Использование технологии **NVIDIA CUDA** в решении СЛАУ высокой размерности

Докладчик:

к.т.н., доцент Литвинов Владимир Николаевич

зав. кафедрой «Математика и Биоинформатика»,

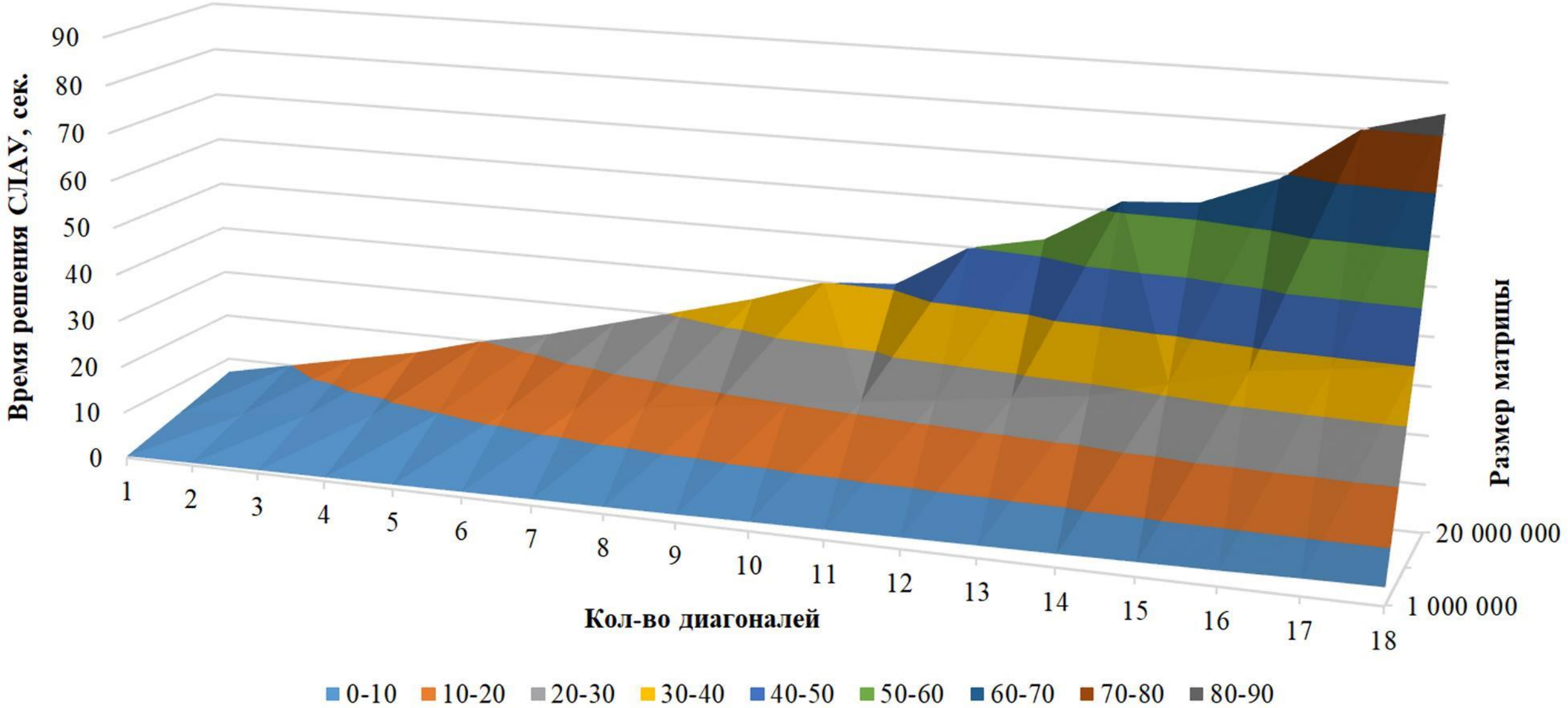
и.о. зам. директора по информатизации Азово-Черноморского инженерного института ФГБОУ ВО Донской ГАУ,

доцент кафедры ПИТСТТС ФГБОУ ВО «Донской государственный технический университет»

