =

"\_\_kernel void lu\_decomposition(\_\_global float\* A, \_\_global float\* L, \_\_global float\* U, int n) {\n"

" int gid = get\_global\_id(0);\n"

"\n"

" for (int k = 0; k < n - 1; ++k) {\n"

" barrier(CLK\_LOCAL\_MEM\_FENCE);\n"

"\n"

" if (gid >= k) {\n"

" L[gid \* n + k] = U[gid \* n + k] / U[k \* n + k];\n"

" }\n"

"\n"

" barrier(CLK\_LOCAL\_MEM\_FENCE);\n"

"\n"

" for (int i = k + 1; i < n; ++i) {\n"

" if (gid >= k && gid < n) {\n"

" U[i \* n + gid] -= L[i \* n + k] \* U[k \* n + gid];\n"

" }\n"

" }\n"

" }\n"

"}\n";

void backwardSubstitution(const std::vector<std::vector<double>>& U, const std::vector<double>& b, std::vector<double>& x) {

int n = U.size();

#pragma omp parallel for

for (int i = n - 1; i >= 0; --i) {

double sum = 0.0;

#pragma omp parallel for reduction(+:sum)

for (int j = i + 1; j < n; ++j) {

sum += U[i][j] \* x[j];

}

x[i] = (b[i] - sum) / U[i][i];

}

}

int main() {

const int n = 4;

// Создание и заполнение матрицы A и вектора b

std::vector<std::vector<double>> A(n, std::vector<double>(n, 0));

std::vector<std::vector<double>> L(n, std::vector<double>(n, 0));

std::vector<std::vector<double>> U(n, std::vector<double>(n, 0));

std::vector<double> b(n, 0);

// Заполните A и b данными, соответствующими вашей задаче

// Создание OpenCL контекста, очереди и буферов

cl\_platform\_id platform;

clGetPlatformIDs(1, &platform, NULL);

cl\_device\_id device;

clGetDeviceIDs(platform, CL\_DEVICE\_TYPE\_GPU, 1, &device, NULL);

cl\_context context = clCreateContext(NULL, 1, &device, NULL, NULL, NULL);

cl\_command\_queue queue = clCreateCommandQueueWithProperties(context, device, NULL, NULL);

cl\_mem buffer\_A = clCreateBuffer(context, CL\_MEM\_READ\_WRITE | CL\_MEM\_COPY\_HOST\_PTR,

sizeof(float) \* n \* n, A.data()[0].data(), NULL);

cl\_mem buffer\_L = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, sizeof(float) \* n \* n, NULL, NULL);

cl\_mem buffer\_U = clCreateBuffer(context, CL\_MEM\_READ\_WRITE | CL\_MEM\_COPY\_HOST\_PTR,

sizeof(float) \* n \* n, A.data()[0].data(), NULL);

cl\_mem buffer\_b = clCreateBuffer(context, CL\_MEM\_READ\_WRITE | CL\_MEM\_COPY\_HOST\_PTR,

sizeof(float) \* n, b.data(), NULL);

// Компиляция и выполнение ядра OpenCL

cl\_program program = clCreateProgramWithSource(context, 1, &kernelSource, NULL, NULL);

clBuildProgram(program, 1, &device, NULL, NULL, NULL);

cl\_kernel kernel = clCreateKernel(program, "lu\_decomposition", NULL);

clSetKernelArg(kernel, 0, sizeof(cl\_mem), &buffer\_A);

clSetKernelArg(kernel, 1, sizeof(cl\_mem), &buffer\_L);

clSetKernelArg(kernel, 2, sizeof(cl\_mem), &buffer\_U);

clSetKernelArg(kernel, 3, sizeof(int), &n);

size\_t globalSize = n;

clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &globalSize, NULL, 0, NULL, NULL);

clFinish(queue);

// Чтение результатов с GPU на CPU

clEnqueueReadBuffer(queue, buffer\_L, CL\_TRUE, 0, sizeof(float) \* n \* n, L.data()[0].data(), 0, NULL, NULL);

clEnqueueReadBuffer(queue, buffer\_U, CL\_TRUE, 0, sizeof(float) \* n \* n, U.data()[0].data(), 0, NULL, NULL);

// Выполнение обратного хода метода Гаусса на CPU

std::vector<double> x(n, 0);

backwardSubstitution(U, b, x);

// Вывод результатов

std::cout << "Solution after LU decomposition and backward substitution:" << std::endl;

for (int i = 0; i < n; ++i) {

std::cout << "x[" << i << "] = " << x[i] << std::endl;

}

// Очистка ресурсов OpenCL

clReleaseMemObject(buffer\_A);

clReleaseMemObject(buffer\_L);

clReleaseMemObject(buffer\_U);

clReleaseMemObject(buffer\_b);

clReleaseKernel(kernel);

clReleaseProgram(program);

clReleaseCommandQueue(queue);

clReleaseContext(context);

return 0;

