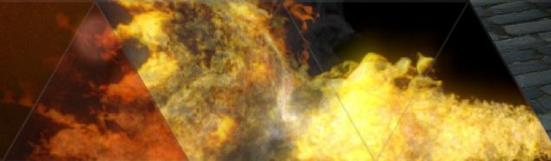
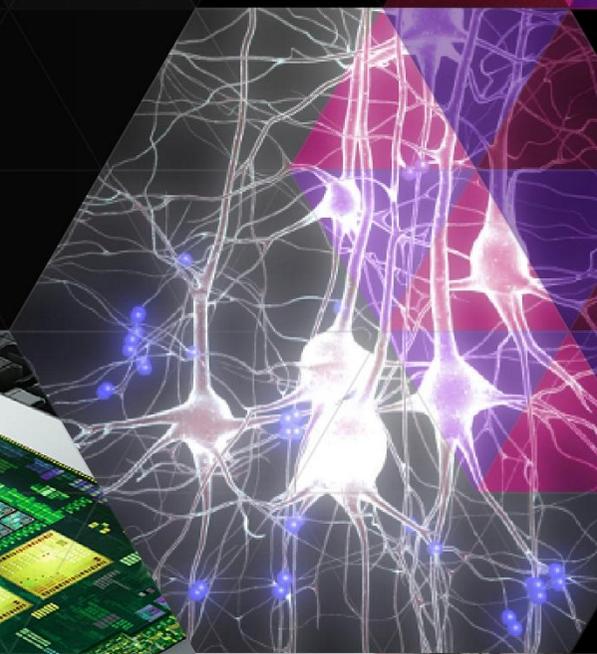
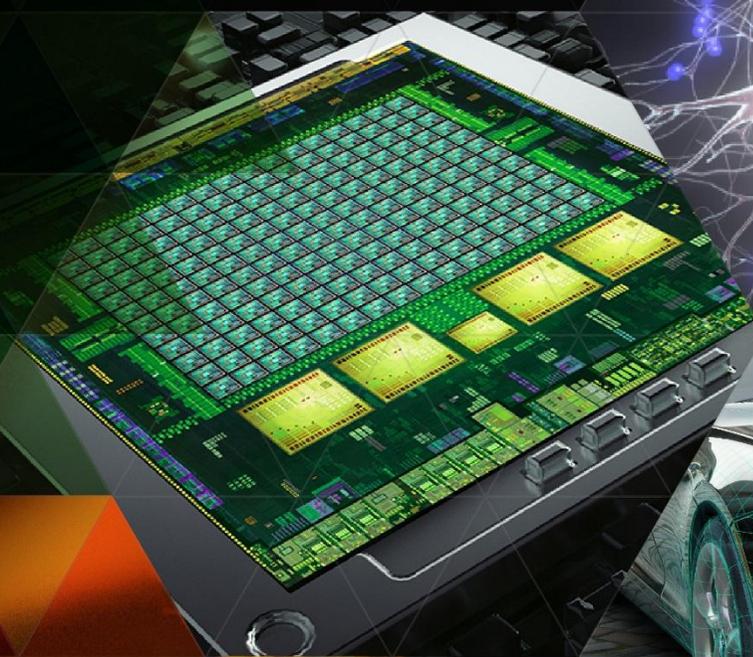