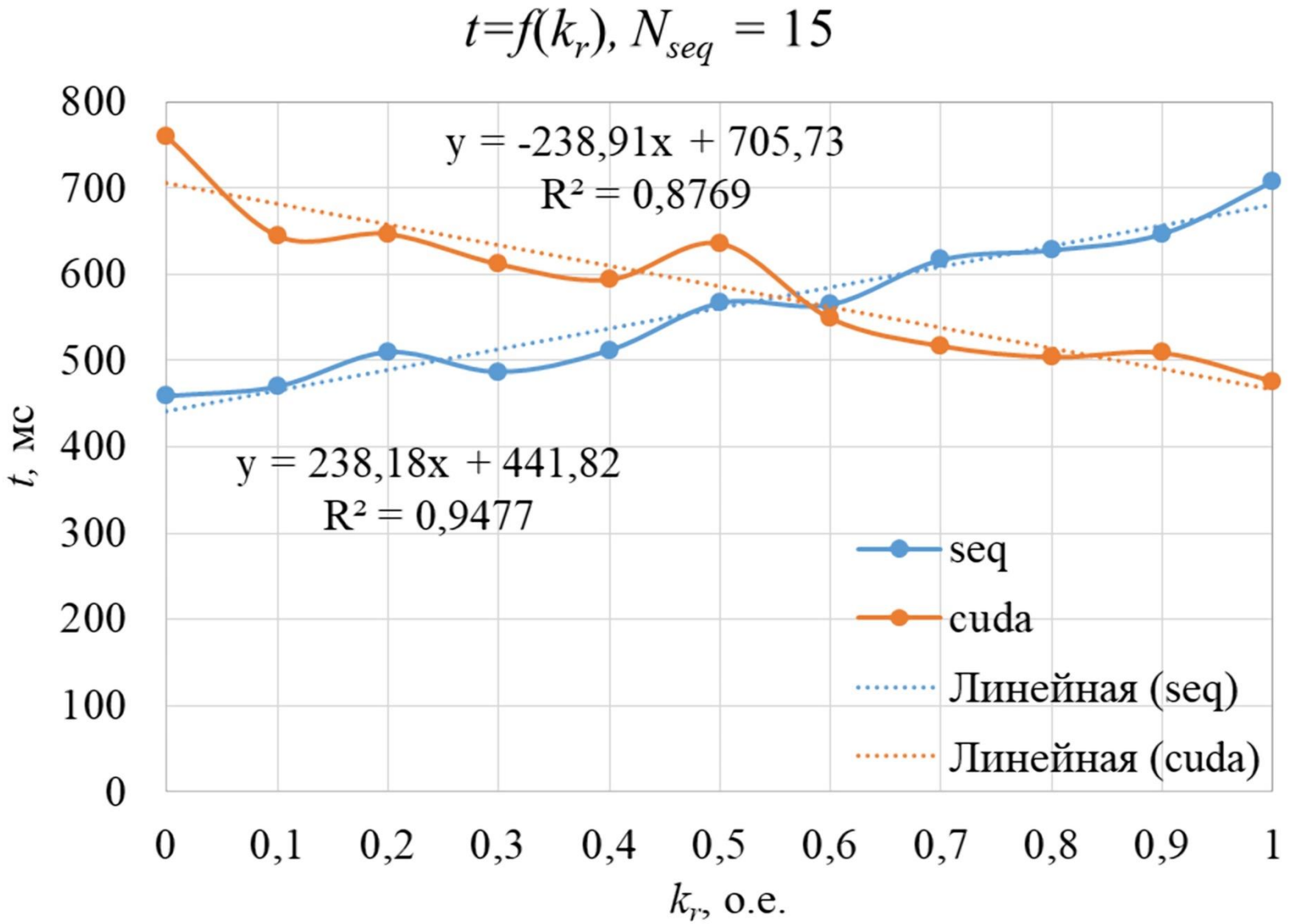
Методы решения СЛАУ

- **Прямые методы**
  - Метод Гаусса
  - Метод Холецкого
  - Метод прогонки и др.
- **Итерационные**
  - Метод простой итерации
  - Метод Зейделя - модификация метода Якоби
  - Метод релаксации
  - **Попеременно-треугольный итерационный метод**

Опыт решения СЛАУ прямыми методами



Проблема преобразования разрежённых матриц между различными форматами хранения



Модифицированный попеременно-треугольный итерационный метод (МПТМ) решения СЛАУ

В конечномерном гильбертовом пространстве H решается операторное уравнение

$$Ax = f, \quad A: H \to H \tag{1}$$

где A – линейный, положительно определенный оператор ($A > 0$).

Для нахождения решения задачи (1) используется неявный итерационный процесс

$$B\frac{x^{m+1} - x^m}{\tau_{m+1}} + Ax^m = f \qquad B: H \to H \tag{2}$$

m – номер итерации, $\tau > 0$ – итерационный параметр, B – некоторый обратимый оператор, который называется предобуславливателем или стабилизатором.

Оператор B построен, исходя из аддитивного представления оператора A_0 – симметричной части оператора A

$$A_0 = R_1 + R_2, \quad R_1 = R_2^*$$

где $A = A_0 + A_1, \quad A_0 = A_0^*, \; A_1 = -A_1^*$

Оператор-предобуславливатель

$$B = (D + \omega R_1) D^{-1} (D + \omega R_2), \; D = D^* > 0, \; \omega > 0$$

Алгоритм расчета сеточных уравнений модифицированным попеременно-треугольным методом вариационного типа

1) $$r^m = Ax^m - f$$ (вектор невязки)

2) $$B(\omega_m)w^m = r^m$$

3) $$\tilde{\omega}_m = \sqrt{\frac{\left(Dw^m, w^m\right)}{\left(D^{-1}R_2w^m, R_2w^m\right)}}$$ (вектор поправки)

4) $$s_m{}^2 = 1 - \frac{\left(A_0w^m, w^m\right)^2}{\left(B^{-1}A_0w^m, A_0w^m\right)\left(Bw^m, w^m\right)}, \quad k_m^2 = \frac{\left(B^{-1}A_1w^m, A_1w^m\right)}{\left(B^{-1}A_0w^m, A_0w^m\right)},$$

5) $$\theta_m = \frac{1 - \sqrt{\frac{s_m{}^2 k_m^2}{\left(1 + k_m^2\right)}}}{1 + k_m^2\left(1 - s_m{}^2\right)}$$

6) $$\tau_{m+1} = \theta_m \frac{\left(A_0 w^m, w^m\right)}{\left(B^{-1} A_0 w^m, A_0 w^m\right)}$$

7) $$x^{m+1} = x^m - \tau_{m+1} w^m$$

8) $$\omega_{m+1} = \tilde{\omega}_m$$

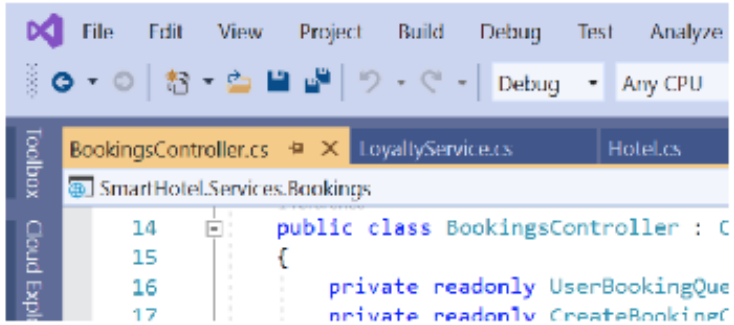
Основные этапы вычислений

- Вычисление вектора невязки – 1,73 %
- **Проход от (1,1,1) до (N_x-1, N_y-1, N_z-1) – 47,61 %**
- **Проход от (N_x-1, N_y-1, N_z-1) до (1,1,1) – 47,61 %**
- Расчет скалярных произведений – 2,69 %
- Пересчет вектора скорости – 0,36 %

