Данный проект рассматривает различные применения вычислений на процессоре и графическом ядре, что позволяет в некоторых задачах получить весомый прирост вычислительной мощности и уменьшить время на выполнение задачи.

**Шаблон работы с глобальной памятью (сложение двух массивов)**

N = 512 \* 50 000

512 нитей в блоке, тогда 50 000 блоков

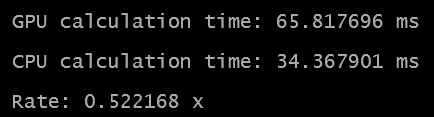


Рисунок 1 – Время выполнения задачи 1.1

Время расчета GPU-варианта включает в себя:

копирование данных с «host» на «device»;

выполнение «функции-ядра»;

копирование данных с «device» на «host».

Оценка времени выполнения только функции-ядра

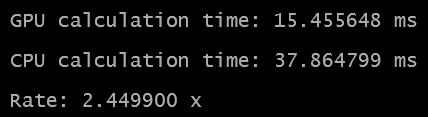


Рисунок 2 – Время выполнения только функции-ядра, задача 1.2

**Шаблон работы с глобальной памятью (Вычисления сложной функции)**

Общее время выполнения:

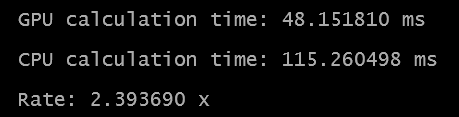


Рисунок 3 – Время выполнения задачи 2.1

Оценка времени выполнения только функции-ядра

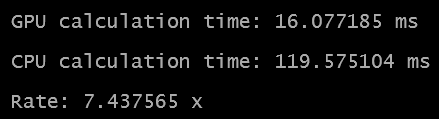


Рисунок 4 – Время выполнения только функции-ядра, задача 2.2

**Использование pinned-памяти**

cudaHostAlloc / cudaMallocHost – резервирование

cudaFreeHost – освобождение

Возможные шаблоны работы с pinned-памятью

malloc ( a )

cudaMallocHost ( b )

cudaMemcpy ( b, a ) cudaHostRegister ( a )

Обработка «a»

cudaMemcpy ( a, b ) cudaHostUnregister ( a )

cudaHostFree ( b )

вместо

hA = (float\*) malloc (mem\_size);

hB = (float\*) malloc (mem\_size);

hC = (float\*) malloc (mem\_size);

и

free ( hA );

free ( hB );

free ( hC );

поставим

cudaHostAlloc ((void\*\*) &hA, mem\_size, cudaHostAllocDefault);

cudaHostAlloc ((void\*\*) &hB, mem\_size, cudaHostAllocDefault);

cudaHostAlloc ((void\*\*) &hC, mem\_size, cudaHostAllocDefault);

и

cudaFreeHost ( hA );

cudaFreeHost ( hB );

cudaFreeHost ( hC );

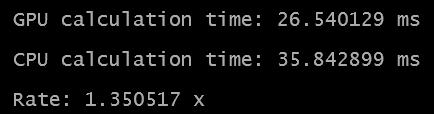


Рисунок 5 – Время выполнения всего расчета с pinned памятью, задача 1.3

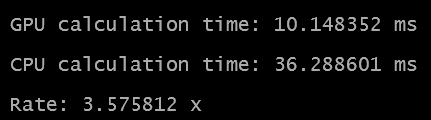


Рисунок 6 – Время выполнения функции-ядра с pinned памятью, задача 1.3

**Cuda-потоки**

Асинхронное копирование данных cudaMemcpyAsync

Использование pinned-памяти

Задание CUDA-потоков

Асинхронное копирование c «host» на «device»

Асинхронный запуск «функции-ядра»

Асинхронное копирование с «device» на «host»

Обработка двух массивов

N = 512 \* 50 000

512 нитей в блоке, тогда 50 000 блоков

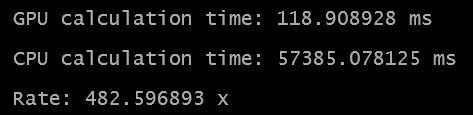


Рисунок 7 – Время выполнения с 1м потоком

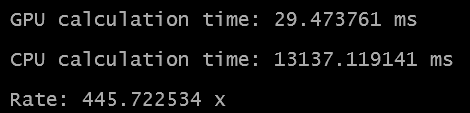


Рисунок 8 – Время выполнения с 4мя потоками

**Решения системы линейных алгебраических уравнений**

Разделяемой памяти и L1-кэша

48КБ SMEM, 16КБ L1 режим: cudaFuncCachePreferShared

16КБ SMEM, 48КБ L1 режим: cudaFuncCachePreferL1

Режим без предпочтения: cudaFuncCachePreferNone

В этом случае будет выбрана конфигурация в соответствии с текущим

контекстом.