NVIDIA CUDA И OPENACC ЛЕКЦИЯ 2

Перепёлкин Евгений



СОДЕРЖАНИЕ

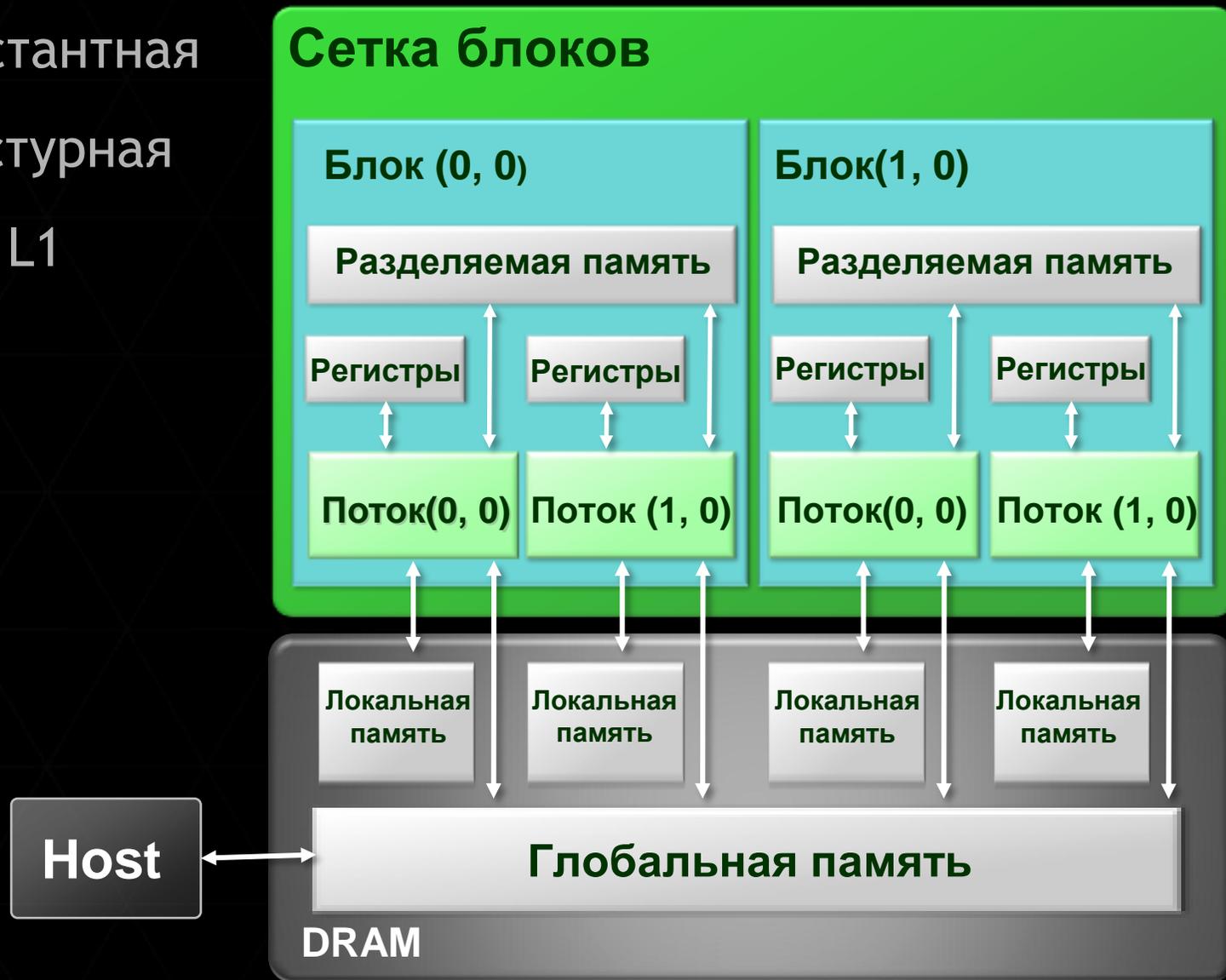
Лекция 2

- ▶ Иерархия памяти на GPU
- ▶ Регистры и локальная память
- ▶ Глобальная память
- ▶ Шаблон работы с глобальной памятью
- ▶ Использование pinned-памяти
- ▶ CUDA-потоки

Иерархия памяти на GPU

ДОСТУП К ПАМЯТИ НА GPU

- Константная
- Тектурная
- Кеш L1



Типы памяти в CUDA

Тип памяти	Доступ	Уровень выделения	Скорость работы
Register (регистровая)	RW	Per-thread	Высокая (on-chip)
Local (локальная)	RW	Per-thread	Низкая (DRAM)
Global (глобальная)	RW	Per-grid	Низкая (DRAM)
Shared (разделяемая)	RW	Per-block	Высокая (on-chip)
Constant (константная)	RO	Per-grid	Высокая (L1 cache)
Texture (текстурная)	RO	Per-grid	Высокая (L1 cache)

