CUDA

Parallel programming on pixels

Disclaimer

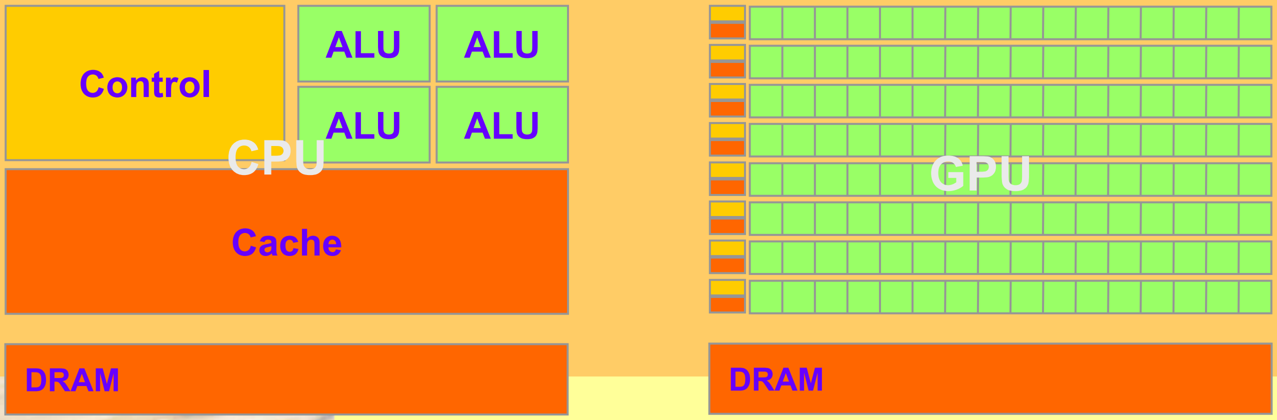
- All pictures in this PowerPoint file are taken from the slides by professor Wen-Mei W. Hwu, distributed by NVidia.

GPU

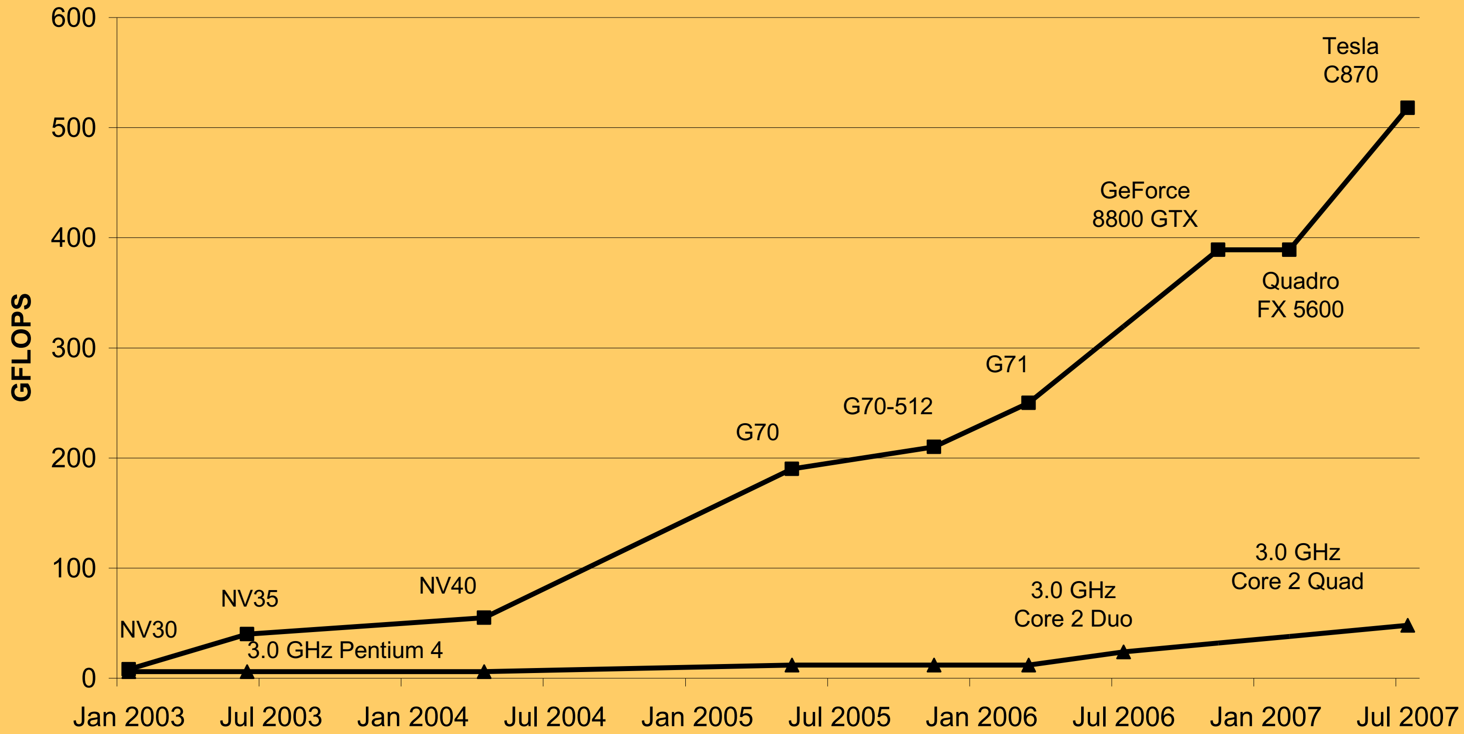
- Graphical Processing Unit
- Used to display 3D graphics on your PC
- Consists of very a very large number of transistors.
- Highly parallel computation
- Cost-effective
  - The only part that you might consider updating in your PC.