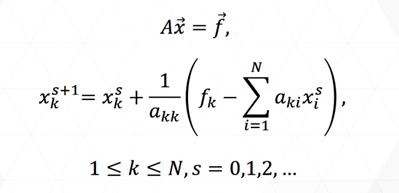
По умолчанию используется конфигурация с большей

разделяемой памятью: cudaFuncCachePreferShared

Переключение функцией: cudaFuncSetCacheConfig

Заполнения массива одинаковыми значениями

cudaError\_t cudaMemset ( void \*devPtr, int value, size\_t count );



Для обыкновенной матрицы:

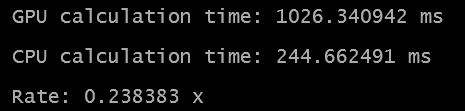


Рисунок 9 – Время решения СЛАУ обычной матрицы с выделением памяти

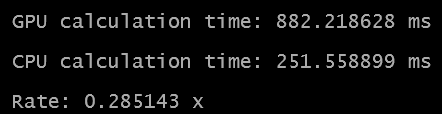


Рисунок 10 – Время решения СЛАУ обычной матрицы без выделения памяти

Для матрицы 𝑨:

dA [ j + t \* N ]

dA [ j ], dA [ j + N ], dA [ j + 2 \* N ],...

Для транспонированной матрицы 𝑨𝑻:

dA [ t + j \* N ]

dA [ j \* N ], dA [ 1 + j \* N ], dA [ 2 + j \* N ]

Для транспонированной матрицы:

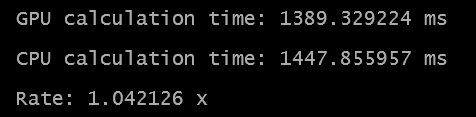


Рисунок 11 – Время решения СЛАУ транспонированной матрицы с выделением памяти

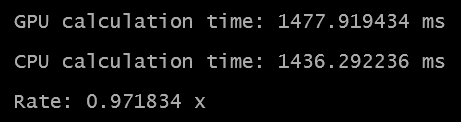
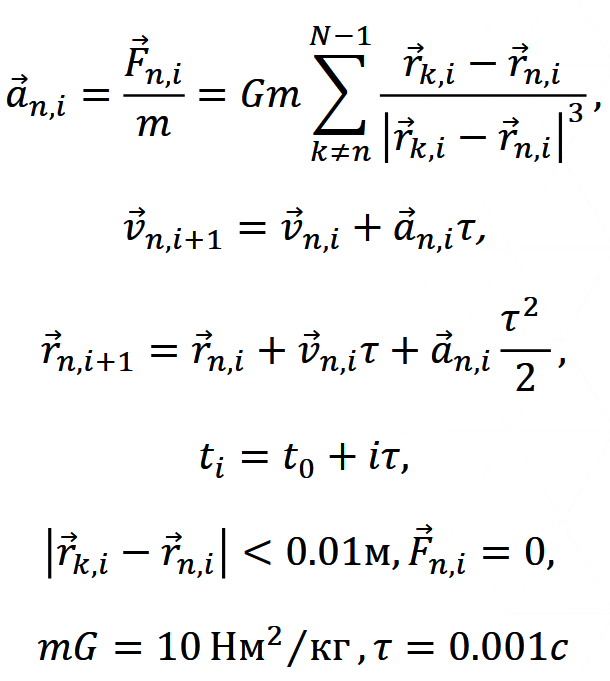


Рисунок 12 – Время решения СЛАУ транспонированной матрицы без выделения памяти

Разделяемая память (shared memory) относиться к быстрому типу памяти. Разделяемую память рекомендуется использовать для минимизации обращение к глобальной памяти, а так же для хранения локальных переменных функций. Адресация разделяемой памяти между нитями потока одинакова в пределах одного блока, что может быть использовано для обмена данными между потоками в пределах одного блока. Для размещения данных в разделяемой памяти используется спецификатор \_\_shared\_\_

