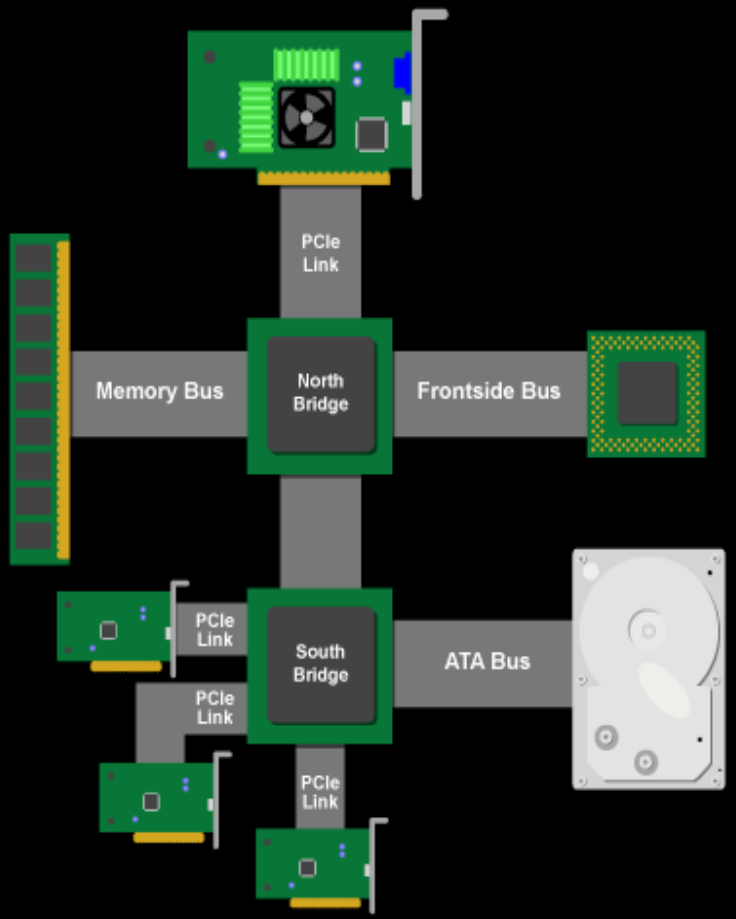
Архитектура и программирование потоковых многоядерных процессоров для научных расчётов

Лекция 2. GPU в составе компьютера
Программная модель CUDA (начало)
Пример программы

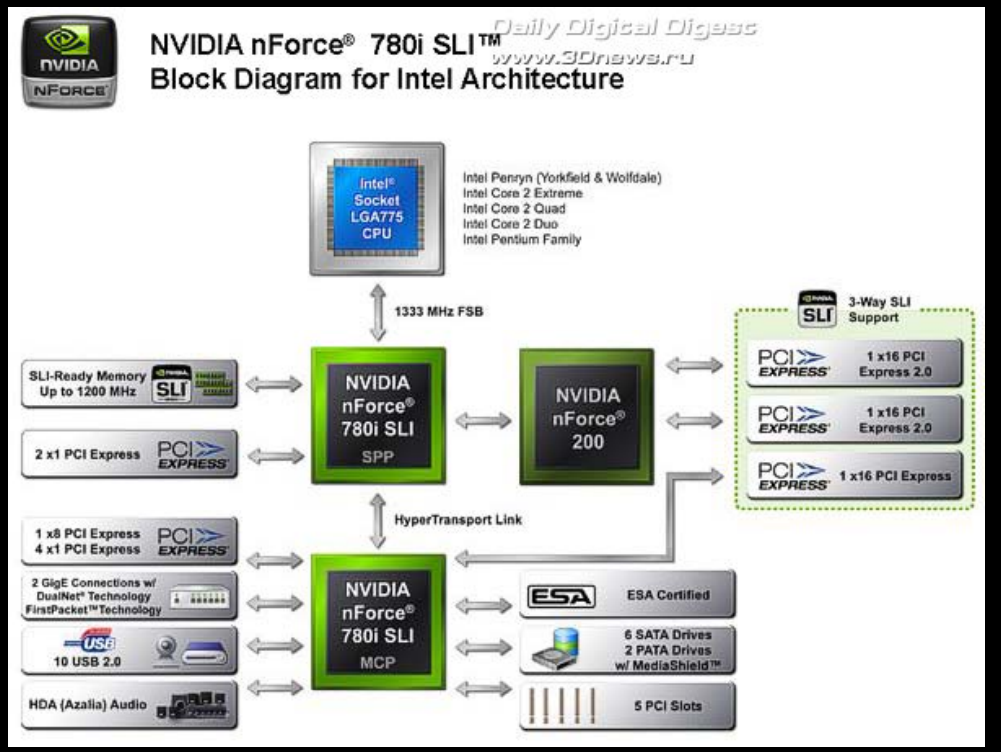
Средства обмена данными в компьютере

- Обмен данными – важнейшая составляющая компьютера
  - Примеры: многопроцессорные системы, FPGA etc.
- По традиции отдельные устройства имеют разные возможности (уровни и способы) обмена данными
- Традиционная архитектура ориентирована на одно, центральное счётное устройство