Программное обеспечение

Visual Studio Community

Полнофункциональная интегрированная среда разработки для написания, отладки, тестирования и развертывания кода на любой платформе. **Подробнее**



Скачать бесплатно

developer.nvidia.com/cuda-downloads

NVIDIA DEVELOPER

ACCOUNT

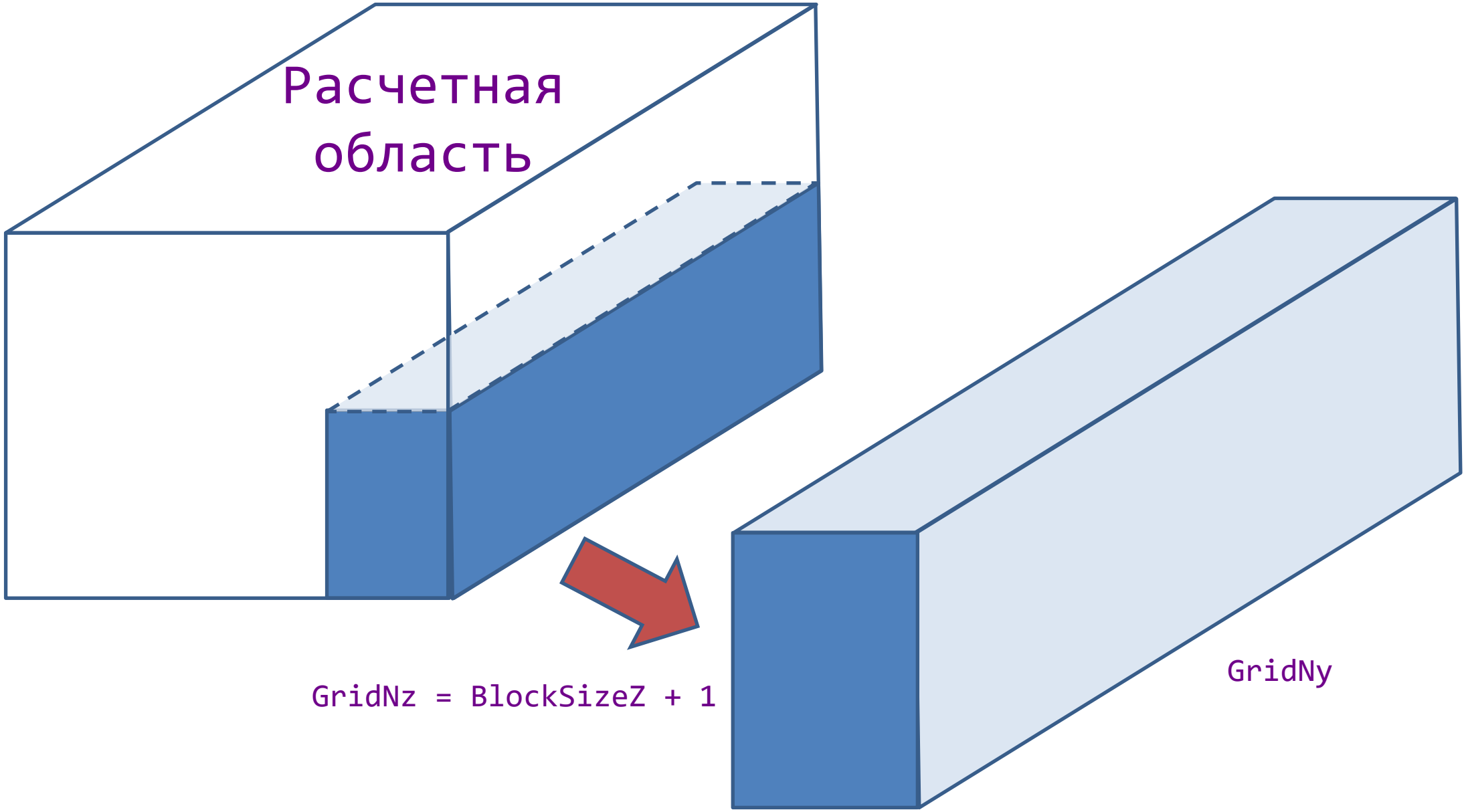
CUDA Toolkit 11.2 Update 1 Downloads

