Graphics Processing Units (GPUs)

Stéphane Zuckerman

(Slides include material from D. Orozco, J. Siegel, Professor X. Li, and the H&P book, 5th Ed.)

Computer Architecture and Parallel Systems Laboratories
http://www.capsl.udel.edu

Initial Purpose of GPUs

According to Wikipedia:

“A graphics processing unit (GPU), also occasionally called visual processing unit (VPU), is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the building of images in a frame buffer intended for output to a display.”

GPUs were initially made to process and output both 2D and 3D computer graphics. Two main areas require efficient 3D graphics processing: computer games, and real-time visualization for scientific processing.

Origins of GPUs

- As said before, GPUs were created to perform a specific task: real-time 3D rendering
  - Before GPUs: the CPU had to perform all this
  - Fun fact: MMX extensions (soon to be called SSE) appeared more or less at the same time
- Other extensions created over time include:
  - IEEE-754 floating-point co-processors (early 1980s)
  - Physics Processing Units (PPUs)
  - Etc.

General Processing Using GPUs

- Started in early 2000s
  - Diverted shader/vertex/etc. units within a GPU (e.g. instead of storing color information, use the 32-bit word to store an integer or floating-point value)
- While doable, programming was basically hell
  - Needed to divert OpenGL commands to do your computation
  - Not all computations can easily be mapped to triangles and/or polygons…
- Everything changed around 2006, with NVIDIA introducing GPUs which started to get more generic, including a C-based programming language (CUDA)

A DAXPY Example

```
/* Sequential code */
void daxpy(double *x, double *y, double a, size_t size){
      for (size_t i = 0; i < size; ++i)
            y[i] = a * x[i] + y[i];
}

/* OpenMP code */
void daxpy(double *x, double *y, double a, size_t size){
#pragma omp parallel default(none) shared(x,y,a,size)
{
      #pragma omp for
      for (size_t i = 0; i < size; ++i)
            y[i] = a * x[i] + y[i];
}
}
```

Taken from Hennessy & Patterson, *Computer Architecture, 5th Ed.*

A DAXPY Example

```
/* CUDA */
int main(void)
{
        double x = …, y = …;
        size_t size = …
        // Something is missing here... Can you see what?
        __host__
        int nblocks = (size + 255) / 256;
        daxpy_cuda<<<nblocks,256>>>(x,y,2.0,size);
}
__global__
void daxpy_cuda(double *x, double *y, double a, size_t
size)
{
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        if (i < n) y[i] = a * x[i] + y[i];
}
```

Taken from Hennessy & Patterson, *Computer Architecture, 5th Ed.*

Features of Current GPUs

- Boards have their own memory.
  - Usually between 1 and 2 GB
    - can go up to 6 GB on high-end models
  - Much faster than the processor's memory.
  - Higher bandwidth than the processor's memory.
- Massive Parallelism
  - GPUS support thousands of threads running at the same time.

Bandwidth

GeForce GTX 480

GPU Engine Specs:

CUDA Cores	480
Graphics Clock (MHz)	700 MHz
Processor Clock (MHz)	1401 MHz
Texture Fill Rate (billion/sec)	42

Memory Specs:

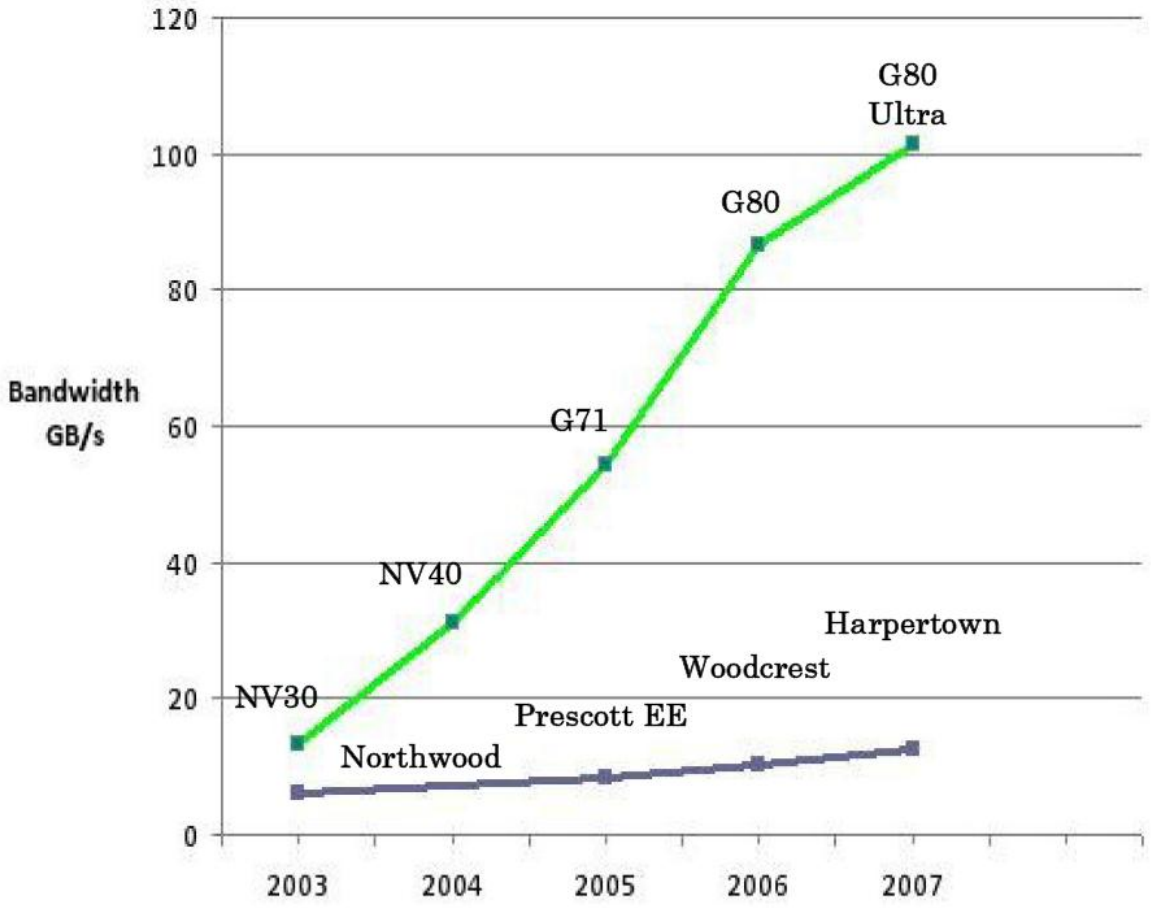
Memory Clock (MHz)	1848
Standard Memory Config	1536 MB GDDR5
Memory Interface Width	384-bit
Memory Bandwidth (GB/sec)	177.4

Intel's i7 Extreme Edition

Launch Date	Q1'10
Processor Number	i7-980X
# of Cores	6
# of Threads	12
Clock Speed	3.33 GHz
Max Turbo Frequency	3.6 GHz
Intel® Smart Cache	12 MB
Bus/Core Ratio	25

Courtesy of Daniel Orozco

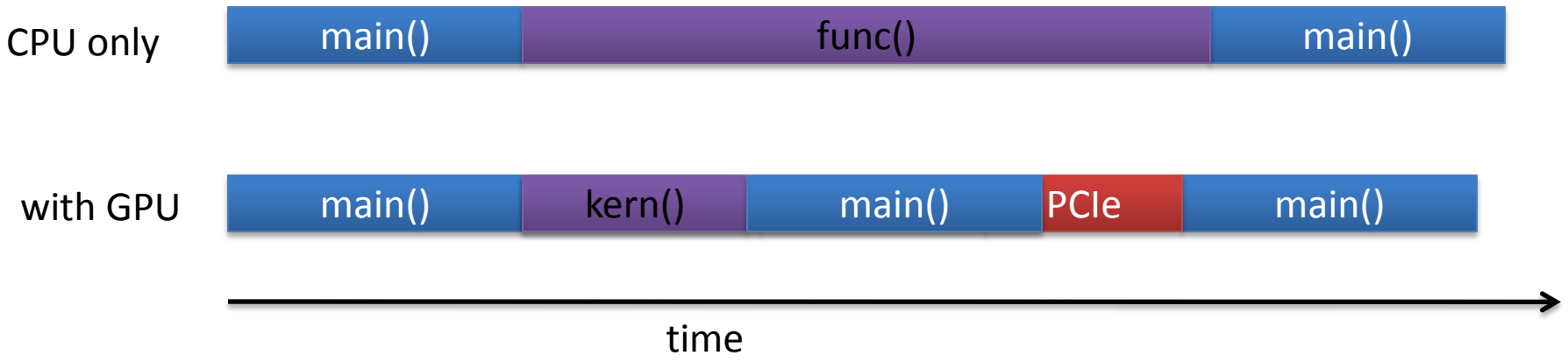
Bandwidth (cont'd)