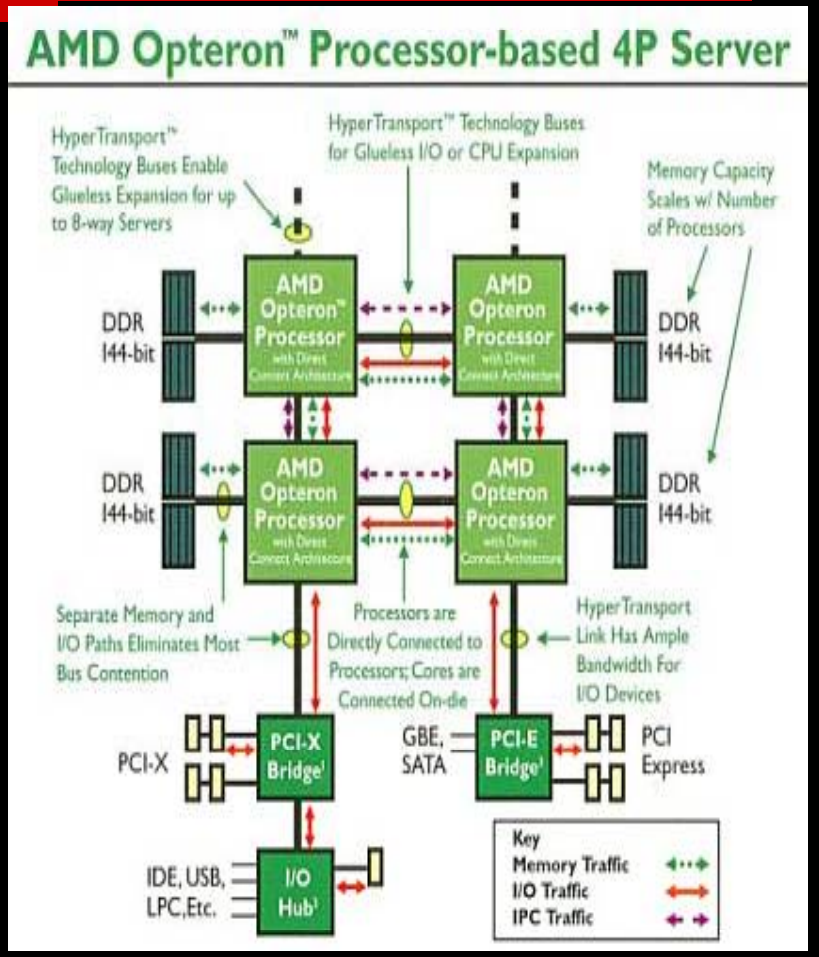
Архитектура Intel

- Традиции
  - Сев. Мост – общение с памятью и быстрыми шинами
  - Юж. Мост – общение с низко-скоростной периферией
- Юж. мост не может напрямую общаться с CPU
- Иерархическая структура



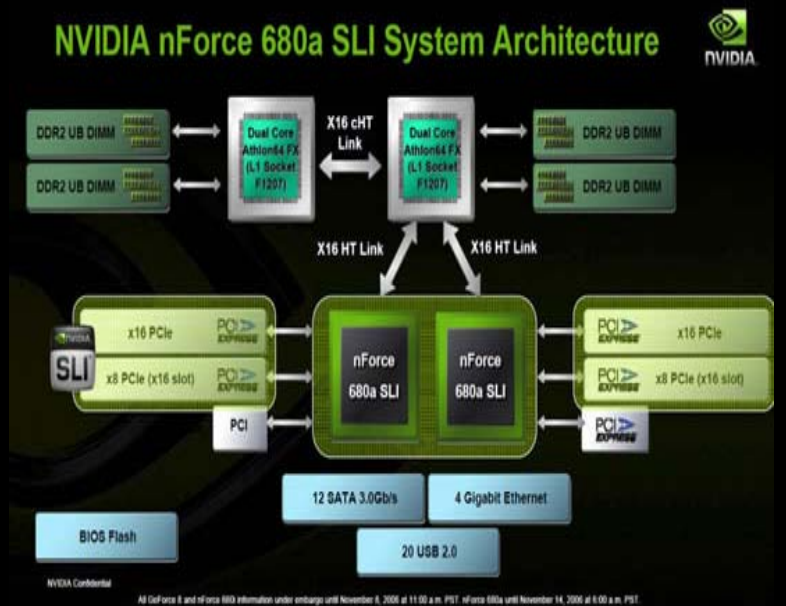
Архитектура AMD

- HyperTransport – шаг к симметричной распределённой архитектуре
- HyperTransport
  - Обмен пакетами
  - Выделенные линии в каждом из направлений
  - 8 Gb/s по каждой линии
  - Сев. Мост реализован на кристалле
- Юж. Мост - общение с внешними устройствами, сам подключен к CPU напрямую
  - Меньше латентность из-за связующей логики

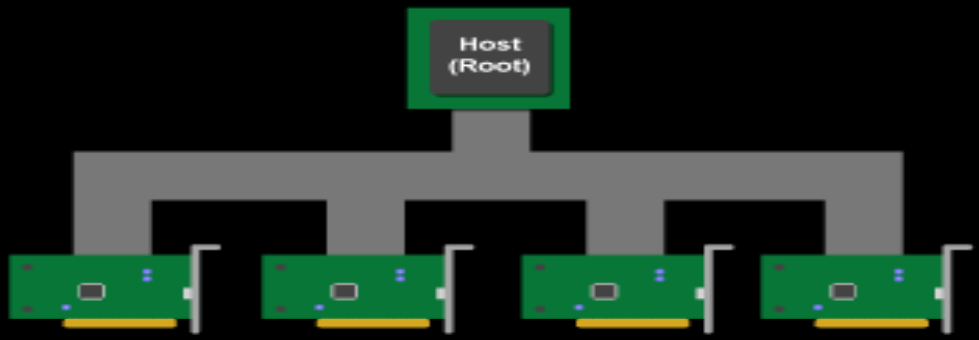


Архитектура AMD (часть 2)