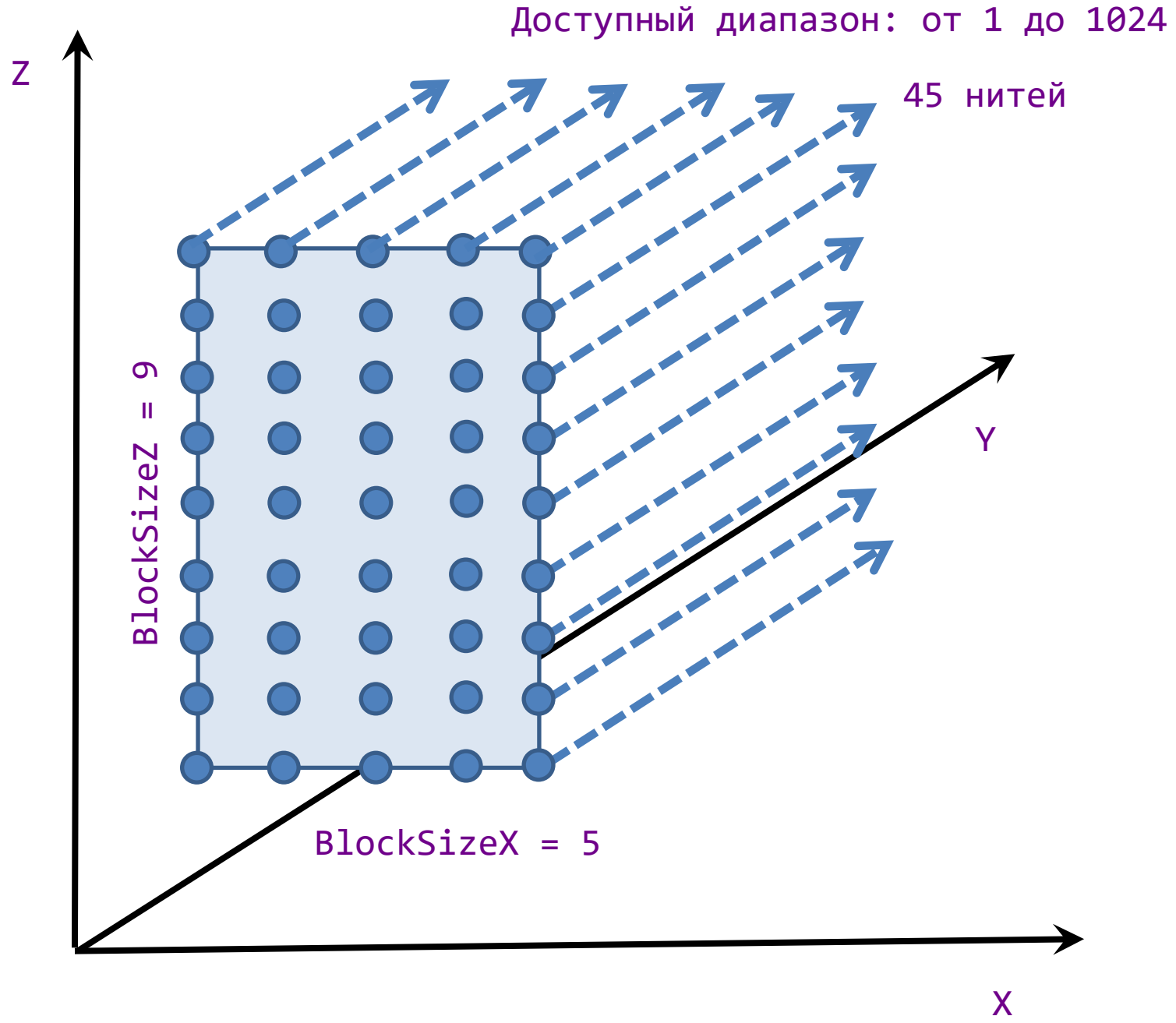
Select Target Platform

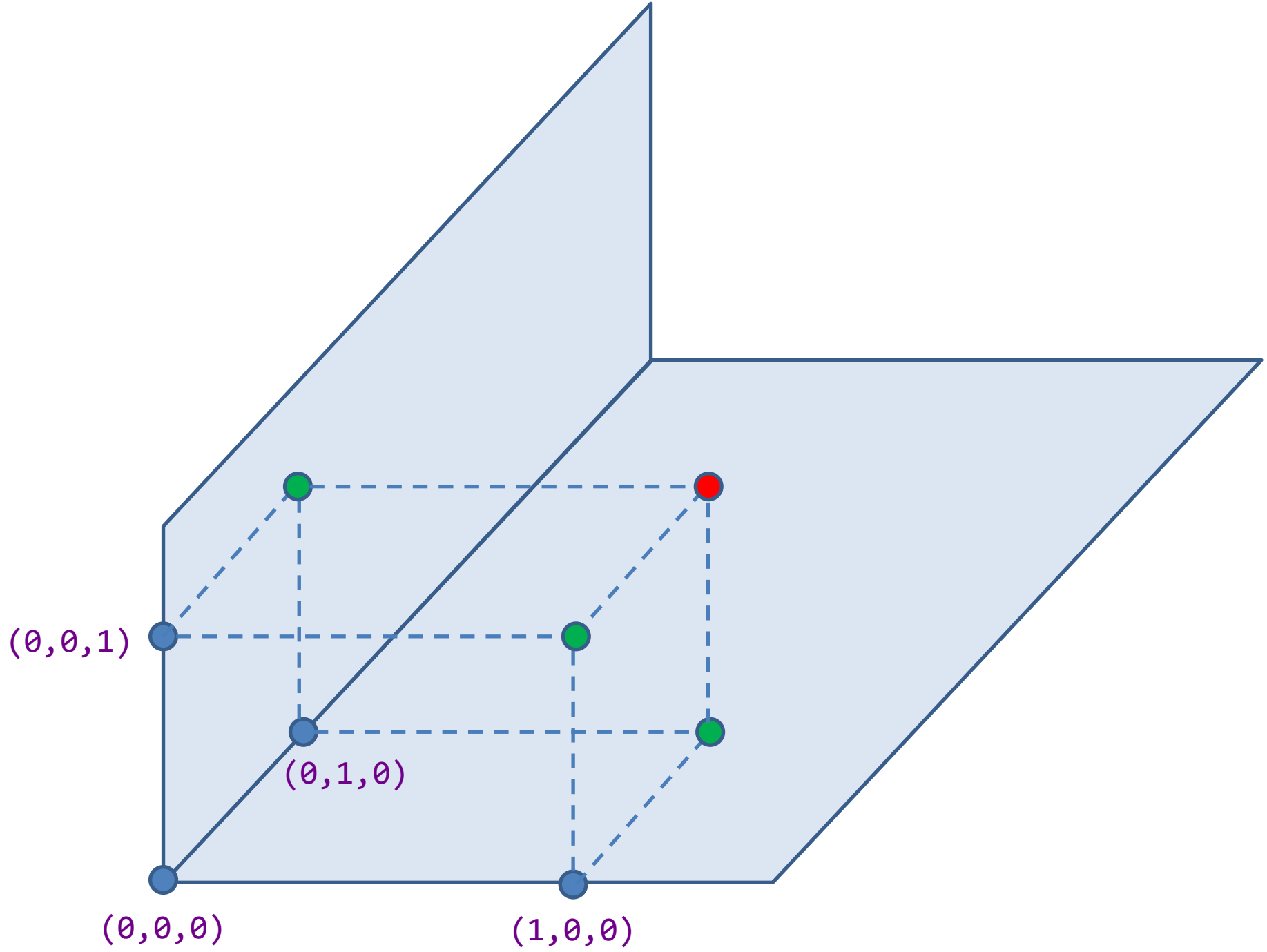
Click on the green buttons that describe your target platform. Only supported platforms will be shown. By downloading and using the software, you agree to fully comply with the terms and conditions of the CUDA EULA.