Courtesy of Daniel Orozco (citing NVIDIA …)

Bandwidth (cont'd)

- Usage of a GPU only makes sense if it performs much faster than the CPU.
  - E.g. we have a CPU-function func() and a GPU-kernel kern() that perform the same task on a large data set.

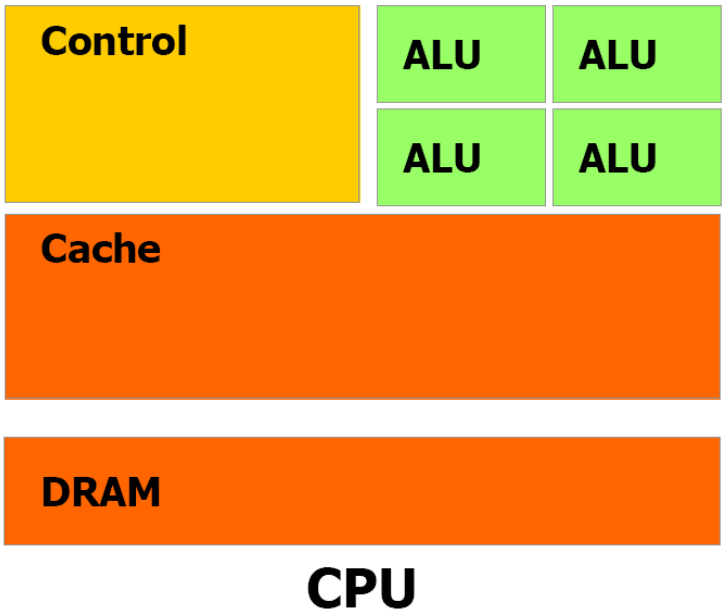


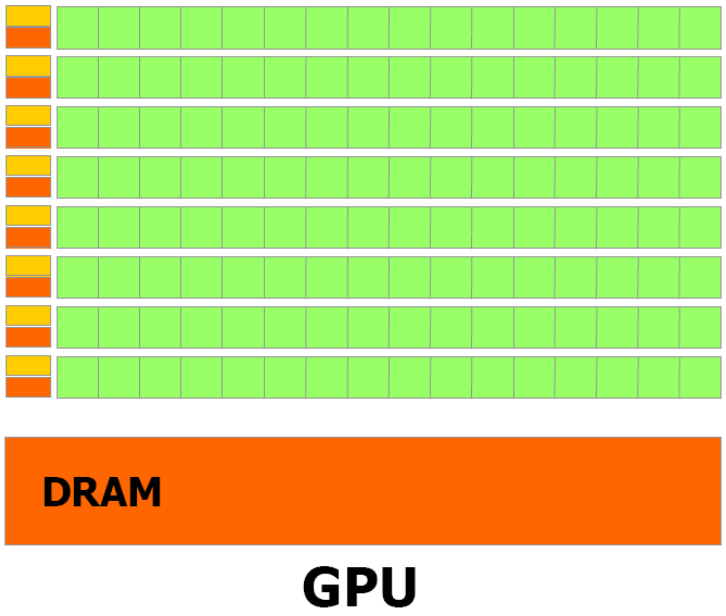
Courtesy of J. Siegel

Architecture

- In general, architectures of state-of-the-art GPUs are kept secret but some general details are published.
- For example NVIDIA publishes documentation about their processors and provides a full development toolchain to use their GPUs.