**Задача N-тел**



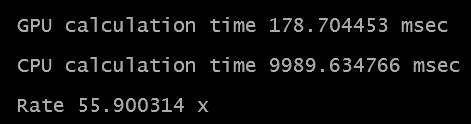


Рисунок 13 – Время решения задачи N-тел с 10240 частиц

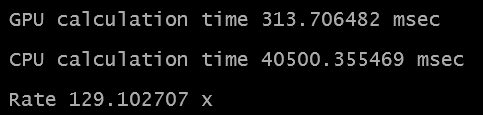


Рисунок 14 – Время решения задачи N-тел с 20480 частиц

Ниже представлены использованные функции-ядра для расчета и решения задачи N-тел:

– **для GPU**

// GPU-вариант. Расчет ускорения

\_\_global\_\_ void Acceleration\_GPU (float \*X, float \*Y,

float \*AX, float \*AY, int nt, int N)

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

float ax = 0.f; float ay = 0.f; float xx, yy, rr; int sh = (nt - 1) \* N;

for ( int j = 0; j < N; j++ ) // цикл по частицам

{if (j != id) // проверка самодействия

{xx = X[j + sh] - X[id + sh]; yy = Y[j + sh] - Y[id + sh];

rr = sqrtf (xx \* xx + yy \* yy);

if (rr > 0.01f) // минимальное расстояние 0.01 м

{rr = 10.f / (rr \* rr \* rr); ax += xx \* rr; ay += yy \* rr;

} // if rr

} // if id

} // for j

AX[id] = ax; AY[id] = ay;

}

**– для CPU**

// CPU – вариант. Вычисление ускорения

void Acceleration\_CPU (float \*X, float \*Y, float \*AX, float \*AY,

int nt, int N, int id)

{float ax = 0.f; float ay = 0.f; float xx, yy, rr; int sh = (nt - 1) \* N;

for ( int j = 0; j < N; j++ ) // цикл по частицам

{if ( j != id ) // поверка самодействия

{xx = X[j + sh] - X[id + sh]; yy = Y[j + sh] - Y[id + sh];

rr = sqrtf (xx \* xx + yy \* yy);

if ( rr > 0.01f ) // минимальное расстояние 0.01 м

{rr = 10.f / (rr \* rr \* rr); ax += xx \* rr; ay += yy \* rr;

} // if rr

} // if id

} // for

AX[id] = ax; AY[id] = ay;

}

Далее протестируем как изменятся результаты времени вычисления расчетов при использовании разделяемой памяти:

\_\_global\_\_ void Acceleration\_Shared (float \*X, float \*Y, float \*AX, float \*AY,

int nt, int N, int N\_block)

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

float ax = 0.f; float ay = 0.f; float xx, yy, rr; int sh = (nt – 1) \* N;

float xxx = X[id + sh]; float yyy = Y[id + sh];

\_\_shared\_\_ float Xs[256]; \_\_shared\_\_ float Ys[256]; // выделение разделяемой памяти

for ( int i = 0; i < N\_block; i++ ) // основной цикл по блокам

{Xs[threadIdx.x] = X[threadIdx.x + i \* blockDim.x + sh]; // копирование из глобальной

Ys[threadIdx.x] = Y[threadIdx.x + i \* blockDim.x + sh]; // в разделяемую память

\_\_syncthreads (); // синхронизация

for ( int j = 0; j < blockDim.x; j++ ) // вычислительная часть

{if ( ( j + i \* blockDim.x ) != id )

{xx = Xs[j] - xxx; yy = Ys[j] - yyy; rr = sqrtf ( xx \* xx + yy \* yy );

if ( rr > 0.01f ) {rr = 10.f /(rr \* rr \* rr); ax += xx \* rr; ay += yy \* rr;} //if

} // if id

} // for j

\_\_syncthreads (); // синхронизация

} // for i

AX[id] = ax; AY[id] = ay;

}

При использовании **ускорения shared памяти, время** **изменяется**:

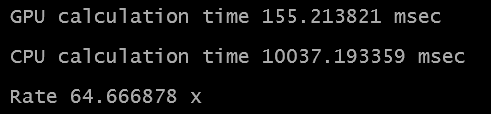


Рисунок 13 – Время решения задачи N-тел acceleration\_shared с 10240 частиц

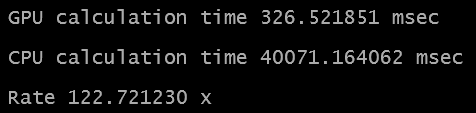
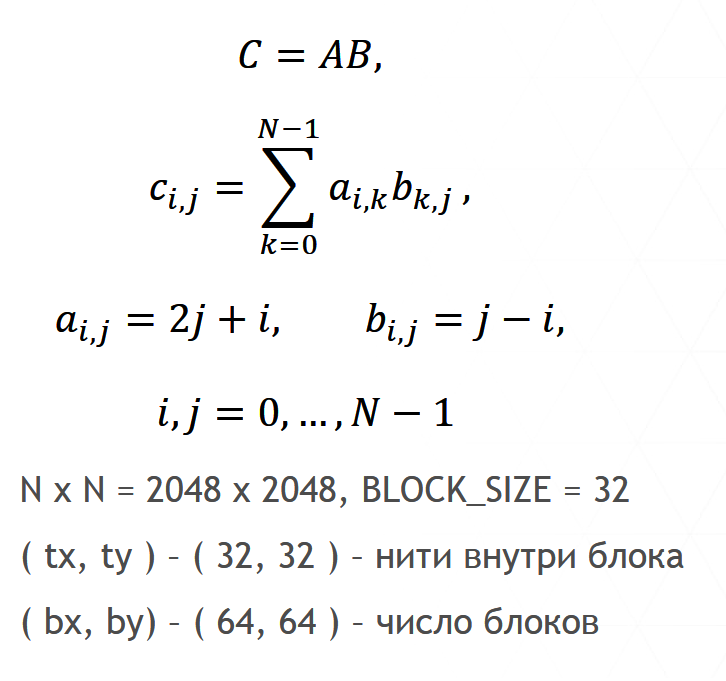


Рисунок 14 – Время решения задачи N-тел acceleration\_shared с 20480 частиц

**Перемножение матриц**



Ниже представлено 2 варианта результатов подсчета скорости выполнения кода с глобальным вариантом.

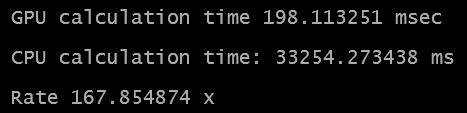


Рисунок 15 – Время решения задачи перемножения матриц примером GLOBAL с данными float

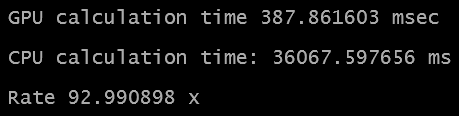


Рисунок 16 – Время решения задачи перемножения матриц примером GLOBAL с данными double

Недостатком этого метода является считывание одной строки N-раз. Проверим работу варианта SMEM используя следующую функцию-ядро:

\_\_global\_\_ void kernel\_smem\_1 ( float \*a, float \*b, int n, float \*c )

{int bx = blockIdx.x, by = blockIdx.y;

int tx = threadIdx.x, ty = threadIdx.y;

int aBegin = n \* BLOCK\_SIZE \* by, aEnd = aBegin + n - 1;

int bBegin = BLOCK\_SIZE \* bx, aStep = BLOCK\_SIZE, bStep = BLOCK\_SIZE \* n;

float sum = 0.0f;

\_\_shared\_\_ float as [BLOCK\_SIZE][BLOCK\_SIZE];

\_\_shared\_\_ float bs [BLOCK\_SIZE][BLOCK\_SIZE];

for ( int ia = aBegin, ib = bBegin; ia <= aEnd; ia += aStep, ib += bStep )

{as [tx][ty] = a [ia + n \* ty + tx]; bs [tx][ty] = b [ib + n \* ty + tx];

\_\_syncthreads ();

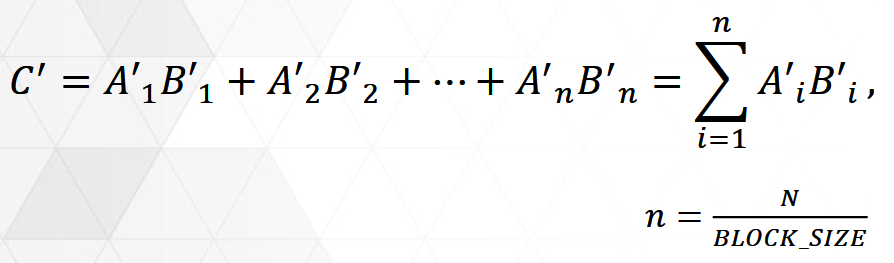
for ( int k = 0; k < BLOCK\_SIZE; k++ ) sum += as [k][ty] \* bs [tx][k];

