GPUs: Doing More Than Just Games

Mark Gahagan

CSE 141

November 29, 2012

Outline

- Introduction: Why multicore at all?
- Background: What is a GPU?
- Quick Look: Warps and Threads (SIMD)
- NVIDIA Tesla: The First step
- NVIDIA Fermi: The mature GPGPU
- CUDA: The programming language behind it all

Introduction

- Up to this point you've seen single-core machines
  - MIPS can be multicore, but not worrying about that now
- In 2003, we realized we can't keep scaling the processor frequency because it would cost too much energy
- Stuck with two choices, change the architecture or multicore

Introduction

- Multicore won, and now it is everywhere
  - CPUs, GPUs, PS3, even cellphones these days
- Details on the challenges of CMPs are covered in lecture notes on the site
- Unfortunately, while CPUs provide good performance, they are once again reaching limits of power
- Today we will look at one such class of multiprocessor machines, GPUs.

Introduction

- GPUs are becoming an attractive option for exploiting highly parallel algorithms.
- While commodity multicore CPUs generally are limited to 4-8 cores per die, GPUs can have up to 500 “cores”.
  - Much faster as well
- NVIDIA is the biggest success story to date in exploiting this fact, and this talk will reflect this.

Introduction

- Multicore design and performance is a big research topic now, and some of us (including me) think current CPUs just aren’t good enough for future workloads.
- GPUs are used heavily in research and high-performance computing because of their ability to run highly parallel code
- Nowadays writing parallel code is becoming a must anyway, given how all CPUs are multicore

Background

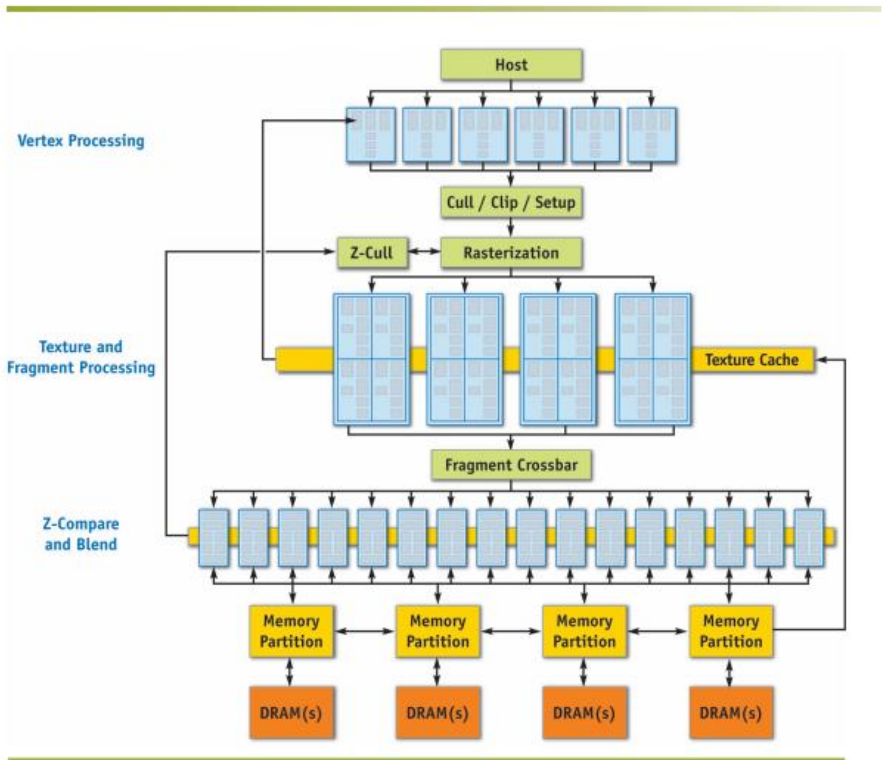
- GPUs had been overlooked for years as a potential source for computation
- Limited scope to actual graphics processing, and even then only really took off in the 1990s with the increase in 3-D video games.
- Prior to this, the focus of GPUs was limited, focusing on 2-D graphics acceleration.

Background

- NVIDIA released one of the first commercial GPUs, the GeForce 256
  - PCI or AGP 4x BUS
  - 32, 64 MB SDR RAM
    - 2.6 GB/s bandwidth
  - 120 MHz Core Clock
  - 220 nm Process
  - 4 Pixel Shaders (“cores”), 4 Texture mapping units, 4 Render Output Units

Background

- Geforce 6/7 series: Prior generation to Tesla
- This is a basic copy of the GPU architecture of the previous ~5 years
- Does show one change: The vertex processor, introduced at the top of the pipeline



Background

- Throughout this period GPUs were considered heterogeneous processors
  - They relied on different types of processors to do different things, but they did those things very quickly.
- This limited their abilities, but some saw the potential for GPUs to do more than just graphics processing.