- Dual Socket Direct Conect архитектура
  - Каждый из CPU имеет свою собственную память
- Два независимых MCP (Media & Communication Processors)
- Распределённая архитектура
- Не совсем – среди процессоров есть вы деленый, соединённый с MCP по двум шинам HypTrans x16



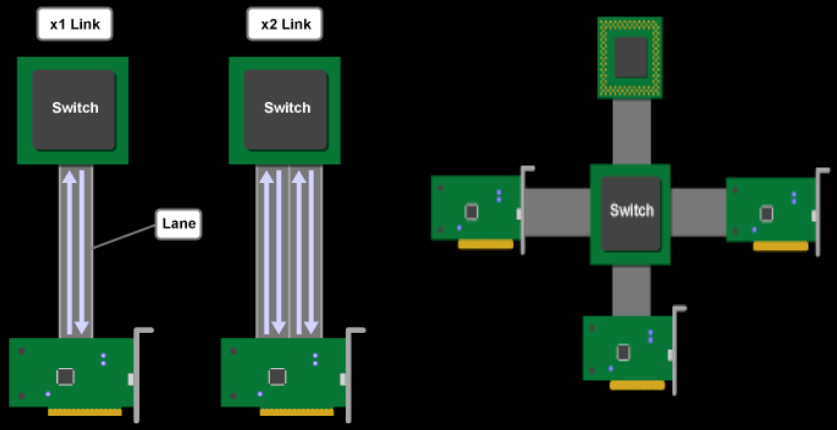
Шина PCI



- Connected to the southBridge
- Originally 33 MHz, 32-bit wide, 132 MB/second peak transfer rate
- More recently 66 MHz, 64-bit, 512 MB/second peak
- Upstream bandwidth remain slow for device (256MB/s peak)
- Shared bus with arbitration
- Winner of arbitration becomes bus master and can connect to CPU or DRAM through the southbridge and northbridge