CPU vs. GPU

- Less data cache and flow control
- More Computation



Performance in Gflops



GPGPU

- General Purpose computation using GPU -- not only for pixels.

Advantages

- Large data arrays, streaming throughput
- Very large memory bandwidth
- Fine-grain SIMD parallelism
  - A very large number of threads
- Low-latency floating point (FP) computation
  - High power computation capability
- Piggyback on the fast advancing GPU technology

GPGPU Applications

- Tons of application on GPGPU.org and Nvidia websites.
- http://www.nvidia.com/object/cuda_home.html#
  - Game effects (FX) physics
  - image processing
  - Physical modeling
  - computational engineering
  - matrix algebra
  - Convolution
  - correlation, sorting

Implication

- Suitable for data-parallel processing
  - The same computation is performed on many data elements in parallel
  - Low control flow overhead
  - high floating point arithmetic intensity
- Computation intensive threads on large number of data hide large memory latency for each
  - No need for large data cache

Why not before?

- Graphics API only
- Addressing modes is limited by texture size/dimension
- Limited outputs due to shader capabilities
- No instruction on integer and bits
- No interaction between pixels

CUDA in a Long Sentence

- Compute Unified Device Architecture
- A general purpose parallel computing architecture that leverages the parallel compute engine in NVIDIA graphics processing units (GPUs) to solve many complex computational problems in a fraction of the time required on a CPU.

CUDA in a Short Sentence

- Integrates CPU+GPU application C program by running serial parts on CPU and highly parallel parts on GPU.

CUDA in You PC

- Can run on your PC with NVIDIA display card.
  - NVIDIA CUDA-enabled products
  - http://www.nvidia.com/object/cuda_learn_products.html
- CUDA download
  - http://www.nvidia.com/object/cuda_get.html