Operating System

Linux Windows


Расчетная область
GridNz = BlockSizeZ + 1
GridNy
GridNx = BlockSizeX + 1


Доступный диапазон: от 1 до 1024
45 нитей
Z
BlockSizeZ = 9
Y
BlockSizeX = 5
X


(0,0,1)
(0,1,0)
(0,0,0)
(1,0,0)

```
// 1. Вычисляем размер массива данных
size_t size = GridN; // Кол-во элементов
size_t sizeInBytesInt = size * sizeof(int);   // Размер массива int в байтах
size_t sizeInBytesDouble = size * sizeof(double);// Размер массива double в байтах

// 2. Выделяем память под массив в ОЗУ
double* host_c0 = (double*)malloc(sizeInBytesDouble);
...

// 3. Выделяем память под массивы на видеокарте
double* dev_c0 = NULL;
cudaMalloc((void**)&dev_c0, sizeInBytesDouble);
...

// 4. Копируем массивы из ОЗУ в GPU
cudaMemcpy(dev_c0, host_c0, sizeInBytesDouble, cudaMemcpyHostToDevice);

// 5. Запуск «ядра» CUDA
ptmKernel3 << < 1, dim3(BlockSizeX, 1, BlockSizeZ) >> > (dev_r, dev_c0, dev_c2, dev_c4,
dev_c6, GridN, omega);

// Копируем массив с результатами вычислений из памяти GPU в ОЗУ
cudaMemcpy(host_r, dev_r, sizeInBytesDouble, cudaMemcpyDeviceToHost);
```

```
__global__ void ptmKernel3(double* r, double* c0, double* c2,
double* c4, double* c6, double omega)
{
    __shared__ double cache[BlockSizeX][BlockSizeZ];

    const size_t threadX = blockDim.x * blockIdx.x + threadIdx.x;
    const size_t threadZ = blockDim.z * blockIdx.z + threadIdx.z;

    // Индекс строки, которую обрабатывает текущий поток
    const size_t idx_x = threadX + 1;
    // Индекс слоя, который обрабатывает текущий поток
    const size_t idx_z = threadZ + 1;

    size_t currentY = 1; // 0 - граница, берём 1
```

```
for (size_t s = 3; s <= GridNx + GridNy + GridNz - 3; s++)
    {
        __syncthreads();
        if (idx_x + currentY + idx_z == s && s < GridNy + idx_x + idx_z)
        {
            size_t nodeIndex = idx_x + (BlockSizeX + 1) * currentY + GridXY * idx_z;

            size_t m0 = nodeIndex;

            double c0m0 = c0[m0];
            if (c0m0 > 0)
            {
                size_t m2 = m0 - 1;
                size_t m4 = m0 - GridNx;
                size_t m6 = m0 - GridXY;
```

```
                double rm4 = 0;
                if (s > 3 + threadX + threadZ)
                {
                    rm4 = cache[threadX][threadZ];
                }
                else
                {
                    rm4 = r[m4];
                }
                // Аналогично rm2 и rm6
                double rm0 = (omega * (c2[m0] * rm2 + c4[m0] * rm4 +
                                       c6[m0] * rm6) +
                                       r[m0]) / ((0.5 * omega + 1) * c0m0);

                cache[threadX][threadZ] = rm0;
                r[m0] = rm0;
            }

            currentY++;
        }
    }
}
```

NVIDIA GeForce GT 710

Спецификации видеопамяти

Объем видеопамяти	2 ГБ
Тип памяти	GDDR3
Эффективная частота памяти	1600 МГц
Разрядность шины памяти	64 бит
Максимальная пропускная способность памяти	12.8 Гбайт/сек

Подключение

Интерфейс подключения	PCI-E
Версия PCI Express	2.0
Поддержка мультипроцессорной конфигурации	не поддерживается