\_\_syncthreads ();

}

c [aBegin + bBegin + ty \* n + tx] = sum;

}



Ниже представлен результат выполнения метода SMEM-2, в котором изменились следующие элементы:

В «функции-ядре» строки (SMEM-1):

\_\_shared\_\_ float as [BLOCK\_SIZE][BLOCK\_SIZE];

\_\_shared\_\_ float bs [BLOCK\_SIZE][BLOCK\_SIZE];

Заменили на строки (SMEM-2):

\_\_shared\_\_ float as [BLOCK\_SIZE][BLOCK\_SIZE + 1];

\_\_shared\_\_ float bs [BLOCK\_SIZE][BLOCK\_SIZE + 1];

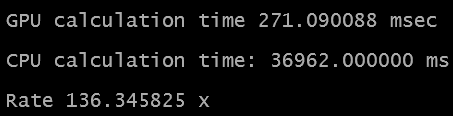


Рисунок 17 – Время решения задачи перемножения матриц примером SMEM-1 с данными float

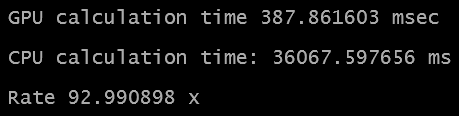


Рисунок 18 – Время решения задачи перемножения матриц примером SMEM-1 с данными double

В «функции-ядре» строки (SMEM-1):

\_\_shared\_\_ float as [BLOCK\_SIZE][BLOCK\_SIZE];

\_\_shared\_\_ float bs [BLOCK\_SIZE][BLOCK\_SIZE];

Заменили на строки (SMEM-2):

\_\_shared\_\_ float as [BLOCK\_SIZE][BLOCK\_SIZE + 1];

\_\_shared\_\_ float bs [BLOCK\_SIZE][BLOCK\_SIZE + 1];

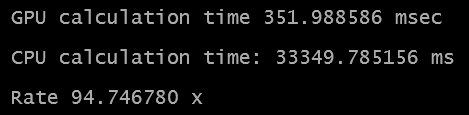


Рисунок 17 – Время решения задачи перемножения матриц примером SMEM-2 с данными float

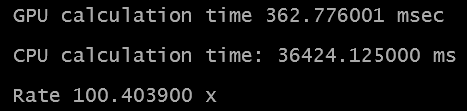


Рисунок 18 – Время решения задачи перемножения матриц примером SMEM-2 с данными double

Для тестирования нового метода SMEM вместо строк (SMEM-1, SMEM-2):

as [tx][ty] = a [ia + n \* ty + tx];

bs [tx][ty] = b [ib + n \* ty + tx];

sum += as [k][ty] \* bs [tx][k];

Поставим стоки (SMEM-3):

as [ty][tx] = a [ia + n \* ty + tx];

bs [ty][tx] = b [ib + n \* ty + tx];

sum += as [ty][k] \* bs [k][tx];

ind = tx + ty \* BLOCK\_SIZE – линейный номер нити

indM = tx + ty \* BLOCK\_SIZE – линейный номер элементов в матрицах «as» и «bs»

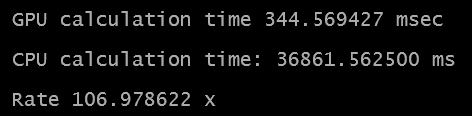


Рисунок 17 – Время решения задачи перемножения матриц примером SMEM-3 с данными float

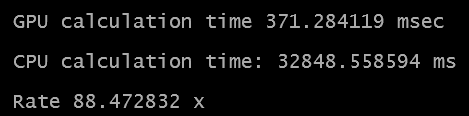


Рисунок 18 – Время решения задачи перемножения матриц примером SMEM-3 с данными double

Вместо строк ( SMEM-3 ):

float sum = 0.0f;

as [ty][tx] = a [ia + n \* ty + tx]; bs [ty][tx] = b [ib + n \* ty + tx];

sum += as [ty][k] \* bs [k][tx];

c [aBegin + bBegin + ty \* n + tx] = sum;