Типы памяти в CUDA

Тип памяти	Доступ	Уровень выделения	Скорость работы
Register (регистровая)	RW	Per-thread	Высокая (on-chip)
Local (локальная)	RW	Per-thread	Низкая (DRAM)
Global (глобальная)	RW	Per-grid	Низкая (DRAM)
Shared (разделяемая)	RW	Per-block	Высокая (on-chip)
Constant (константная)	RO	Per-grid	Высокая (L1 cache)
Texture (текстурная)	RO	Per-grid	Высокая (L1 cache)

Типы памяти в CUDA

Тип памяти	Доступ	Уровень выделения	Скорость работы
Register (регистровая)	RW	Per-thread	Высокая (on-chip)
Local (локальная)	RW	Per-thread	Низкая (DRAM)
Global (глобальная)	RW	Per-grid	Низкая (DRAM)
Shared (разделяемая)	RW	Per-block	Высокая (on-chip)
Constant (константная)	RO	Per-grid	Высокая (L1 cache)
Texture (текстурная)	RO	Per-grid	Высокая (L1 cache)

Шаблон работы с глобальной памятью

ШАБЛОН

работы с глобальной памятью

```
float *devPtr; // указатель на память на device

// выделение памяти на device
cudaMalloc ( (void **) &devPtr, N * sizeof ( float ) );

// копирование данных с host на device
cudaMemcpy ( devPtr, hostPtr, N * sizeof ( float ), cudaMemcpyHostToDevice );

// запуск функции-ядра

// копирование результатов с device на host
cudaMemcpy ( hostPtr, devPtr, N * sizeof ( float ), cudaMemcpyDeviceToHost );

// освобождение памяти
cudaFree ( devPtr );
```

ПРИМЕР 1

СЛОЖЕНИЯ ДВУХ МАССИВОВ

$$c_i = a_i + b_i,$$

$$a_i = \frac{1}{(i+1)^2}, b_i = e^{1/(i+1)}$$

$$i = 0, \dots, N - 1$$

- ▶ $N = 512 * 50\,000$
- ▶ 512 нитей в блоке, тогда 50 000 блоков

КОД ПРОГРАММЫ НА CUDA

Часть 1. Функция-ядро

```
__global__ void function (float *dA, float *dB, float *dC, int size )  
{  
    int i = blockIdx.x * blockDim.x + threadIdx.x;  
    if ( i < size ) dC [i] = dA[i] + dB[i];  
}
```

КОД ПРОГРАММЫ НА CUDA

Часть 2. Функция main

```
#include <stdlib.h>
#include <stdio.h>

int main( int argc, char* argv[] )
{ // инициализация переменных-событий для таймера
  float timerValueGPU, timerValueCPU;
  cudaEvent_t start, stop;
  cudaEventCreate ( &start );
  cudaEventCreate ( &stop );

  float *hA,*hB,*hC,*dA,*dB,*dC;
  int size = 512 * 50000; // размер каждого массива
  int N_thread = 512; // число нитей в блоке
  int N_blocks,i;
```

КОД ПРОГРАММЫ НА CUDA

Часть 3. Функция main

```
// задание массивов hA,hB,hC для host
unsigned int mem_size = sizeof(float)*size;
hA = (float*) malloc (mem_size);
hB = (float*) malloc (mem_size);
hC = (float*) malloc (mem_size);
// выделение памяти на device под массивы hA, hB, hC
cudaMalloc ((void**) &dA, mem_size);
cudaMalloc ((void**) &dB, mem_size);
cudaMalloc ((void**) &dC, mem_size);
// заполнение массивов hA,hB и обнуление hC
for ( i=0; i < size; i++ )
{hA[i] = 1.0f / ( ( i + 1.0f ) * ( i + 1.0f ) );
  hB[i] = expf ( 1.0f / ( i + 1.0f ) );
  hC[i]= 0.0f;
}
```

КОД ПРОГРАММЫ НА CUDA

Часть 4. Функция main

```
// определение числа блоков
if ((size % N_thread)==0)
{ N_blocks = size / N_thread;
} else
{ N_blocks = ( int )( size / N_thread )+1;
}
dim3 blocks( N_blocks );
```