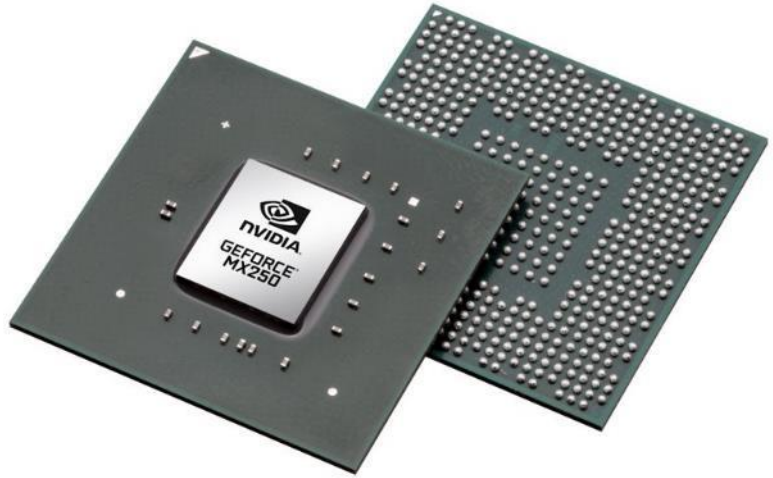
Спецификации видеопроцессора

Микроархитектура	Kepler
Техпроцесс	28 нм
Штатная частота работы видеочипа	954 МГц
Турбочастота	нет
Шейдерные ALU	192