}

Неработающее разложение с матрицами на векторах  
  
#include <iostream>

#include <vector>

#include <CL/cl.h>

using namespace std;

const char\* kernelSource = R"(

\_\_kernel void lu\_decomposition(\_\_global float\* A, \_\_global float\* L, \_\_global float\* U, int n) {

int row = get\_global\_id(0);

int col = get\_global\_id(1);

if (row < col) {

L[row \* n + col] = 0.0f;

} else {

L[row \* n + col] = A[row \* n + col];

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

L[row \* n + col] -= L[row \* n + k] \* U[k \* n + col];

}

}

barrier(CLK\_GLOBAL\_MEM\_FENCE);

if (row > col) {

U[row \* n + col] = 0.0f;

} else if (row == col) {

U[row \* n + col] = 1.0f;

} else {

U[row \* n + col] = A[row \* n + col];

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

U[row \* n + col] -= L[row \* n + k] \* U[k \* n + col];

}

}

barrier(CLK\_GLOBAL\_MEM\_FENCE);

}

)";

void checkError(cl\_int err, const char\* operation) {

if (err != CL\_SUCCESS) {

cerr << "Error during " << operation << ": " << err << endl;

exit(EXIT\_FAILURE);

}

}

void printMatrix(const char\* name, const vector<vector<float>>& matrix, int rows, int cols) {

cout << name << ":" << endl;

for (int i = 0; i < rows; ++i) {

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

cout << "\t" << matrix[i][j] << "\t";

}

cout << endl;

}

cout << endl;

}

void proisv(vector <vector <float>> A, vector <vector <float>> B,

vector <vector <float>>& R, int n)

{

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

for (int k = 0; k < n; k++)

R[i][j] += A[i][k] \* B[k][j];

}

int main() {

const int n = 4;

vector<vector<float>> A(n, vector<float>(n, 0.0f));

vector<vector<float>> L(n, vector<float>(n, 1.0f));

vector<vector<float>> U(n, vector<float>(n, 0.0f));

vector<vector<float>> R(n, vector<float>(n, 0.0f));

for (int i = 0; i < n; i++) {

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

A[i][j] = static\_cast<float>(rand() % 20 - 10);

}

}

U = A;

// Create OpenCL context and command queue

cl\_platform\_id platform;

clGetPlatformIDs(1, &platform, NULL);

cl\_device\_id device;

clGetDeviceIDs(platform, CL\_DEVICE\_TYPE\_GPU, 1, &device, NULL);

cl\_context context = clCreateContext(NULL, 1, &device, NULL, NULL, NULL);

// Define command queue properties

cl\_queue\_properties properties[] = { CL\_QUEUE\_PROPERTIES, CL\_QUEUE\_PROFILING\_ENABLE, 0 };

cl\_command\_queue queue = clCreateCommandQueueWithProperties(context, device, properties, NULL);

// Flatten the matrices to 1D vectors

vector<float> flatA, flatL, flatU;

for (const auto& row : A) {

flatA.insert(flatA.end(), row.begin(), row.end());

}

for (const auto& row : U) {

flatU.insert(flatU.end(), row.begin(), row.end());

}

for (const auto& row : L) {

flatL.insert(flatL.end(), row.begin(), row.end());

}

//flatL.resize(n \* n, 0.0f);

//flatU.resize(n \* n, 0.0f);

// Create OpenCL buffers

cl\_mem bufferA = clCreateBuffer(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR,

sizeof(float) \* n \* n, flatA.data(), NULL);

cl\_mem bufferL = clCreateBuffer(context, CL\_MEM\_READ\_WRITE,

sizeof(float) \* n \* n, flatL.data(), NULL);

cl\_mem bufferU = clCreateBuffer(context, CL\_MEM\_READ\_WRITE,

sizeof(float) \* n \* n, flatU.data(), NULL);

// Load and build the OpenCL program

cl\_program program = clCreateProgramWithSource(context, 1, &kernelSource, NULL, NULL);

clBuildProgram(program, 1, &device, NULL, NULL, NULL);

// Create OpenCL kernel

cl\_kernel kernel = clCreateKernel(program, "lu\_decomposition", NULL);

// Set OpenCL kernel arguments

clSetKernelArg(kernel, 0, sizeof(cl\_mem), &bufferA);

clSetKernelArg(kernel, 1, sizeof(cl\_mem), &bufferL);

clSetKernelArg(kernel, 2, sizeof(cl\_mem), &bufferU);

clSetKernelArg(kernel, 3, sizeof(int), &n);

// Enqueue the OpenCL kernel for execution

size\_t globalSize[2] = { static\_cast<size\_t>(n), static\_cast<size\_t>(n) };

cl\_event event;

clEnqueueNDRangeKernel(queue, kernel, 2, NULL, globalSize, NULL, 0, NULL, &event);

clWaitForEvents(1, &event);

// Read the result back to host

clEnqueueReadBuffer(queue, bufferL, CL\_TRUE, 0, sizeof(float) \* n \* n, flatL.data(), 0, NULL, NULL);

clEnqueueReadBuffer(queue, bufferU, CL\_TRUE, 0, sizeof(float) \* n \* n, flatU.data(), 0, NULL, NULL);