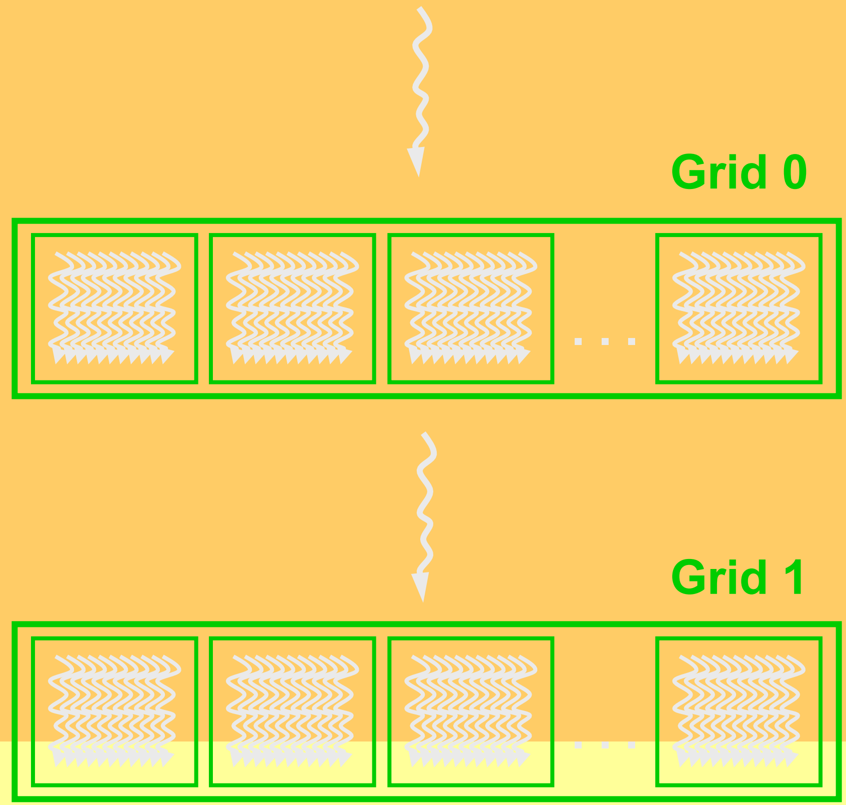
An Illustration

CPU Serial Code

GPU Parallel Kernel
KernelA<<< nBlk, nTid >>>(args);

CPU Serial Code

GPU Parallel Kernel
KernelB<<< nBlk, nTid >>>(args);