Background

- So what needed to change?
  - The processors had to become more “general purpose”.
  - There needed to be enough of a desire to program in parallel relatively easily (you’ll see in the CMP lecture next week doing parallel programming is anything but easy).

Background

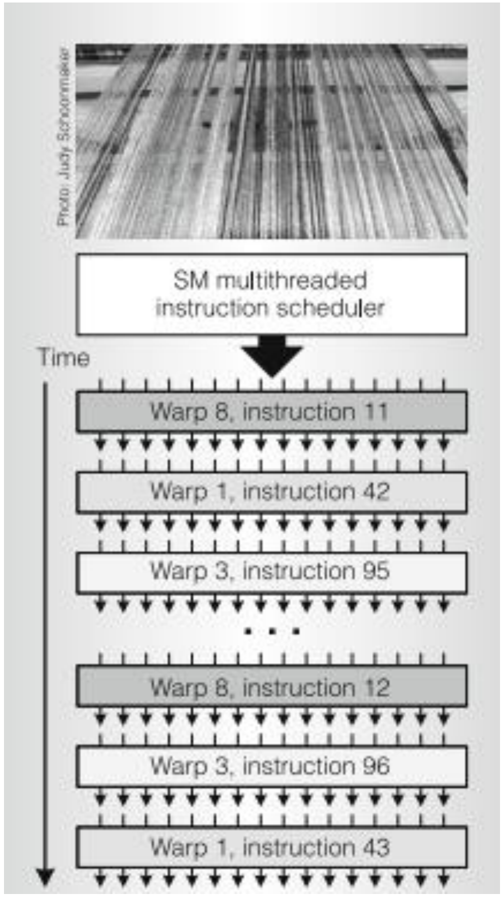
- FLOPs – Floating point operations per second
  - A metric used in computer to determine the actual processing power of a computer processor
  - Most high end Intel CPUs top out between 150-300 GFLOPs
  - NVIDIA cards, on the other hand, have the capability of going above 2000 GFLOPs, for given applications

Warps?

- SIMT – Single Instruction, Multiple Thread
  - The instruction unit manages threads in groups of 32 parallel threads, known as a warp
  - This wrap is used by the GPU to execute a set of instructions in one batch.
  - Difference between SIMT and SIMD is that multiple threads are executing in parallel, not just multiple data paths
    - But really, the term is more marketing than anything else, its just SIMD.

Warps?

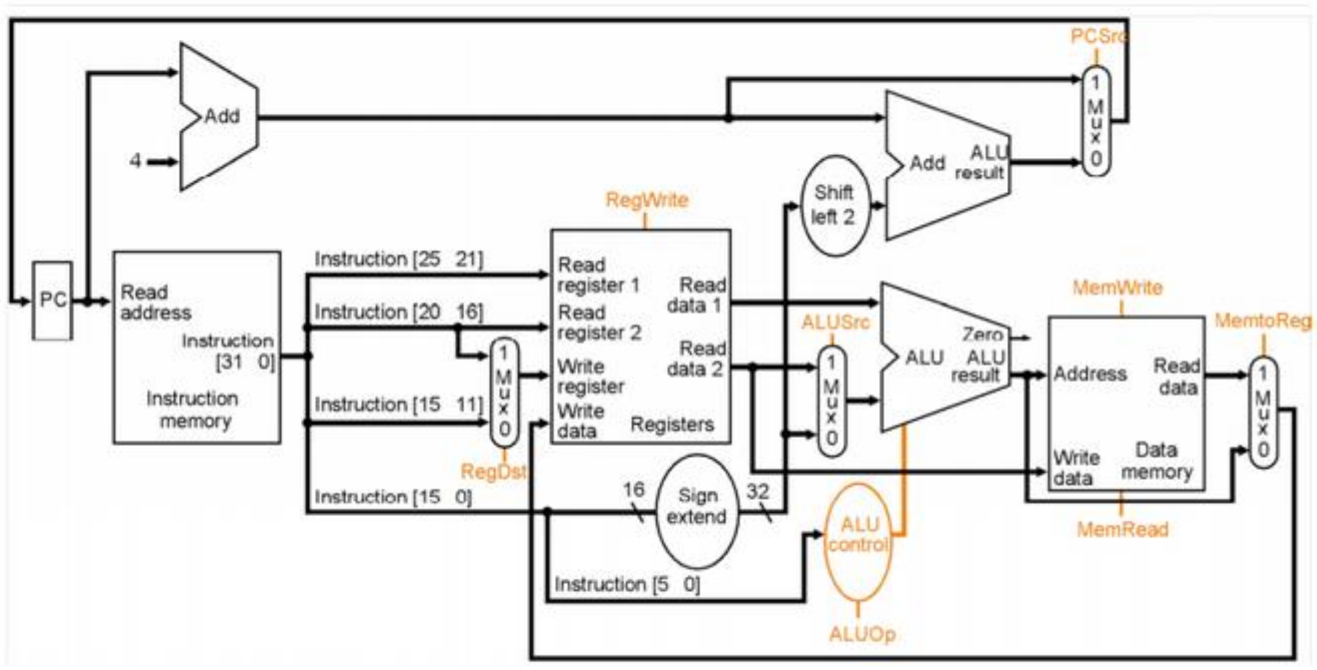
- What this all means
  - The GPU schedules instructions in warps to its various multiprocessors
  - These warps are then executed out of order. This means the processor needs to schedule properly so there are no hazards