Поставим строки ( SMEM-4 ):

float sum1 = 0.0f, sum2 = 0.0f;

as [ty][tx] = a [ia + n \* ty + tx];

bs [ty][tx] = b [ib + n \* ty + tx];

as [ty + 16][tx] = a [ia + n \* ( ty + 16 ) + tx];

bs [ty + 16][tx] = b [ib + n \* ( ty + 16 ) + tx];

sum1 += as [ty][k] \* bs [k][tx];

sum2 += as [ty + 16][k] \* bs [k][tx];

c [aBegin + bBegin + ty \* n + tx] = sum1;

c [aBegin + bBegin + ( ty + 16 ) \* n + tx] = sum2;

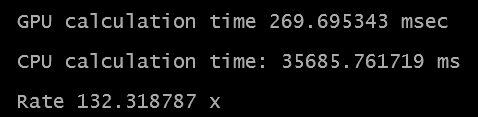


Рисунок 19 – Время решения задачи перемножения матриц примером SMEM-4 с данными float

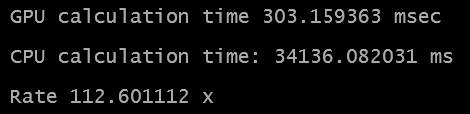


Рисунок 20 – Время решения задачи перемножения матриц примером SMEM-4 с данными double

Добавим новые строки при копировании данных:

float sum1 = 0.0f, sum2 = 0.0f, sum3 = 0.0f, sum4 = 0.0f;

as [ty][tx] = a [ia + n \* ty + tx];

bs [ty][tx] = b [ib + n \* ty + tx];

as [ty + 8][tx] = a [ia + n \* ( ty + 8 ) + tx];

bs [ty + 8][tx] = b [ib + n \* ( ty + 8 ) + tx];

as [ty + 16][tx] = a [ia + n \* ( ty + 16 ) + tx];

bs [ty + 16][tx] = b [ib + n \* ( ty + 16 ) + tx];

as [ty + 24][tx] = a [ia + n \* ( ty + 24 ) + tx];

bs [ty + 24][tx] = b [ib + n \* ( ty + 24 ) + tx];

зменения при перемножении матриц:

sum1 += as [ty][k] \* bs [k][tx];

sum2 += as [ty + 8][k] \* bs [k][tx];

sum3 += as [ty + 16][k] \* bs [k][tx];

sum4 += as [ty + 24][k] \* bs [k][tx];

при сохранении данных:

c [aBegin + bBegin + ty \* n + tx] = sum1;

c [aBegin + bBegin + ( ty + 8 ) \* n + tx] = sum2;

c [aBegin + bBegin + ( ty + 16 ) \* n + tx] = sum3;

c [aBegin + bBegin + ( ty + 24 ) \* n + tx] = sum4;

Добавим строки для блока нитей ( SMEM-5):

dim3 threads\_5 ( BLOCK\_SIZE, BLOCK\_SIZE / 4);

для новой «функции-ядра» ( SMEM-5 ):

kernel\_smem\_5 <<< blocks, threads\_5 >>> ( adev, bdev, N, cdev );

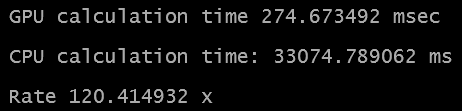


Рисунок 21 – Время решения задачи перемножения матриц примером SMEM-5 с данными float

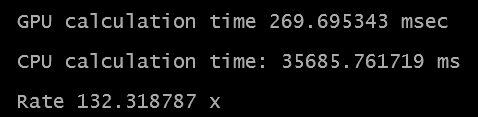


Рисунок 22 – Время решения задачи перемножения матриц примером SMEM-5 с данными double

**Обработка массивов**

Скорость вычисления заданной функции по двум массивам с полученной ошибкой, а также временем выполнения, представлена ниже.

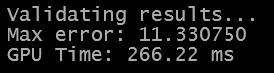


Рисунок 23 – Время выполнения задания обработки массивов

**Моделирование частиц с общей памятью**

Задача N-тел – моделировании поведения частиц, где время выполнения задачи, используя общую память, показано ниже.



Рисунок 24 – Время выполнения процесса моделирования частиц с общей памятью