Лекция 1

$$N_block = N / n_thread$$

или

$$N_block = (int) (N / n_thread) + 1$$

КОД ПРОГРАММЫ НА CUDA

Часть 5. Функция main

```
// -----GPU вариант -----  
// Старт таймера  
  cudaEventRecord ( start, 0 );  
// Копирование массивов с host на device  
  cudaMemcpy ( dA, hA, mem_size, cudaMemcpyHostToDevice );  
  cudaMemcpy ( dB, hB, mem_size, cudaMemcpyHostToDevice );  
// Запуск функции-ядра  
  function <<< N_blocks, N_thread >>> (dA, dB, dC, size);  
// Копирование результат с device на host  
  cudaMemcpy ( hC, dC, mem_size, cudaMemcpyDeviceToHost );  
// Остановка таймера и вывод времени  
// вычисления GPU варианта  
  cudaEventRecord ( stop, 0 );  
  cudaEventSynchronize ( stop );  
  cudaEventElapsedTime ( &timerValueGPU, start, stop );  
printf ( "\n GPU calculation time: %f ms\n", timerValueGPU );
```

КОД ПРОГРАММЫ НА CUDA

Часть 6. Функция main

```
// ----- CPU вариант -----  
// Старт таймера  
  cudaEventRecord ( start, 0 );  
// вычисления  
  for ( i = 0; i < size; i++ ) hC[i] = hA[i] + hB[i];  
// Остановка таймера и вывод времени  
// вычисления CPU варианта  
  cudaEventRecord ( stop, 0 );  
  cudaEventSynchronize ( stop );  
  cudaEventElapsedTime ( &timerValueCPU, start, stop );  
  printf ( "\n CPU calculation time: %f ms\n", timerValueCPU );  
// Вывод коэффициента ускорения  
  printf ( "\n Rate: %f x\n", timerValueCPU/timerValueGPU );
```

КОД ПРОГРАММЫ НА CUDA

Часть 7. Функция main

```
// Освобождение памяти на host и device
free ( hA );
free ( hB );
free ( hC );
cudaFree ( dA );
cudaFree ( dB );
cudaFree ( dC );
// уничтожение переменных-событий
cudaEventDestroy ( start );
cudaEventDestroy ( stop );

return 0;
}
```

РЕЗУЛЬТАТ 1.1

CPU Core2 Quad Q8300 2.5 ГГц ICC x64 1-ядро GPU Tesla K40c CUDA 6.0

GPU calculation time: 147.5 ms

CPU calculation time: 62 ms

Rate: 0.42 x

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

```
// -----GPU вариант -----  
// Копирование массивов с host на device  
cudaMemcpy ( dA, hA, mem_size, cudaMemcpyHostToDevice );  
cudaMemcpy ( dB, hB, mem_size, cudaMemcpyHostToDevice );  
  
// Старт таймера  
cudaEventRecord ( start, 0 );  
// Запуск функции-ядра  
function <<< N_blocks, N_thread >>> (dA, dB, dC, size);  
// Ожидание завершения выполнения функции-ядра  
cudaDeviceSynchronize ();  
// Остановка таймера и вывод времени  
// вычисления GPU варианта  
cudaEventRecord ( stop, 0 );  
cudaEventSynchronize ( stop );  
cudaEventElapsedTime ( &timerValueGPU, start, stop );  
printf ( "\n GPU calculation time: %f ms\n", timerValueGPU );
```

РЕЗУЛЬТАТ 1.2

CPU Core2 Quad Q8300 2.5 ГГц ICC x64 1-ядро GPU Tesla K40c CUDA 6.0

GPU calculation time: 1.56 ms

CPU calculation time: 62 ms

Rate: 39.7 x

Время расчета GPU-варианта включает в себя:

- ▶ выполнение «функции-ядра»;

ПРИМЕР 2

Вычисления сложной функции

$$c_i = \sin(\sin(a_i b_i)), i = 0, \dots, N - 1$$

В «функции-ядре» и в «main» функции вместо строчек

```
dC [i] = dA[i] + dB[i];
```

```
hC [i] = hA[i] + hB[i];
```

ПОСТАВИМ

```
dC [i] = sinf ( sinf ( dA[i] * dB[i] ) );
```

```
hC [i] = sinf ( sinf ( hA[i] * hB[i] ) );
```

РЕЗУЛЬТАТ 2.1

CPU Core2 Quad Q8300 2.5 ГГц ICC x64 1-ядро GPU Tesla K40c CUDA 6.0

GPU calculation time: 147 ms

CPU calculation time: 249 ms

Rate: 1.7 x

Время расчета GPU-варианта включает в себя:

- ▶ копирование данных с «host» на «device»;
- ▶ выполнение «функции-ядра»;
- ▶ копирование данных с «device» на «host».