NVIDIA Tesla - Introduction

- By the mid-2000s, work was being done at NVIDIA to try to better generalize the GPU.
  - The pixel shaders and vertex shaders were combined to a single unified architecture
    - This is because they did largely the same subset of calculations, they just focused on different things
  - Goal was to create a scalable processor array that not only would provide superior graphics capabilities, but useful parallel computing uses

NVIDIA Tesla - Introduction



NVIDIA Tesla - Architecture

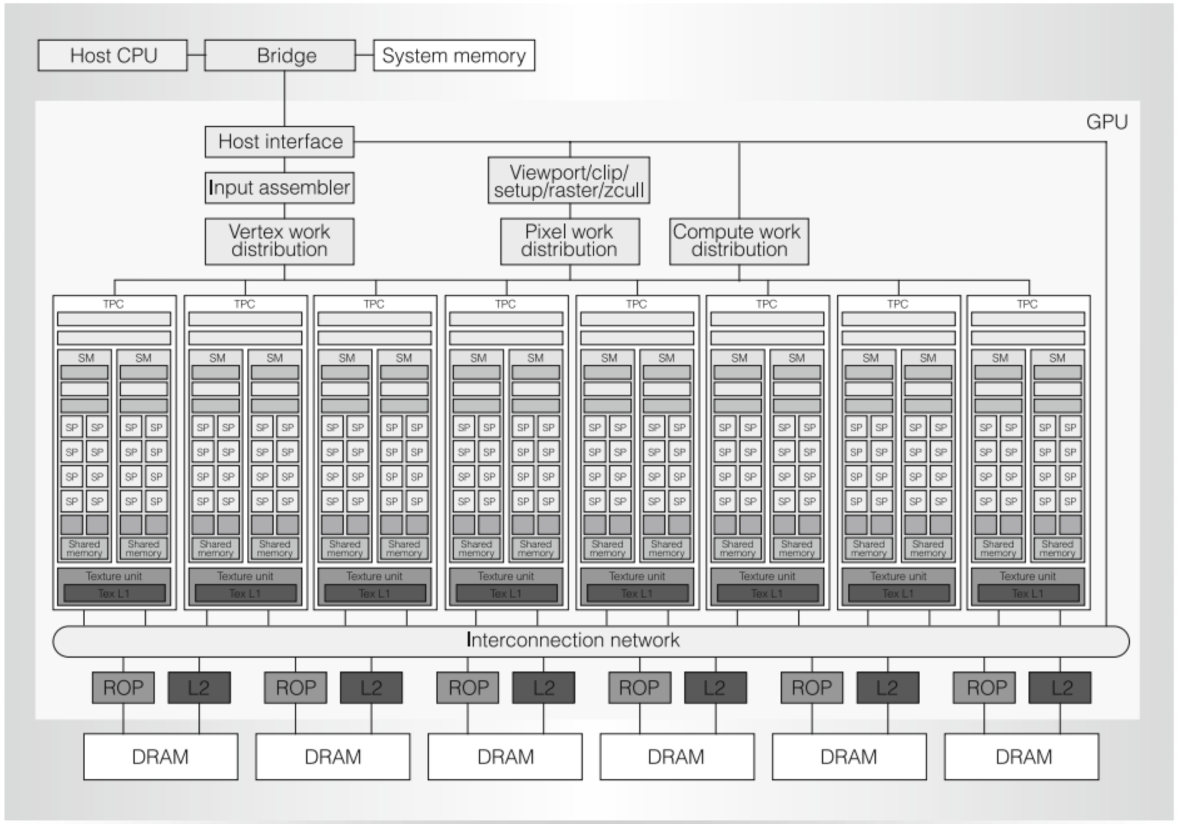
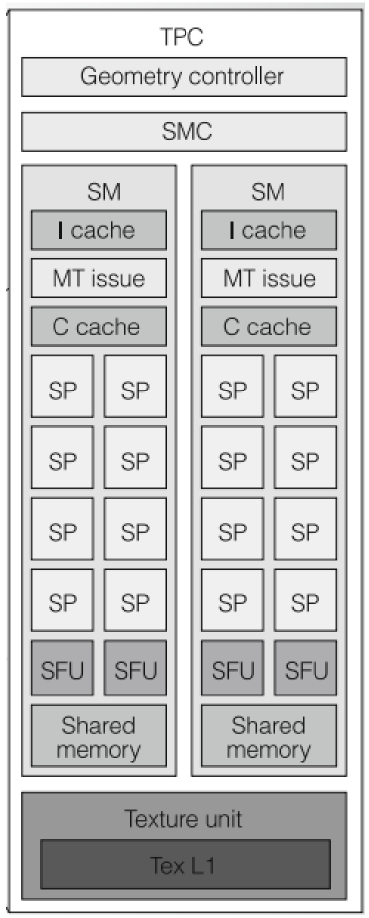