// Release OpenCL resources

clReleaseMemObject(bufferA);

clReleaseMemObject(bufferL);

clReleaseMemObject(bufferU);

clReleaseProgram(program);

clReleaseKernel(kernel);

clReleaseCommandQueue(queue);

clReleaseContext(context);

// Convert 1D vectors back to 2D matrices

for (int i = 0; i < n; ++i) {

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

L[i][j] = flatL[i \* n + j];

U[i][j] = flatU[i \* n + j];

}

}

// Print the matrices

printMatrix("A Matrix", A, n, n);

printMatrix("L Matrix", L, n, n);

printMatrix("U Matrix", U, n, n);

proisv(L, U, R, n);

printMatrix("L\*U Matrix", R, n, n);

return 0;

}

[1] Архитектура вычислительных систем [Электронный ресурс]: учебное пособие – Эл. изд. - Электрон. текстовые дан. (1 файл pdf: 77 с.). – Грейбо С.В., Новосёлова Т.Е., Пронькин Н.Н., Семёнычева И.Ф. 2019. – Режим доступа: http://scipro.ru/conf/computerarchitecture.pdf – Дата доступа 28.09.2023

[2] Архитектура вычислительной системы, разновидности [Электронный ресурс]. – Режим доступа: https://studfile.net/preview/16387895/ Дата доступа 28.09.2023

[3] Каптерев А.И. Электронный учебник по информатике [Электронный ресурс]. – Режим доступа: http://www.mediagnosis.ru/Autorun/Page6/5\_3\_.htm – Дата доступа 28.09.2023

[4] Скотт Мюллер Модернизация и ремонт ПК 17-е изд. М. Вильямс, 2007, 499 – 572, 653 – 700 c.

[5] Яндекс Глоссарий [Электронный ресурс]. – Режим доступа: https://cloud.yandex.ru/docs/glossary/gpu.

[6] Microsoft Documents [Электронный ресурс]. – Режим доступа https://obzorposudy.ru/polezno/arxitektura-x64-cto-eto-znacit-i-kak-ona-rabotaet – Дата доступа 28.09.2023

[7] История развития оперативной памяти [Электронный ресурс]. – Режим доступа: https://m-i-kuznetsov.livejournal.com/213674.html – Дата доступа 28.09.2023

[8] NVMe-накопитель Samsung [Электронный ресурс]. – Режим доступа: https://3dnews.ru/1021227/obzor-nvmenakopitelya-samsung-980-pro – Дата доступа 28.09.2023

[9] История развития графики Intel [Электронный ресурс]. – Режим доступа: https://club.dns-shop.ru/blog/t-100-protsessoryi/78375-istoriya-razvitiya-grafiki-intel-chast-2/ – Дата доступа 28.09.2023

[10] Microsoft Windows 10 – полный обзор [Электронный ресурс]. – Режим доступа: https://xn--80aa0aebnilejl.xn--p1ai/%D0%9F%D0%BE%D0% BB%D0%B5%D0%B7%D0%BD%D0%B0%D1%8F\_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F/Microsoft\_Windows\_10/ – Дата доступа 03.10.2023

[11] Что такое операционная система [Электронный ресурс]. – Режим доступа: https://help.reg.ru/support/servery-vps/oblachnyye-servery/ustanovka-programmnogo-obespecheniya/chto-takoye-operatsionnaya-sistema/ – Дата доступа 03.10.2023

[12] Интегрированная среда разработки [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/glossary/ide// – Дата доступа 05.10.2023

[13] Язык программирования [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/glossary/yazyk-programmirovaniya/ – Дата доступа 05.10.2023

[14] Microsoft Learn [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2022 – Дата доступа 05.10.2023

[15] Web proger C/C++ [Электронный ресурс]. – Режим доступа: http://web.spt42.ru/index.php/chto-takoe-c-plus-plus – Дата доступа 05.10.2023

[16] Преимущества распределенных вычислений [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/what-is/distributed-computing/ – Дата доступа 15.10.2023

[17] Многозадачность и многопоточность. Основные понятия [Электронный ресурс]. – Режим доступа: https://radioprog.ru/post/1402 – Дата доступа 15.10.2023

[18] OpenCL. Открытый язык для параллельных программ [Электронный ресурс]: учебное пособие – Эл. изд. - Электрон. текстовые дан. (1 файл pdf: 88 с.). – Антонюк В.А. 2017. – Режим доступа: https://cmp.phys.msu.ru/sites/default/files/OpenCL.pdf – Дата доступа 15.10.2023

[19] Introduction of OpenMP [Электронный ресурс]. – Режим доступа: https://carleton.ca/rcs/rcdc/introduction-to-openmp/ – Дата доступа 17.10.2023

[20] Методы и средства отладки Visual Studio [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/visualstudio/debugger/write-better-code-with-visual-studio?view=vs-2022 – Дата доступа 17.10.2023

[21] Влияние различных характеристик на быстродействие процессоров современных архитектур. [Электронный ресурс]. – Режим доступа: https://ixbt.com/cpu/archspeed-2009-3.shtml – Дата доступа 17.10.2023