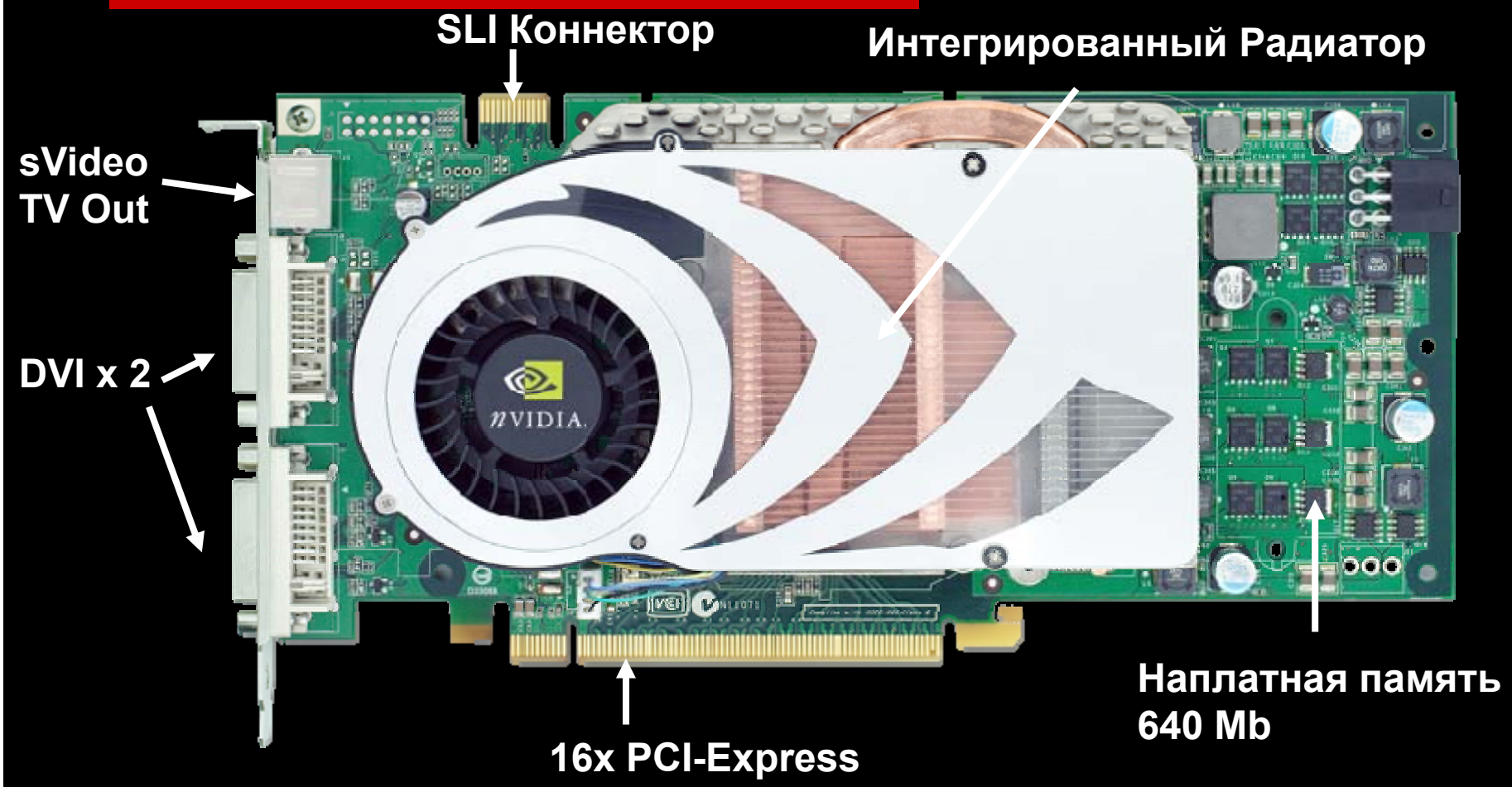
Шина PCI-Express

- Switched, point-to-point connection
- Each card has a dedicated "link" to the central switch, no bus arbitration.
- Packet switches messages form virtual channel
- Prioritized packets for QoS
- Each link consists of one more lanes



- Each lane is 1-bit wide (4 wires, each 2-wire pair can transmit 2.5Gb/s in one direction)
- Upstream and downstream now simultaneous and symmetric
- Each Link can combine 1, 2, 4, 8, 12, 16 lanes- x1, x2, etc.
- Each byte data is 8b/10b encoded into 10 bits with equal number of 1's and 0's; net data rate 2 Gb/s per lane each way.
- Thus, the net data rates are 250 MB/s (x1) 500 MB/s (x2), 1GB/s (x4), 2 GB/s (x8), 4 GB/s (x16), each way

Графическая плата NVIDIA



NVIDIA
World Leader in Visual Computing Technologies

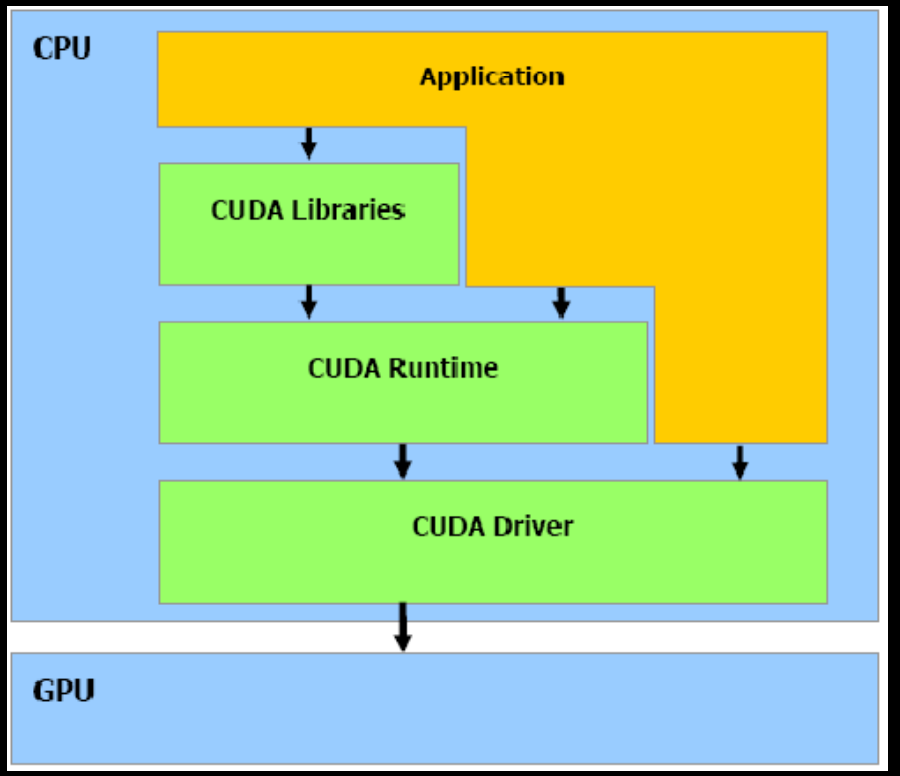
Программный стек CUDA

- Программы могут использовать GPU посредством:
  - Обращения к стандартным функциям библиотек (BLAS, FFTW)
    - cublas.dll (cublasemu.dll)
    - cufft.dll (cufftemu.dll)

    \+ очень просто

    \- НЕ очень эффективно

  - Использования CUDA runtime API
  - Использования CUDA driver API



Application Program Interface (кратко)