Architecture





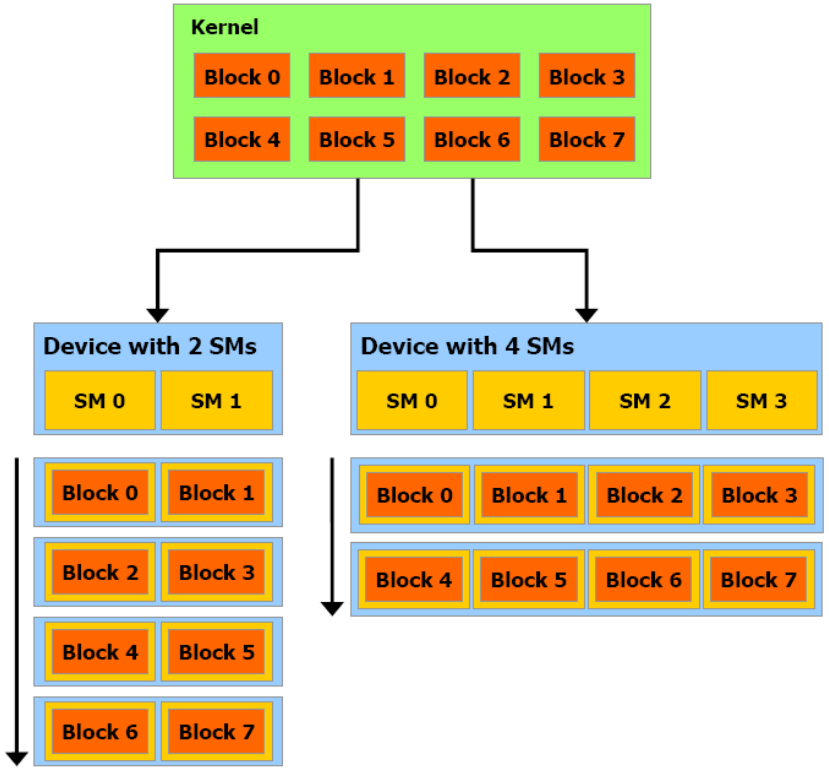
GPUs devote most of their transistors to computational units, and very few of them for control and cache.

GPUs are best suited for applications with simple control flow and high arithmetic intensity.

Courtesy of Daniel Orozco

Cuda Architecture Model

- Each CPU chip has several **multiprocessors.**
- Each multiprocessor executes exactly one block.
- Time sharing is possible when the number of blocks is bigger than the number of multiprocessors.



Courtesy of Daniel Orozco

NVIDIA GPUs: Terminology

Program abstractions

- Grid
  - a vectorizable loop
- Thread Block
  - A group of threads processing a portion of the loop
- (CUDA) Thread
  - Thread that processes one iteration of the loop

Machine Object

- Warp
  - A thread of SIMD instruction
- PTX instruction
  - SIMD instruction

Memory hardware

- Global Memory
  - DRAM available to all threads
- Local Memory
  - Private to the thread
- Shared Memory
  - Accessible to all threads of a Streaming Processor
- Thread Processor Registers

Processing hardware

- Streaming Multiprocessor
  - Multithreaded SIMD processor
- Giga Thread Engine
  - Thread block scheduler
- Warp Scheduler
  - SIMD Thread Scheduler
- Thread Processor
  - SIMD lane

Warps

The program on the right is slow because not all threads execute the same instructions.