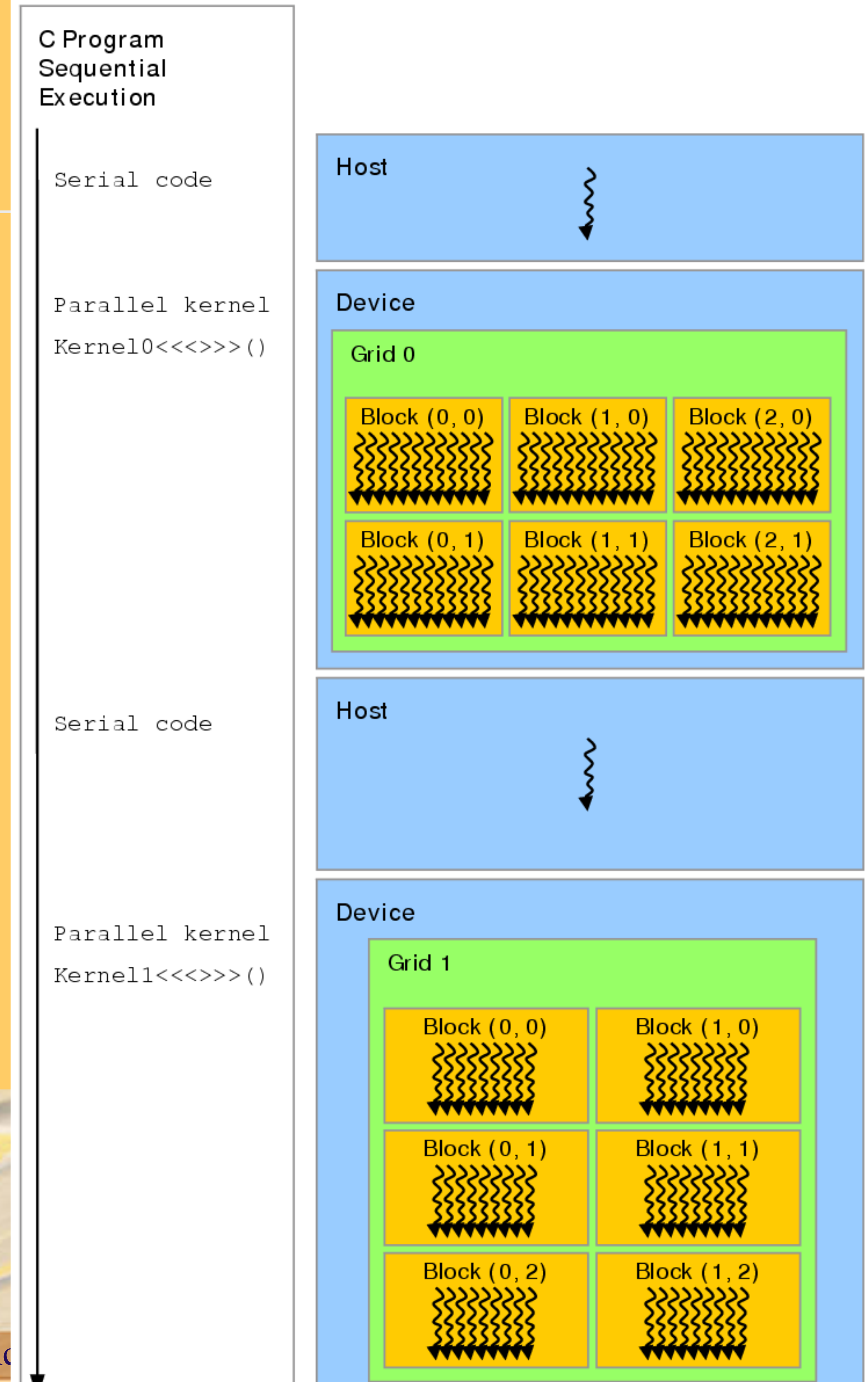


Terminology

- Host
  - CPU
- Device
  - GPU
- Kernel
  - Computation intensive C functions running on devices as threads.
- Grid
  - The view of cores within a device
- Block
  - A basic unit in a device

The Programming Model

- The *host* runs the *sequential* part of the application.
- The *parallel* data intensive parts are written as *kernel* functions, and sent to *devices* for execution (as *threads*).
- The *blocks* are the unit for computation and they are organized as a *grid*.


C Program
Sequential
Execution
Serial code
Parallel kernel
Kernel0<<<>>>()
Serial code
Parallel kernel
Kernel1<<<>>>()
Host
Device
Grid 0
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Host
Device
Grid 1
Block (0, 0)
Block (1, 0)
Block (0, 1)
Block (1, 1)
Block (0, 2)
Block (1, 2)

GPU and CPU Thread

- GPU threads are very lightweight and have very little creation overhead, but CPU thread creation overhead is very high.
- The implication is that GPU can use a huge number of threads to achieve high performance, but CPU can only use a relatively small number of threads.

Thread id

- Each thread has an id so that it will be able to determine which part of data it is supposed to handle.
  - The thread id is stored in a variable threadIdx
  - The contents of threadIdx depends on the dimensionality of the grid.

Thread Blocks

- A block is an N-dimensional array of threads.
  - N could be 1, 2, or 3.
- The coordinates of a thread within a block are stored within the variable threadId as a vector of three elements, as x, y, and z.
- When the host calls the kernel function to run on the devices, it can specify the thread block layout between the name of the kernel function and the parameter list.
  - initialize <<<1, 5>>> (int_array);

The ThreadIdx

- The kernel function can retrieve the thread index by referenceing threadIdx variable.
  - Int_array[threadIdx.x] = threadIdx.x;
  - Note that this is for a one-dimensional layout, so we use the x element only.

Global Memory

- The host and devices can share memory.
- This special global memory must be allocated using special allocation routine. Note that the prototypes of cudaMalloc is different from malloc.
  - cudaMalloc(&pointer, size);
    - The first argument is the address of the pointer.
  - cudaFree(pointer);

Host Memory

- The host can allocate and free its own memory using malloc and free.

CUDA Computation

- The host allocates and initializes it memory.
- The host allocates the global memory.
- The host copies the host memory into global memory.
- The hosts calls kernel function to performs computation on the global memory.
- The host copies the global memory back to the host memory.
- The host outputs the results.

Memory Transfer

- Host to device
  - cudaMemcpy(device_memory, host_memory, cudaMemcpyHostToDevice);
- Device to host
  - cudaMemcpy(host_memory, device_memory, size, cudaMemcpyDeviceToHost);
- Note that it is always from the second argument to the first argument.