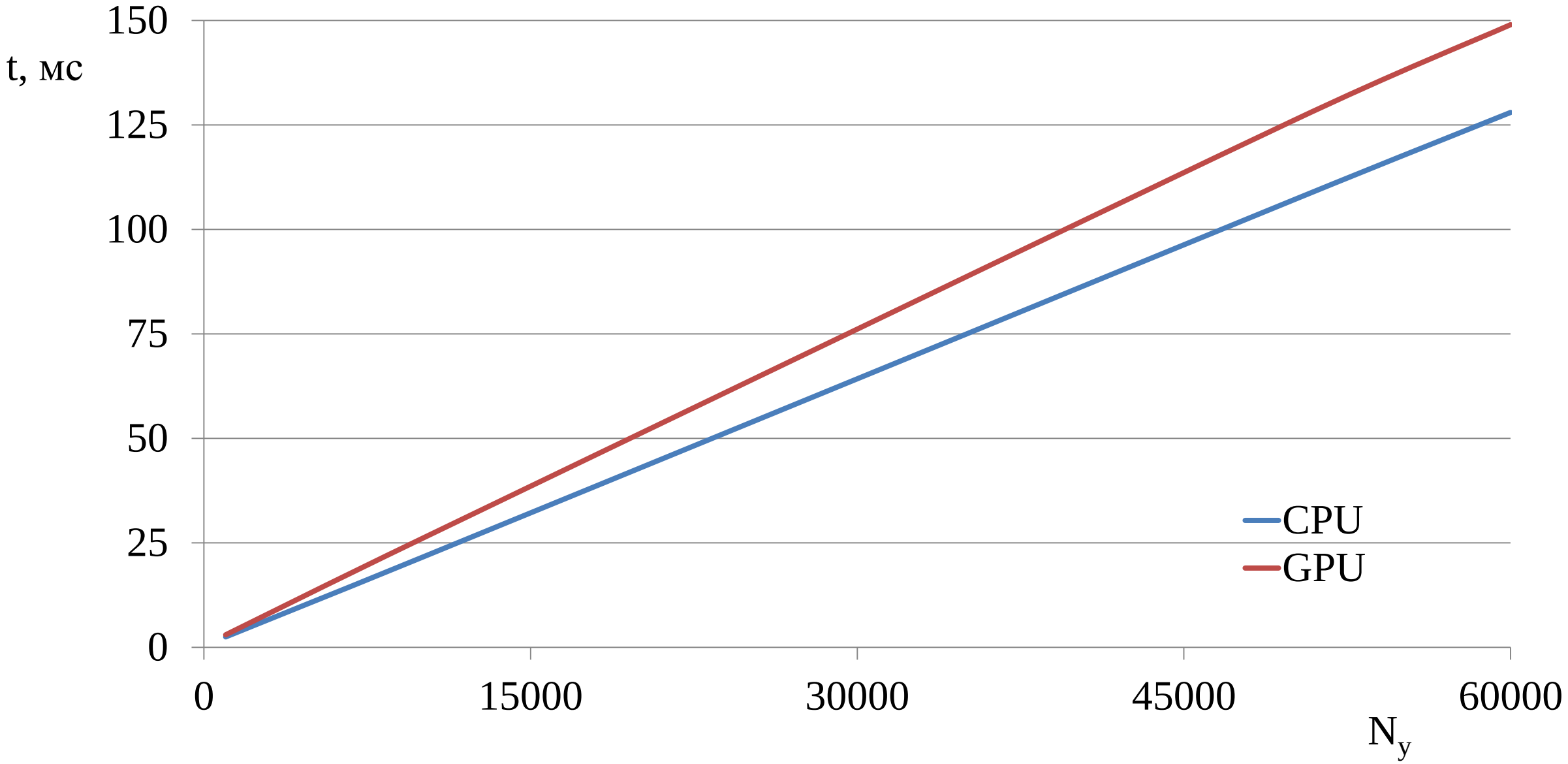
NVIDIA GEFORCE MX250

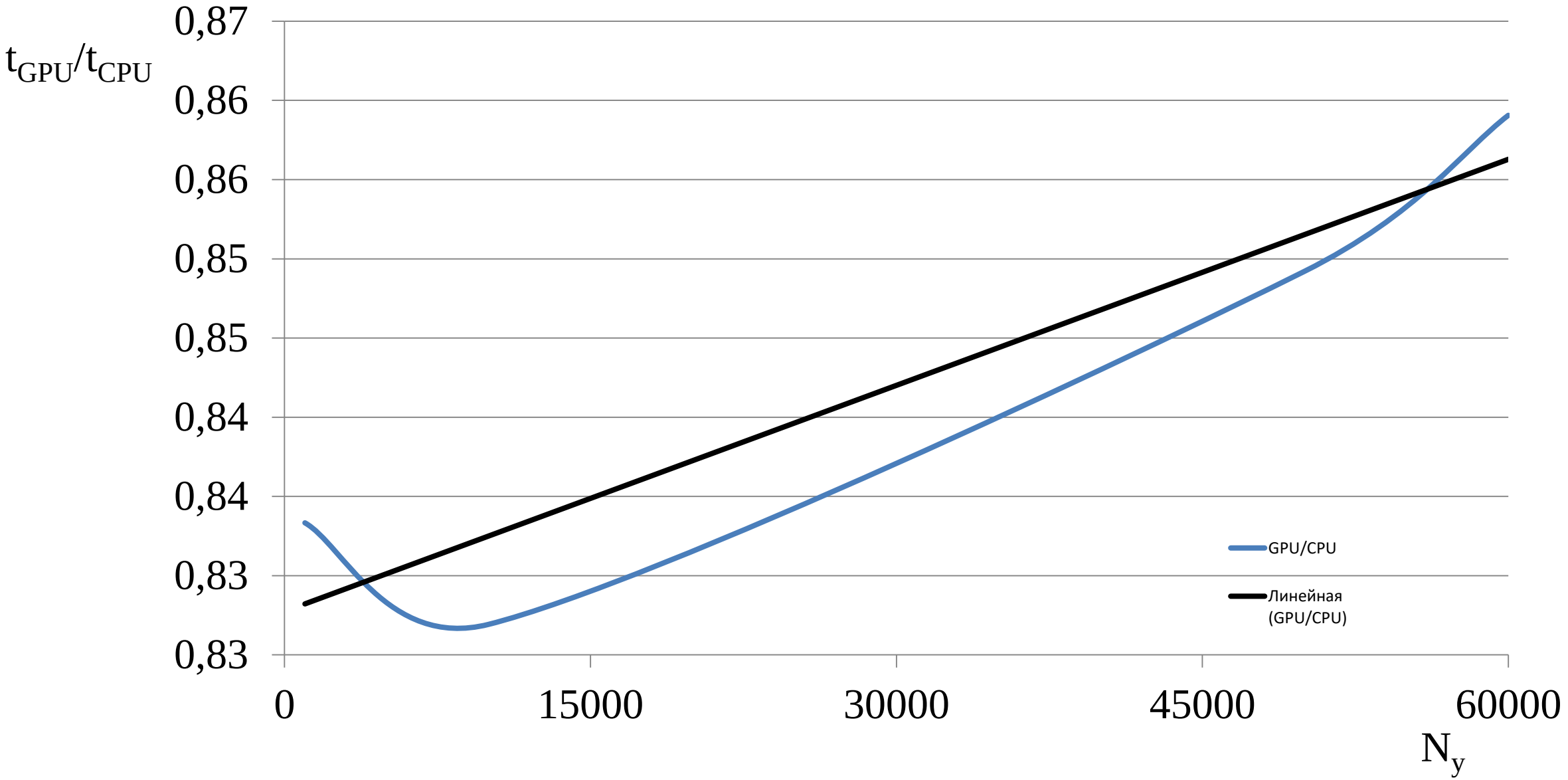
ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

Тип видеокарты	Мобильная
Кодовое имя чипа	GP108
Архитектура	Pascal
Количество ядер CUDA	384
Тактовая частота, МГц	1519 + boost
Частота памяти, МГц	7000
Тип памяти	GDDR5
Разрядность шины памяти, бит	64
Объем видеопамяти, Мб	2048 (4096)



Зависимость времени исполнения алгоритма пересчета вектора невязки (1 проход) от числа узлов сетки по оси y (N_x = 100, N_z=4)




t_{GPU}/t_{CPU}
0,87
0,86
0,86
0,85
0,85
0,84
0,84
0,83
0,83
0
15000
30000
45000
60000
N_y
GPU/CPU
Линейная (GPU/CPU)

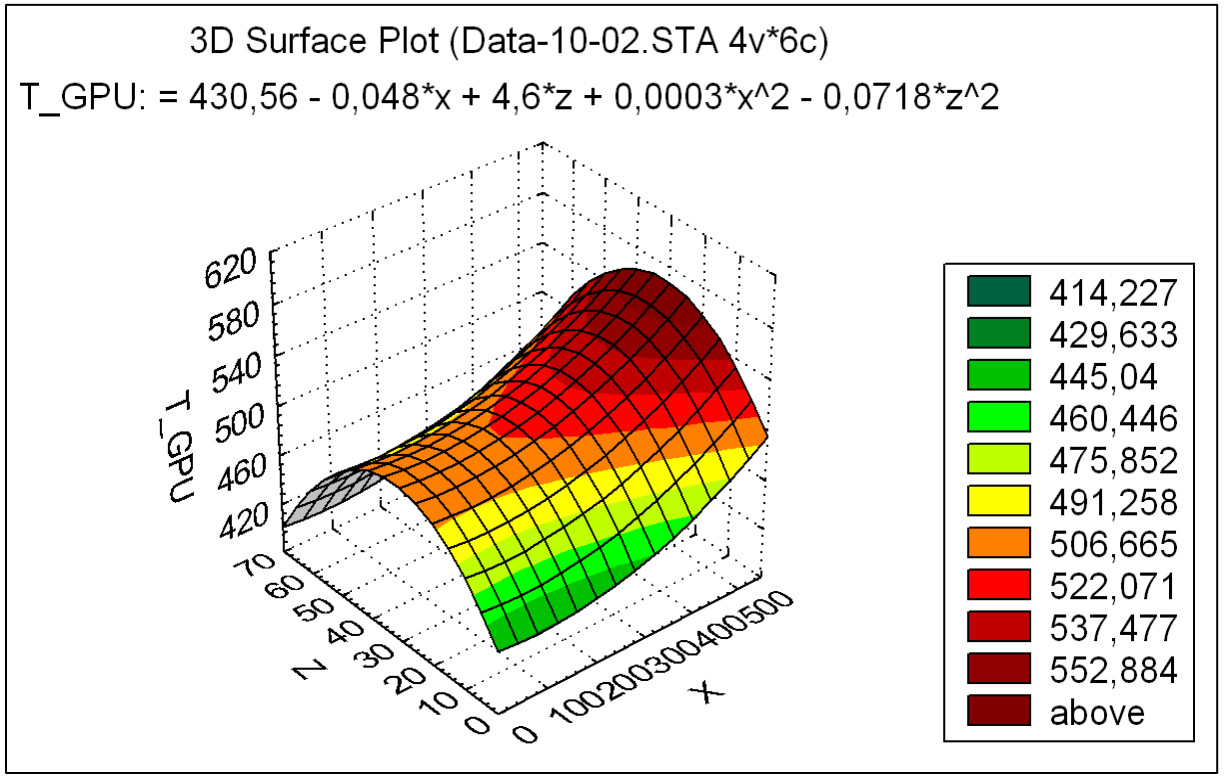
Цель – определить, как распределить потоки по осям X и Z расчетной сетки при фиксированном значении узлов сетки по оси Y=10000, чтобы время вычисления на GPU одного шага ПТМ на GPU было минимальным.