РЕЗУЛЬТАТ 2.2

CPU Core2 Quad Q8300 2.5 ГГц ICC x64 1-ядро GPU Tesla K40c CUDA 6.0

GPU calculation time: 1.8 ms

CPU calculation time: 249 ms

Rate: 138 x

Время расчета GPU-варианта включает в себя:

- ▶ выполнение «функции-ядра»;

Использование pinned-памяти

PINNED-ПАМЯТЬ

`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)`

`cudaHostFree (b)`

`cudaHostUnregister (a)`

PINNED-ПАМЯТЬ

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 );
```

РЕЗУЛЬТАТ 1.3

CPU Core2 Quad Q8300 2.5 ГГц ICC x64 1-ядро GPU Tesla K40c CUDA 6.0

GPU calculation time: 53.6 ms (147.5)

CPU calculation time: 62 ms

Rate: 1.2 x (0.42)

Время расчета GPU-варианта включает в себя:

- ▶ копирование данных с «host» на «device»;
- ▶ выполнение «функции-ядра»;
- ▶ копирование данных с «device» на «host».

РЕЗУЛЬТАТ 2.3

CPU Core2 Quad Q8300 2.5 ГГц ICC x64 1-ядро GPU Tesla K40c CUDA 6.0

GPU calculation time: 53 ms (147)

CPU calculation time: 249 ms

Rate: 4.7 x (1.7)

Время расчета GPU-варианта включает в себя:

- ▶ копирование данных с «host» на «device»;
- ▶ выполнение «функции-ядра»;
- ▶ копирование данных с «device» на «host».

CUDA-номоку

CUDA-ПОТОКИ

- ▶ Асинхронное копирование данных `cudaMemcpyAsync`
- ▶ Использование `pinned`-памяти
- ▶ Задание `CUDA`-потоков

Лекция 2

```
My_Kernel <<< nBlock, nThread,  
              nShMem, nStream >>> ( param )
```

Шаблон

- ▶ Асинхронное копирование с «`host`» на «`device`»
- ▶ Асинхронный запуск «`функции-ядра`»
- ▶ Асинхронное копирование с «`device`» на «`host`»

ШАБЛОН (CUDA-ПОТОКИ)

```
// Создание двух CUDA-streams

cudaStream_t stream[2];

for ( int i = 0; i < 2; ++i ) cudaStreamCreate ( &stream[i] );

// Создание в pinned-памяти массива hostPtr

unsigned int mem_size= sizeof ( float ) * size;

float* hostPtr;

cudaMallocHost ( ( void** ) &hostPtr, 2 * mem_size );

// Резервирование на device места для массива hostPtr

float* inputDevPtr;

cudaMalloc ( ( void** ) &inputDevPtr, 2 * mem_size );
```

ШАБЛОН (CUDA-ПОТОКИ)

```
// Заполнение массива hostPtr
{...}

// Асинхронное копирование массива hostPtr
for ( int i = 0; i < 2; ++i ) cudaMemcpyAsync ( inputDevPtr + i * size,
        hostPtr + i * size, mem_size, cudaMemcpyHostToDevice, stream[i] );

// Обработка массива hostPtr на device
for ( int i = 0; i < 2; ++i ) myKernel <<< 100, 512, 0, stream[i] >>>
        ( outputDevPtr + i * size, inputDevPtr + i * size, size );

// Асинхронное копирование с device на host
for ( int i = 0; i < 2; ++i ) cudaMemcpyAsync ( hostPtr + i * size,
outputDevPtr + i * size, mem_size, cudaMemcpyDeviceToHost, stream[i] );
```

ШАБЛОН (CUDA-ПОТОКИ)