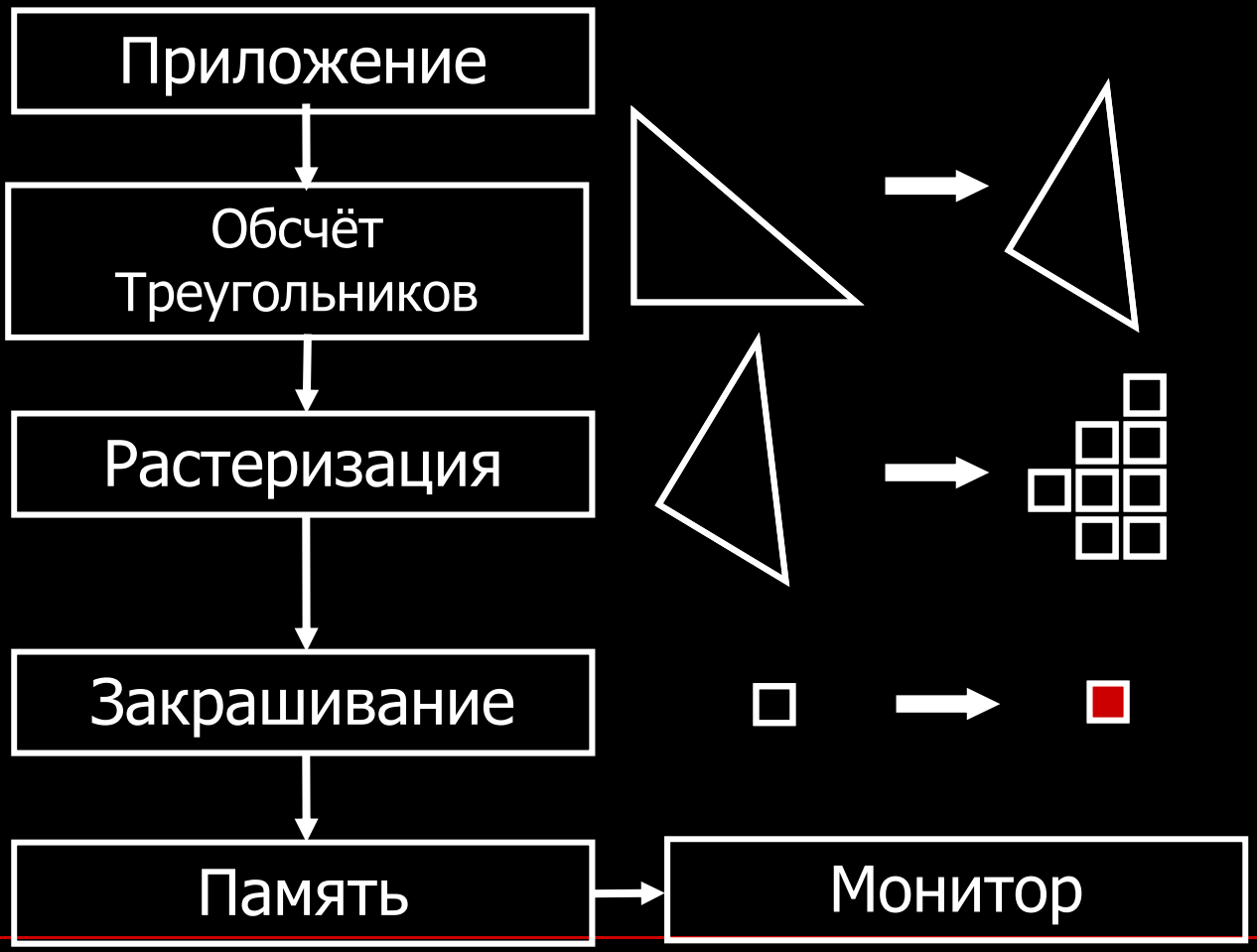
- API нужен для абстракции программного интерфейса от кода пользователя
- CUDA driver API (функции cu*)
  - cuda.dll
  - Низкий уровень
    - Тяжелее программировать
    - Больший контроль над процессом
- CUDA runtime API (функции cuda*)
  - cudart.dll
  - Более абстрактно чем driver API (простота)
  - Возможность использовать эмуляцию устройства

- ! Смешивать нельзя !