В качестве факторов приняты две величины: X – количество потоков по оси X, Z – количество потоков по оси Z.

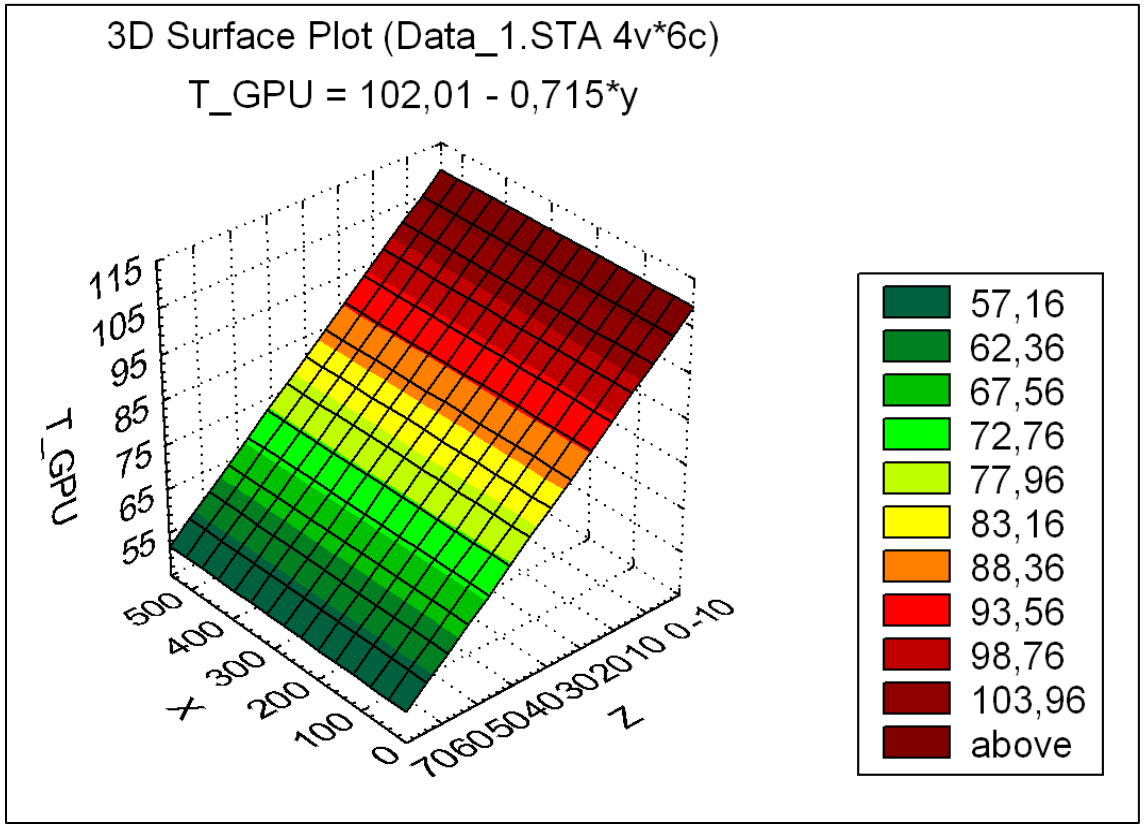
Целевая функция: T_GPU – время вычисления одного шага ПТМ на GPU, мс.

Произведение потоков X·Z не должно превышать 1024. Такое ограничение накладывает CUDA, т.к. 1024 – это максимальное количество потоков в одном блоке.

X	Z	t_{GPU}, мс	
		GT 710	MX 250
2	512	X	X
4	256	X	X
8	128	X	X
16	64	**430**	**64**
32	32	**503**	**65**
64	16	**484**	**81**
128	8	**462**	**109**
256	4	**457**	**100**
512	2	**501**	**103**


3D Surface Plot (Data-10-02.STA 4v*6c)
T_GPU: = 430,56 - 0,048*x + 4,6*z + 0,0003*x^2 - 0,0718*z^2
T_GPU
620
580
540
500
460
420
Z
70
60
50
40
30
20
10
0
X
0
100
200
300
400
500
414,227
429,633
445,04
460,446
475,852
491,258
506,665
522,071
537,477
552,884
above

GT 710


3D Surface Plot (Data_1.STA 4v*6c)
T_GPU = 102,01 - 0,715*y
T_GPU
115
105
95
85
75
65
55
X
500
400
300
200
100
0
Z
70
60
50
40
30
20
10
0
-10
57,16
62,36
67,56
72,76
77,96
83,16
88,36
93,56
98,76
103,96
above

MX250