```
// Синхронизация CUDA-streams
```

```
cudaDeviceSynchronize ();
```

```
// Уничтожение CUDA-streams
```

```
for ( int i = 0; i < 2; ++i ) cudaStreamDestroy ( &stream[i] );
```

ПРИМЕР 3

обработка двух массивов

$$c_i = \sum_{j=0}^{99} \sin(a_i b_i + j),$$

$$a_i = \sin(i), b_i = \cos(2i - 5),$$

$$i = 0, \dots, N - 1$$

- ▶ $N = 512 * 50\,000$
- ▶ 512 нитей в блоке, тогда 50 000 блоков

ПРИМЕР 3 (КОД ПРОГРАММЫ)

Часть 1. Функция-ядро

```
__global__ void function (float *dA, float *dB, float *dC, int size )
{int i = blockIdx.x * blockDim.x + threadIdx.x;

  int j;

  float ab, sum = 0.f;

  if ( i < size )
  {ab = dA[i] * dB[i];

    for ( j = 0; j < 100; j++) sum = sum + sinf( j + ab );

    dC[i] = sum;

  } // if
}
```

ПРИМЕР 3 (КОД ПРОГРАММЫ)

Часть 2. Фрагмент функции main

```
...  
float *hA,*hB,*hC,*dA,*dB,*dC;  
int nStream = 4; // число CUDA-потоков  
int size = 512 * 50000 / nStream; // размер каждого массива  
int N_thread = 512; // число нитей в блоке  
int N_blocks,i;  
// выделение памяти для массивов hA,hB,hC для host  
unsigned int mem_size= sizeof(float)*size;  
cudaMallocHost ( (void**) &hA, mem_size * nStream );  
cudaMallocHost ( (void**) &hB, mem_size * nStream );  
cudaMallocHost ( (void**) &hC, mem_size * nStream );  
// выделение памяти на device под копии массивов hA, hB, hC  
cudaMalloc ( (void**) &dA, mem_size * nStream );  
cudaMalloc ( (void**) &dB, mem_size * nStream );  
cudaMalloc ( (void**) &dC, mem_size * nStream );
```

ПРИМЕР 3 (КОД ПРОГРАММЫ)

Часть 3. Фрагмент функции main

```
...  
// заполнение массивов hA, hB и обнуление hC  
for ( i=0; i < size; i++ )  
{hA[i] = sinf ( i ); hB[i] = cosf ( 2.0f * i - 5.0f ) ; hC[i] = 0.0f;  
}  
// Создание CUDA-streams  
cudaStream_t stream[4];  
for ( i = 0; i < nStream; ++i ) cudaStreamCreate ( &stream[i] );  
...
```


ПРИМЕР 3 (КОД ПРОГРАММЫ)

Часть 5. Фрагмент функции main

```
// Синхронизация CUDA-streams
  cudaDeviceSynchronize ();
// Уничтожение CUDA-streams
for ( i = 0; i < nStream; ++i ) cudaStreamDestroy ( &stream[i] );
...
return 0;
}
```

РЕЗУЛЬТАТ 3.1

CPU Core2 Quad Q8300 2.5 ГГц ICC x64 1-ядро GPU Tesla K40c CUDA 6.0

GPU calculation time: 102 ms

CPU calculation time: 9516 ms

Rate: 93 x

CUDA-Streams: 1

Время расчета GPU-варианта включает в себя:

- ▶ копирование данных с «host» на «device»;
- ▶ выполнение «функции-ядра»;
- ▶ копирование данных с «device» на «host».

РЕЗУЛЬТАТ 3.2

CPU Core2 Quad Q8300 2.5 ГГц ICC x64 1-ядро GPU Tesla K40c CUDA 6.0

GPU calculation time: 63 ms (102)

CPU calculation time: 9516 ms

Rate: 151 x (93)

CUDA-Streams: 4

Время расчета GPU-варианта включает в себя:

- ▶ копирование данных с «host» на «device»;
- ▶ выполнение «функции-ядра»;
- ▶ копирование данных с «device» на «host».