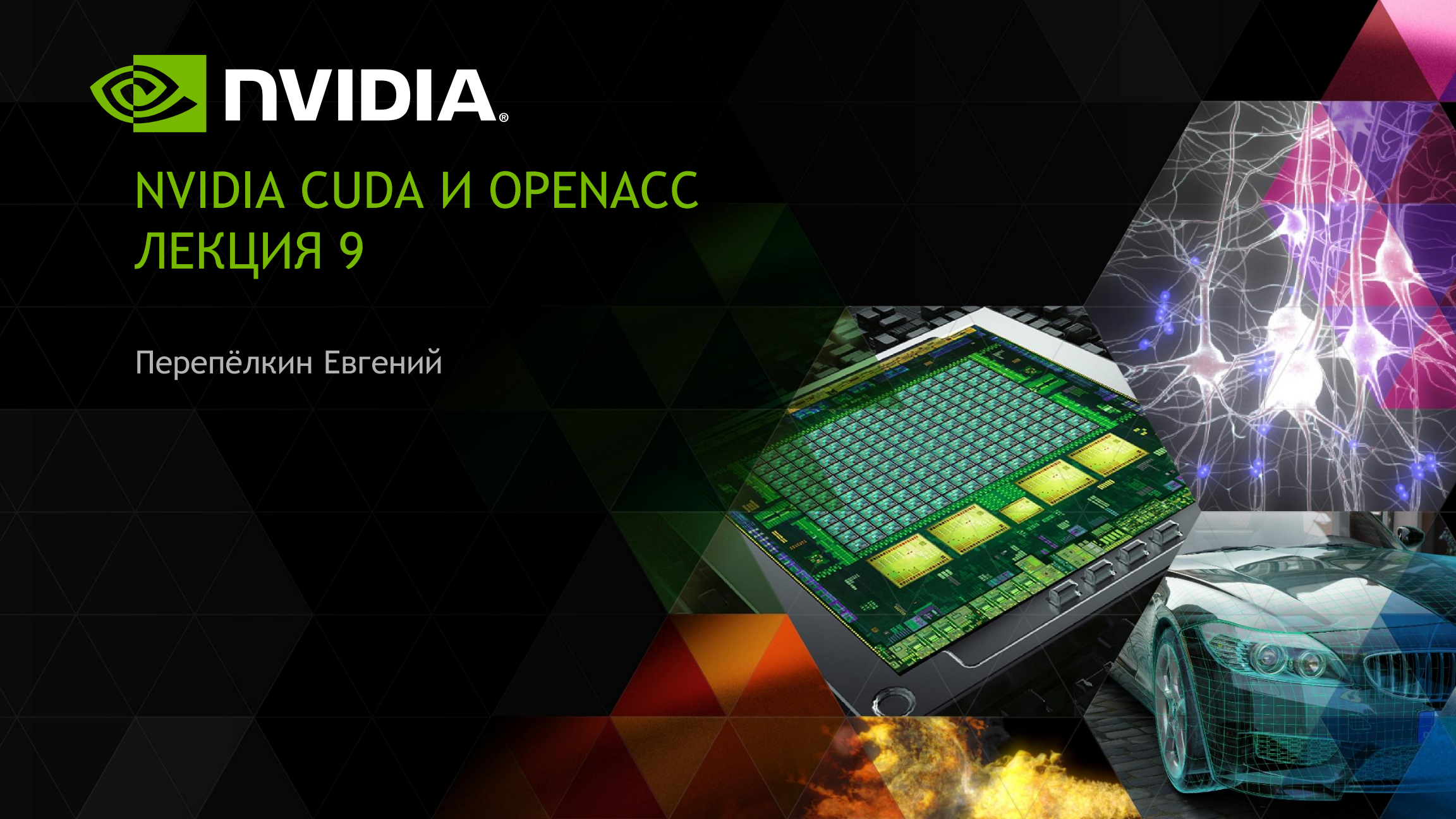

NVIDIA CUDA И OPENACC
ЛЕКЦИЯ 9

Перепёлкин Евгений

СОДЕРЖАНИЕ

Лекция 9

- Основные положения
- Работа с памятью
- Multi-GPU и OpenMP

Основные положения

ОСНОВНЫЕ ПОЛОЖЕНИЯ

Режим Multi-GPUs

- **`cudaGetDeviceCount`** – количество доступных GPUs на host
- **`cudaSetDevice`** – выбор текущего GPU
- CUDA-Streams и Events создаются и ассоциируются для текущего GPU
- Выделение памяти на GPU и запуск функции-ядра происходит для текущего GPU
- По умолчанию текущим GPU является GPU с номером 0
- Возможно копирование данных CUDA-Stream'ом, не ассоциированным с текущим GPU

ОСНОВНЫЕ ПОЛОЖЕНИЯ

Количество доступных GPUs на host

```
int deviceCount;
cudaGetDeviceCount(&deviceCount);
int device;

for (device = 0; device < deviceCount; ++device)
{cudaDeviceProp deviceProp;
 cudaGetDeviceProperties(&deviceProp, device);
 printf("Device %d has compute capability %d.%d.\n",device,deviceProp.major,
                                                    deviceProp.minor);
}
```