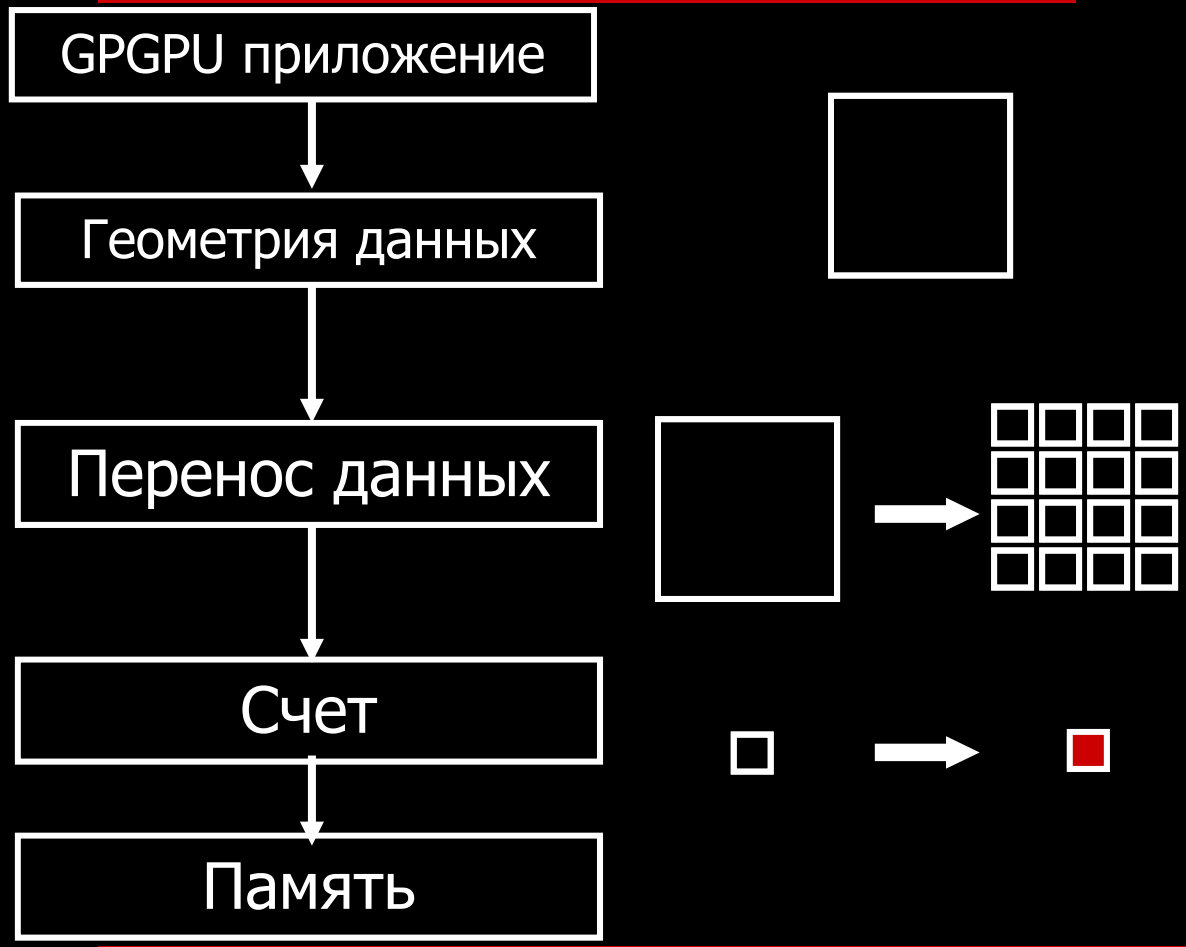
Интерфейс Разработчика

- Необходимые элементы
  - 1а. Драйвер видеокарты, поддерживающий CUDA
  - 1б. Использование режима “эмуляции” (nvcc -deviceemu)
  - 2. CUDA Toolkit - содержит
    - Драйвер компилятора nvcc
    - Виртуальную машину ptxas
    - Библиотеки подключаемых исполняемых модулей (*.dll, *.o)
  - 3. CUDA SDK - содержит
    - Примеры программных проектов
    - Вспомогательные утилиты (библиотеки, Occupancy Calculator)
- **Всё это богатство доступно для свободного скачивания**
  - http://www.nvidia.com/object/cuda_get.html
  - Исполняемые модули (без исходных листингов) для WinXP (x86, x86_64), Fedora, RHEL, SUSE, Open SUSE (x86, x86_64), Mac OS X (10.5.2)