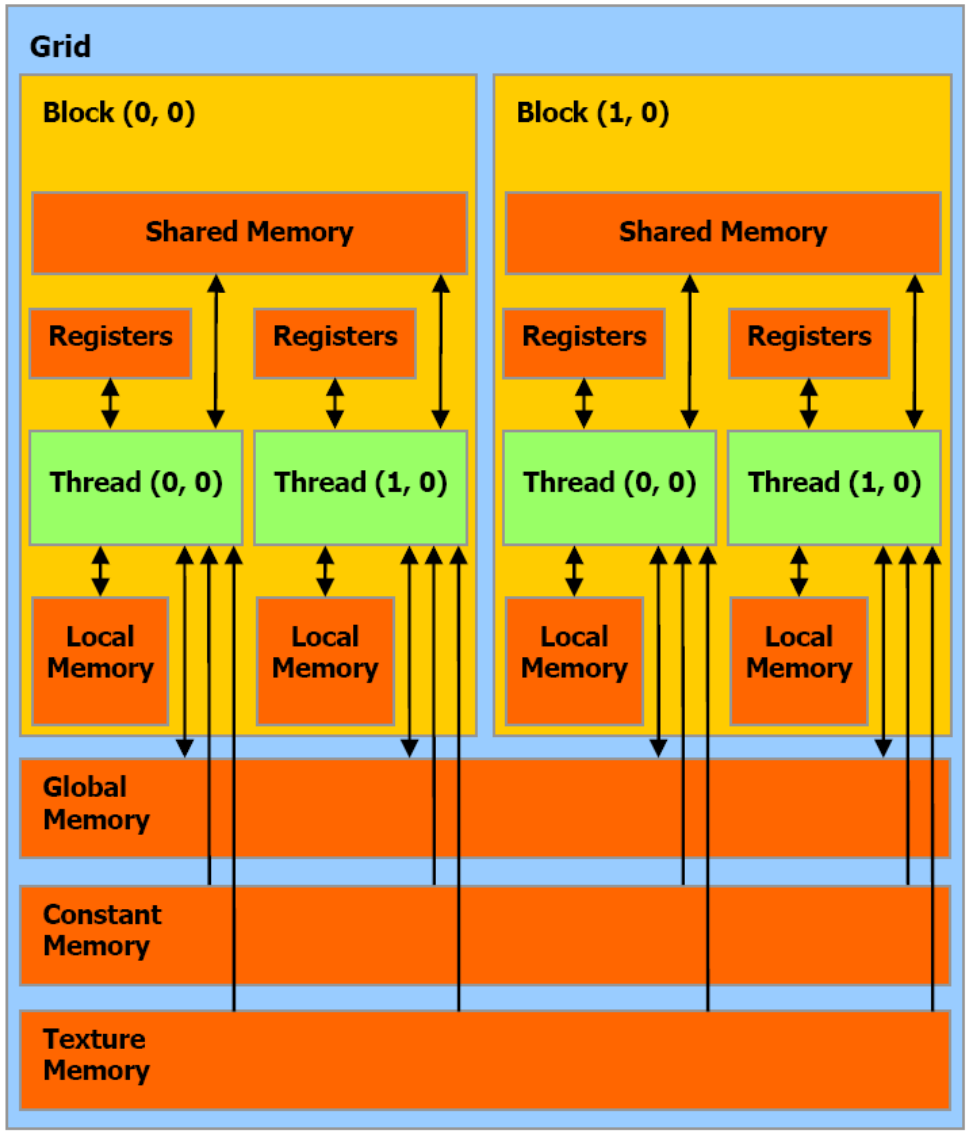
The total time for execution is:

$time(\ Compute\ 1\ )\ +\ time\ (\ Compute\ 2\ ).$

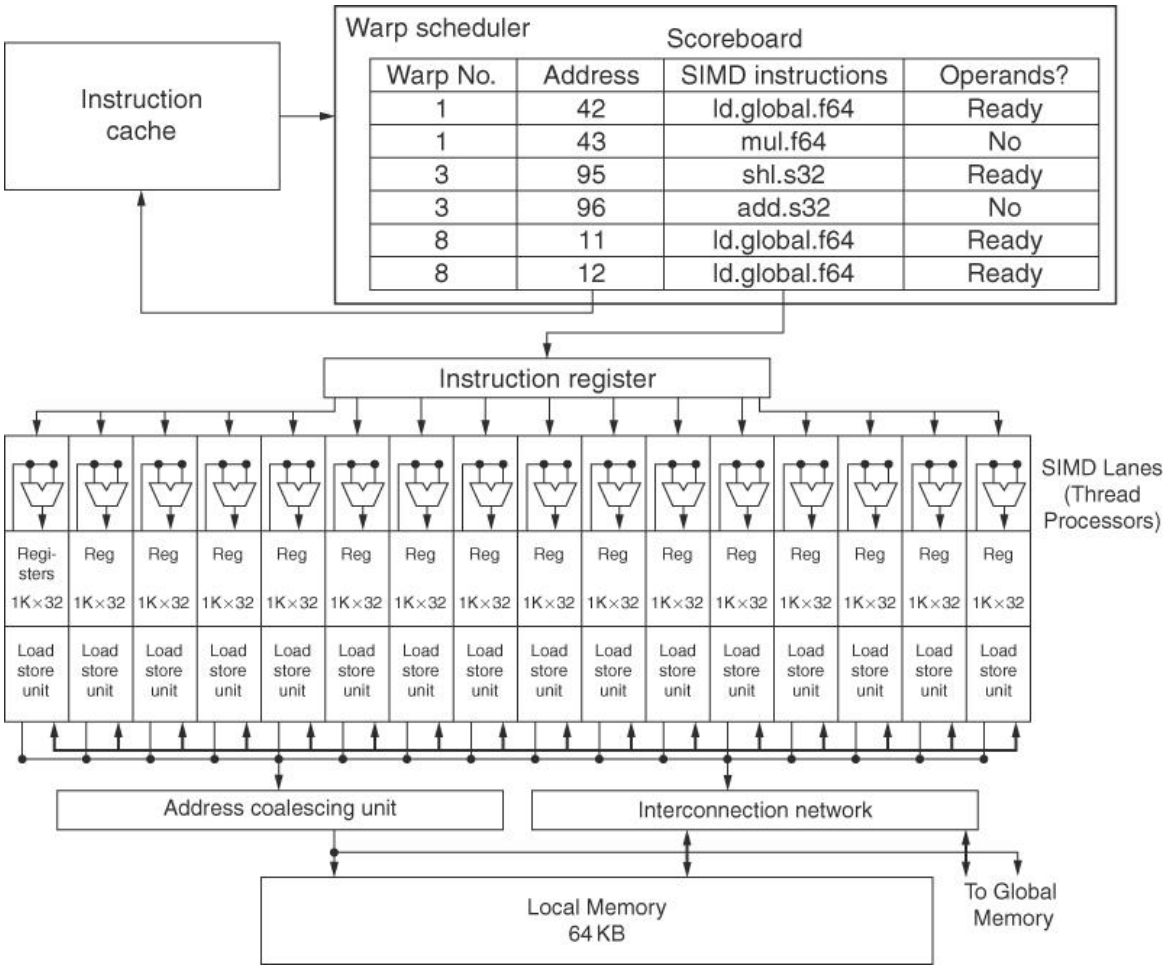
```
// All threads
if (my_thd_id < 16)
{
      // Compute 1
}
else
{
      // Compute 2
}
```

Block Diagram of an NVIDIA GPU

- Each thread has its own PC
- Thread schedulers use scoreboard to dispatch
- No data dependencies between threads
- Keeps track of up to 48 threads of SIMD instructions to hide memory latencies
- Thread block scheduler schedules blocks to SIMD processors
- Within each SIMD processor:
  - 32 SIMD lanes
  - Wide and shallow compared to vector processors



Block Diagram of an NVIDIA GPU (cont'd)



Simplified block diagram of a Multithreaded SIMD Processor. It has 16 SIMD lanes. The SIMD Thread Scheduler has, say, 48 independentthreads of SIMD instructions that it schedules with a table of 48 PCs.