ОСНОВНЫЕ ПОЛОЖЕНИЯ

Выбор текущего GPU на host

```
size_t size = 1024 * sizeof(float);

cudaSetDevice(0); // Выбор текущим GPU-0
float *p0;
cudaMalloc(&p0, size); // Выделение памяти на GPU-0
MyKernel<<<1000, 128>>>(p0); // Запуск функции-ядра на GPU-0

cudaSetDevice(1); // Выбор текущим GPU-1
float *p1;
cudaMalloc(&p1, size); // Выделение памяти на GPU-1
MyKernel<<<1000, 128>>>(p1); // Запуск функции-ядра на GPU-1
```

ОСНОВНЫЕ ПОЛОЖЕНИЯ

Работа с GPUs через CUDA-Streams

```
cudaSetDevice(0); // Выбор текущим GPU-0
cudaStream_t s0;
cudaStreamCreate(&s0); // Создание CUDA-Stream s0 на GPU-0
MyKernel<<<100, 64, 0, s0>>>(); // Запуск функции-ядра на GPU-0
                                // из CUDA-Stream s0

cudaSetDevice(1); // Выбор текущим GPU-1
cudaStream_t s1;
cudaStreamCreate(&s1); // Создание CUDA-Stream s1 на GPU-1
MyKernel<<<100, 64, 0, s1>>>(); // Запуск функции-ядра на GPU-1
                                // из CUDA-Stream s1

// Невозможный запуск функции-ядра на GPU-1 из CUDA-Stream s0
MyKernel<<<100, 64, 0, s0>>>(); !!!
```

Работа с памятью

РАБОТА С ПАМЯТЬЮ

Доступ к памяти «Peer-to-Peer»

- При запуске приложения в 64-битном режиме на видеокартах серии Tesla с CC 2.x и выше функция-ядро, запущенная на одном GPU может на прямую обращаться к памяти другого GPU
- Возможность Peer-to-Peer доступа к памяти между двумя видеокартами определяется функцией **`cudaDeviceEnablePeerAccess`**
- Каждое GPU максимально может поддерживать 8 Peer-to-Peer соединений

РАБОТА С ПАМЯТЬЮ

Доступ к памяти «Peer-to-Peer»

```
cudaSetDevice(0); // Установка текущим GPU-0
float *p0;
size_t size = 1024 * sizeof(float);
cudaMalloc(&p0, size); // Выделение памяти на GPU-0
MyKernel<<<1000, 128>>>(p0); // Запуск функции-ядра на GPU-0

cudaSetDevice(1); // Установка текущим GPU-1
cudaDeviceEnablePeerAccess(0, 0); // Установка peer-to-peer доступа к GPU-0

// Запуск функции-ядра на GPU-1, имеющей доступ к данным p0, расположенным
// в памяти GPU-0
MyKernel<<<1000, 128>>>(p0);
```

РАБОТА С ПАМЯТЬЮ

Копирование памяти «Peer-to-Peer»

- Копирование данных между двумя видеокартами, поддерживающими UVA режим может осуществляться стандартными методами, описанными в лекции 3
- Также можно воспользоваться функциями
  - `cudaMemcpyPeer, cudaMemcpyPeerAsync`
  - `cudaMemcpy3DPeer, cudaMemcpy3DPeerAsync`
- Если peer-to-peer доступ между двумя видеокартами возможен (`cudaDeviceEnablePeerAccess`), тогда использование памяти host не обязательно, что ускоряет процесс

РАБОТА С ПАМЯТЬЮ

Копирование памяти «Peer-to-Peer»

```
cudaSetDevice(0); // Установка текущим GPU-0
float *p0;
size_t size = 1024 * sizeof(float);
cudaMalloc(&p0, size); // Выделение памяти на GPU-0

cudaSetDevice(1); // Установка текущим GPU-1
float *p1;
cudaMalloc(&p1, size); // Выделение памяти на GPU-1

cudaSetDevice(0); // Установка текущим GPU-0
MyKernel<<<1000, 128>>>(p0); // Запуск функции-ядра на GPU-0

cudaSetDevice(1); // Установка текущим GPU-1
cudaMemcpyPeer(p1, 1, p0, 0, size); // Копирование p0 в p1
MyKernel<<<1000, 128>>>(p1); // Запуск функции-ядра на GPU-1
```

Multi-GPU u OpenMP

MULTI-GPU И OPENMP

Шаблон работы

```
int nElem = 1024; // размер задачи
cudaGetDeviceCount(&num_gpus);
if(num_gpus >= 1)
{omp_set_num_threads(num_gpus); // задание количества CPU-потоков
 #pragma omp parallel // OpenMP дирректива
 {unsigned int cpu_thread_id = omp_get_thread_num();
  unsigned int num_cpu_threads = omp_get_num_threads();

  cudaSetDevice (cpu_thread_id % num_gpus);
  int threadNum = nElem / num_cpu_threads;
  int startIdx = cpu_thread_id * threadNum;

  dim3 gpu_threads (128);
  dim3 gpu_blocks (threadNum/gpu_threads.x);
  Kernel<<<gpu_blocks,gpu_threads>>>(pData, startIdx, threadNum);
  ...
 }
}
```