Модель программирования графических приложений

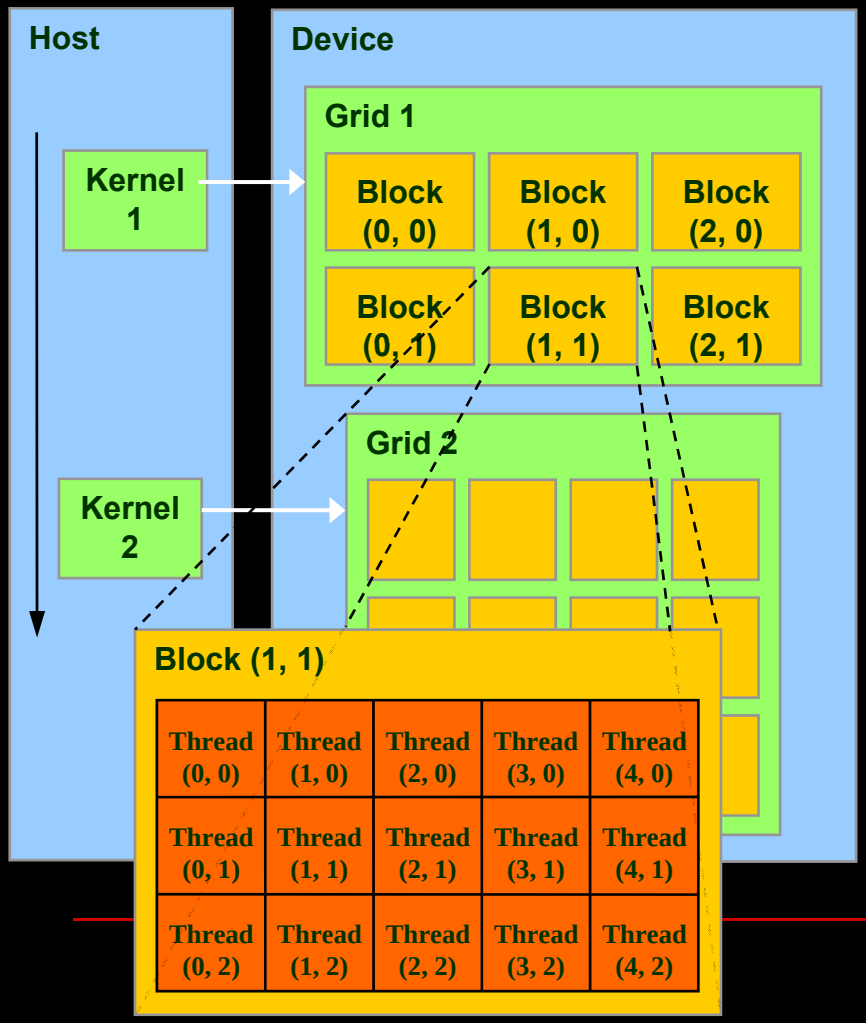


Программирование приложений при помощи графич. процессора



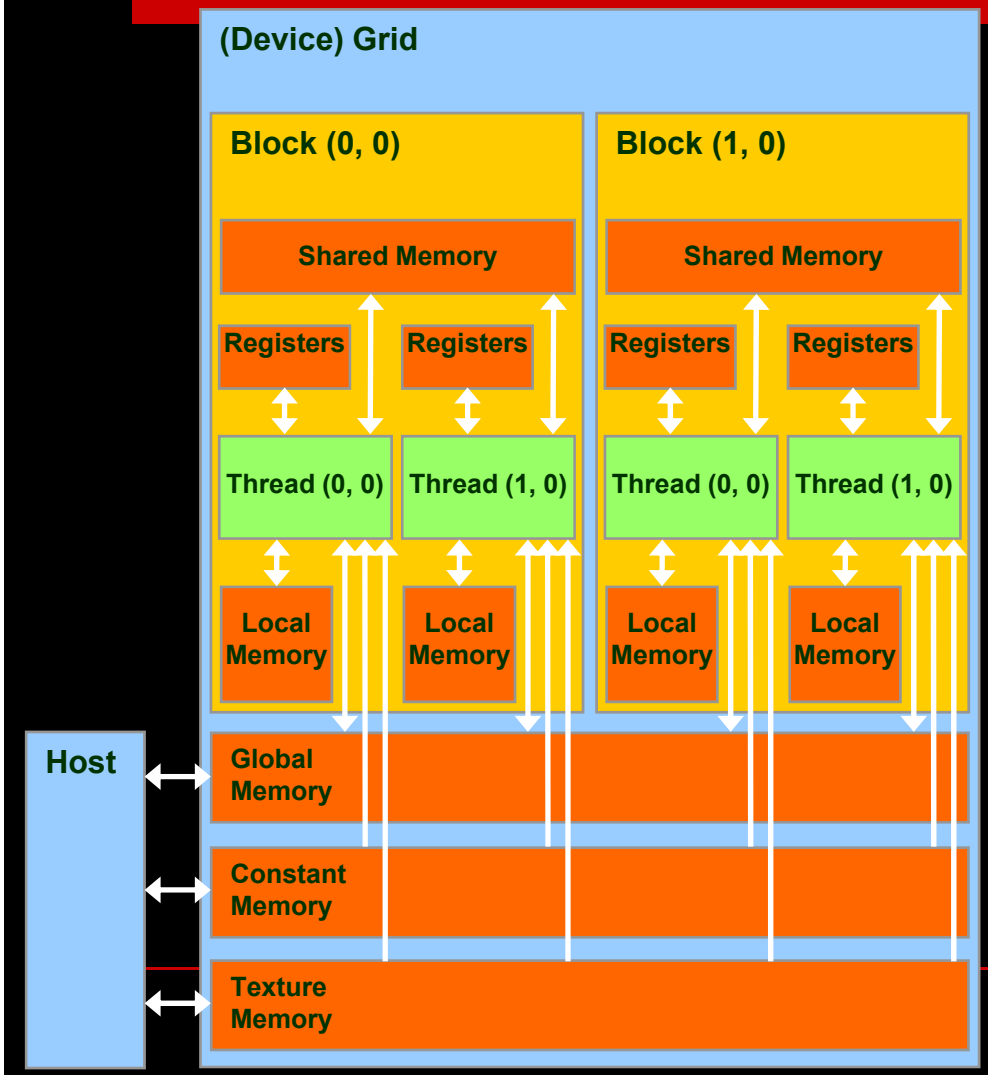
- **Схема обработки данных схожа с методом обработки графических данных**
- **Необходимость планирования “геометрии данных” при реализации алгоритма**

Вычислительная конфигурация GPU



- Процессы объединяются в блоки (**blocks**), внутри которых они имеют общую память (**shared memory**) и синхронное исполнение
- Блоки объединяются в сетки (**grids**)
  - Нет возможности предсказать очерёдность запуска блоков в сетки
  - Между блоками нет и не может быть (см. выше) общей памяти

Модель памяти GPU



- GPU может читать
  - Constant Memory
  - Texture Memory
- GPU может читать/писать
  - Global Memory
- Каждый из процессов ч/п
  - Local Memory
  - Registers
- Каждый из процессов внутри блока ч/п
  - Shared memory
  - **Gather/Scatter MA pattern**
- Хост имеет возможность читать/писать:
  - Global Memory
  - Constant Memory
  - Texture Memory

C модификация – декларация функций

	Executed on the:	Only callable from the:
`__device__ float DeviceFunc()`	device	device
`__global__ void KernelFunc()`	device	host
`__host__ float HostFunc()`	host	host

- **__global__** определяет вычислительное ядро, исполняемое **устройством** и вызываемое **хостом**
  - Ядро обязательно должно возвращать **void**
- **__device__** и **__host__** можно использовать совместно
- **__host__** можно опускать

Ограничения на C в программах устройства

- __device__ и __global__ **не** могут
  - содержать рекурсий
  - Объявлять static переменных внутри себя
  - Иметь переменное количество аргументов
- Адрес функции __device__ нельзя использовать при указании на функцию (а __global__ **можно**)
- __global__ и __host__ не могут быть использованы одновременно
- При вызове __global__ **обязательно** указывать конфигурацию вызова