Taken from Hennessy & Patterson, *Computer Architecture, 5th Ed.*

Example of NVIDIA GPU

- NVIDIA GPU has 32,768 registers
  - Divided into lanes
  - Each SIMD thread is limited to 64 registers
  - SIMD thread has up to:
    - 64 vector registers of 32 32-bit elements
    - 32 vector registers of 32 64-bit elements
  - Fermi has 16 physical SIMD lanes, each containing 2048 registers

Taken from Hennessy & Patterson, *Computer Architecture, 5th Ed.*

NVIDIA Instruction Set Architecture

- ISA is an abstraction of the hardware instruction set
  - "Parallel Thread Execution (PTX)"
  - Uses virtual registers
  - Translation to machine code is performed in software

- Example:

```
shl.s32 R8, blockIdx, 9     ; Thd Blk ID * Blk sz (512 or 29)
add.s32 R8, R8, threadIdx   ; R8 = i = my CUDA thread ID
ld.global.f64 RD0, [X+R8]   ; RD0 = X[i]
ld.global.f64 RD2, [Y+R8]   ; RD2 = Y[i]
mul.f64 R0D, RD0, RD4       ; RD0 = RD0 * RD4 (scalar a)
add.f64 R0D, RD0, RD2       ; Sum in RD0 = RD0 + RD2 (Y[i])
st.global.f64 [Y+R8], RD0   ; Y[i] = sum (X[i]*a + Y[i])
```

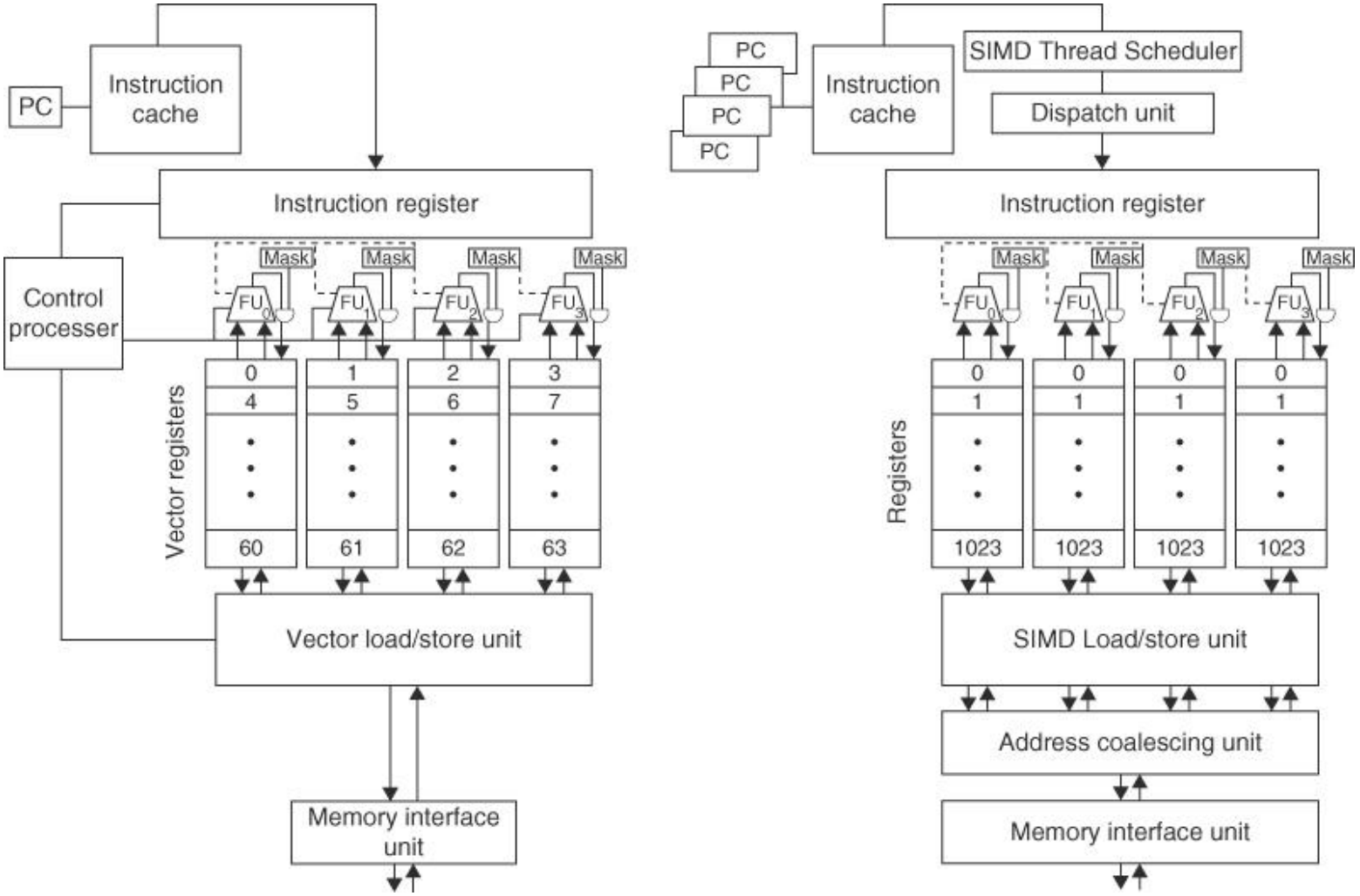
Taken from Hennessy & Patterson, *Computer Architecture, 5th Ed.*