Конфигурация вызова вычислительного ядра

<<<Gs,Bs,Ms>>>

- Определение размерностей сетки блоков и блока тредов для данного ядра
  - Gs (grid size) тип - dim3 - размерность сетки блоков
  - Bs (Block size) тип - dim3 - размерность ,блока тредов
  - Ms (shared Memory size) – тип size_t – количество общей памяти динамически аллоцированной под данный вызов (можно опускать)

- **!!! GridId BlockID при исполнении указывают конкретному треду на его контекст в рамках общей задачи**

```
__global__ void functionA (type argument);
functionA<<<Gs,Bs,Ms>>>(arg1);
```

Подготовка устройства

- Инициализация устройства
  - cudaGetDeviceCount, cudaSetDevice, cudaGetDevice
    - Позволяют правильно выбрать между устройствами

- Выделение памяти
  - **cudaMalloc**
  - ```
    if(cudaMalloc((void**)&bvCU,(size_t)neurs*sizeof(FLT32))!=cudaSuccess)
    throw  Unsupported(this->Name,"CUDA has failed to allocate BV.");
    ```

  - **cudaMallocPitch**
  - ```
    if(cudaMallocPitch((void**)&wmCU,(size_t*)&wm_pitch,ins*sizeof(FLT32),neurs
    )!=cudaSuccess) throw  Unsupported(this->Name,"CUDA has failed to allocate
    WM.");
    ```

Пересылка данных

- Копирование между хостом и устройством
  - **cudaMemset**
  - ```
    if(cudaMemset(ldCU,0,ns*sizeof(float))!=cudaSuccess)
    throw  Unsupported(this->Name,"CUDA has failed to zero-init LD.");
    ```
  - **cudaMemcpy**
  - ```
    if(cudaMemcpy(bvCU,bvF32,ns*sizeof(FLT32),cudaMemcpyHostToDevice)!=cudaSuccess) throw Unsupported(this->Name,"CUDA failed to upload BV.");
    ```
- Освобождение памяти
  - **cudaFree**
  - ```
    if(cudaFree(wmCU)!=cudaSuccess)
    throw  Unsupported(this->Name,"CUDA has failed to de-allocate WM.");
    ```
    ```

Вызов CUDA из произвольного хост-кода (пример)

```
void fflayerCUDA::backpropagate()
{
/*   U16 i;
     for(i=0;i<ns;i++)
          for(U16 k=0;k<outSize[0]/ns;k++)
                    ld[i]+=Output[0][i+k];
(…)
     memset(Input[0],0,inSize[0]*memSize[0]*sizeof(FLT64));
     for (i=0;i<ns;i++)
          for(U16 k=0;k<inSize[0];k++)
                    Input[0][k]+=ld[i]*wm[i+k*ns]; */
if(LocLayerID==1)
{
     for(U32 ii=0;ii<ns;ii++) ldF32[ii]=(FLT32)((FLT64**)Output)[0][ii];
     if(cudaMemcpy(ldCU,ldF32,ns*sizeof(FLT32),cudaMemcpyHostToDevice)!=cudaSuccess)
          throw Unsupported(this->Name,"CUDA failed to upload LD.");
}

CUDAbackpropagate(ldCU,loCU,wmCU,wmdCU,bvdCU,inCU,indCU,inSize[0],ns,wm_pitch/sizeof(float));

}
```

Функция, использующая CUDA – пример (файл *.cu)

```
void CUDAbackpropagate(float *ld,float *lo,float *wm,float *wmd,float *bvd,float *in, float
     *ind, int insize,int outsize,int wm_pitch)
{
dim3 threads;
dim3 grid;
int repetit=1;
threads.x = MAX_THREAD_PER_BLOCK;
repetit = outsize/MAX_THREAD_PER_BLOCK;
if(repetit*MAX_THREAD_PER_BLOCK<outsize) repetit++;
if(insize<=MAX_BLOCKS_PER_GRID_DIM) {grid.x = insize;}
else {printf("CUDA device: Can't handle layers with more than 65535 inputs (extendible to
     65535^3)\n");
     return;}

CUDA_SAFE_CALL( cudaThreadSynchronize() );
CUDAbackpropagateCALC<<<grid,
     threads,threads.x*sizeof(float)>>>(ld,lo,wm,wmd,bvd,in,ind,insize,outsize,repetit,wm_pitc
     h);
CUDA_SAFE_CALL( cudaThreadSynchronize() );
}
```

Вычислительное ядро - пример (файл *.cu)

```
__global__ void CUDAupdateCALC(float *wm,float *wmd,float *bv,float *bvd,float
      lr,float mom, int insize,int outsize,int r,int wm_pitch)

{
      int i=0;
      for(i=0;i<r;i++)
      {
            int idx = threadIdx.x+blockDim.x*i;
            if(idx<outsize)
            {
                  wm[idx*wm_pitch+blockIdx.x]-=lr*wmd[idx*wm_pitch+blockIdx.x];
                  if(blockIdx.x==0) bv[idx] -= lr*bvd[idx];
            }
      }
      __syncthreads();

}
```

Итоги лекции

- **В результате лекции Вы должны :**
  - **Понимать возможности использования GPU для расчётов с точки зрения пропускной способности системы обмена данными компьютера**
  - **Иметь понятие об организации разработки приложений**
    - **Интрефейс разработчика**
    - **API стек**
    - **Модификации языка C используемые для конфигурации устройства**
  - **Понять приведенный пример программы использующей CUDA**
  - **Достаточные знания для начала самостоятельной работы**