Conditional Branching

- Like vector architectures, GPU branch hardware uses internal masks
- Also uses
  - Branch synchronization stack
    - Entries consist of masks for each SIMD lane
    - I.e. which threads commit their results (all threads execute)
  - Instruction markers to manage when a branch diverges into multiple execution paths
    - Push on divergent branch
  - …and when paths converge
    - Act as barriers
    - Pops stack
- Per-thread-lane 1-bit predicate register, specified by programmer

Taken from Hennessy & Patterson, *Computer Architecture, 5th Ed.*

Vector Processors vs. GPUs



A vector processor with four lanes on the left and a multithreaded SIMD Processor of a GPU with four SIMD Lanes on the right.

Taken from Hennessy & Patterson, *Computer Architecture, 5th Ed.*

Vector Processors vs. GPUs (cont'd)

- Similarities to vector machines:
  - Works well with data-level parallel problems
  - Scatter-gather transfers
  - Mask registers
  - Large register files

- Differences:
  - No scalar processor
  - Uses multithreading to hide memory latency
  - Has many functional units, as opposed to a few deeply pipelined units like a vector processor

What About Non-NVIDIA GPUs?

- GPUs have two main vendors: NVIDIA and AMD
- AMD's GPUs have a very different micro-architecture compared to NVIDIA's
- AMD focuses on SIM**D** (ILP) where NVIDIA focuses on SIM**T** (TLP)
- However, the comparison between vector processors and GPUs is still mostly valid

The Future of GPUs

- Computer history is cyclic:
  - Floating-point co-processors initially were off-chip, but finally got integrated at the die or even core level
  - Intel recently added AES instruction in its SSE ISA to help with cryptography
  - TCP/IP stack implemented on network interface cards
  - AES hardware implementation (often integrated on NICs)

- The convergence between CPU and GPU seems unavoidable:
  - Intel is already providing GPUs on recent micro-architectures (Sandy Bridge, Ivy Bridge) on the same die
  - Intel also provides many-core chips on a board
    - They don't like to call them "accelerators" because of the obvious link to GPUs, but really, they look A LOT like a GPU…
  - AMD proposed its so-called Accelerator Processing Unit (APU) with the Fusion micro-processor
    - Single address space for both GPUs and CPUs
    - MMU augmented with IOMMU for transfers between GPUs and CPUs
  - NVIDIA has approached ARM to see how to use "fat cores" to embed on their cards in order to have a more "control-friendly" set of CPUs embedded with the GPU

Warning: These are guestimations at best

References

- John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th Ed., Chapter 4
- NVIDIA white paper: NVIDIA's Next Generation CUDA Compute Architecture: FERMI, available at http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf
- CUDA tutorial at SC'09: http://www.nvidia.com/object/SC09_Tutorial.html
- Anand Tech review of some AMD GPUs: http://www.anandtech.com/show/4061/amds-radeon-hd-6970-radeon-hd-6950/5
- AMD's manual for the R600 GPU series: http://developer.amd.com/gpu_assets/r600isa.pdf
- An interesting document on GPGPUs: http://arkanis.de/weblog/2011-04-02-finished-my-practical-term/gpgpu-origins-and-gpu-hardware-architecture.pdf