Figure 1. Tesla unified graphics and computing GPU architecture. TPC: texture/processor cluster; SM: streaming multiprocessor; SP: streaming processor; Tex: texture, ROP: raster operation processor.

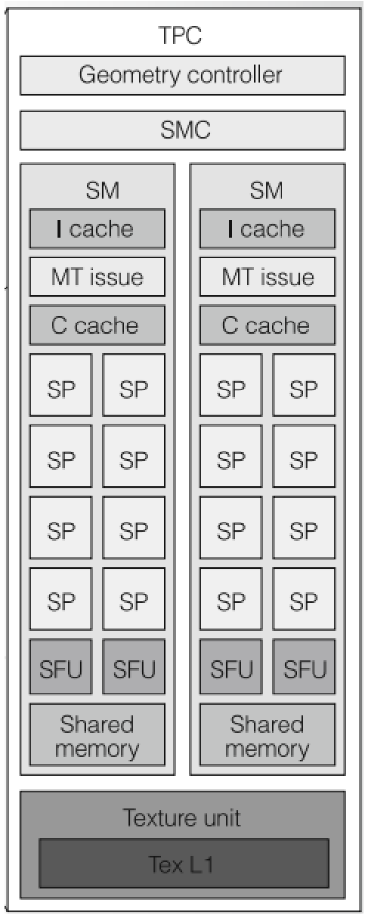
NVIDIA Tesla - Components

- There is a clear hierarchy for the Tesla cards, with the work being primarily done in the texture/processor clusters
- These clusters are scalable, can have just one, or can have eight or more.
- Idea was to have an easily expandable array of clusters



NVIDIA Tesla - Components

- Geometry Controller – handles vertex processing to the physical streaming multiprocessors
- Streaming Multiprocessor
  - Collection of streaming cores
  - Special Function Units
  - Multithreaded I-fetch and issue unit
  - Each Multiprocessor has shared memory for its cores
- Texture Unit – Processes one group of threads per cycle, optimized for texture computations



NVIDIA Tesla - Components

- Raster operations processor (ROP)
  - Processors like this have been around since the Geforce 256 era
  - Paired with a specific memory partition and texture/processor cluster
  - Supports an interconnect with both DDR2 and GDDR3 memory for up to 16 GB/s bandwidth
  - Processor is used to aid in antialiasing (in other words, still very much a solely graphics card property)

NVIDIA Tesla – Dataflow & Memory

- Warp capability
  - Each streaming multiprocessor handles 24 warps, or 768 threads
- Memory Access
  - Three types of memory spaces
    - Local memory for per-thread, private, temp data
    - Shared memory for low-latency access to data shared per SM
    - Global memory for data shared by all threads

NVIDIA Tesla - Dataflow & Memory

- Memory and Interconnect
  - Bus of 384 pins with 6 independent partitions
    - Means many possible connections
  - Use GDDR3 RAM, which has much higher bandwidth, though requires more power, than DDR DRAM
  - Interconnect network interfaces with the PCI-express bus and the front ends of the chip
  - Memory traffic within the chip goes through a specific component of the hardware that combines the various components together (the ROP)

Goals of the NVIDIA Tesla

- Extensive Data parallelism
  - Very SIMD, can execute the same instruction many times on different data points quickly
- Intensive floating point (FP) arithmetic
  - Have fast FP units to calculate difficult numbers
- Latency tolerant
  - Allow data to continuously be calculated without worrying about prior results
- Streaming Data flow
  - New data is constantly entering the GPU
- Modest inter-thread communication
  - Keep threads to doing their own thing.

Goals of the NVIDIA Tesla

- The architecture, coupled with the CUDA programming model, will aid in making this goal a reality.

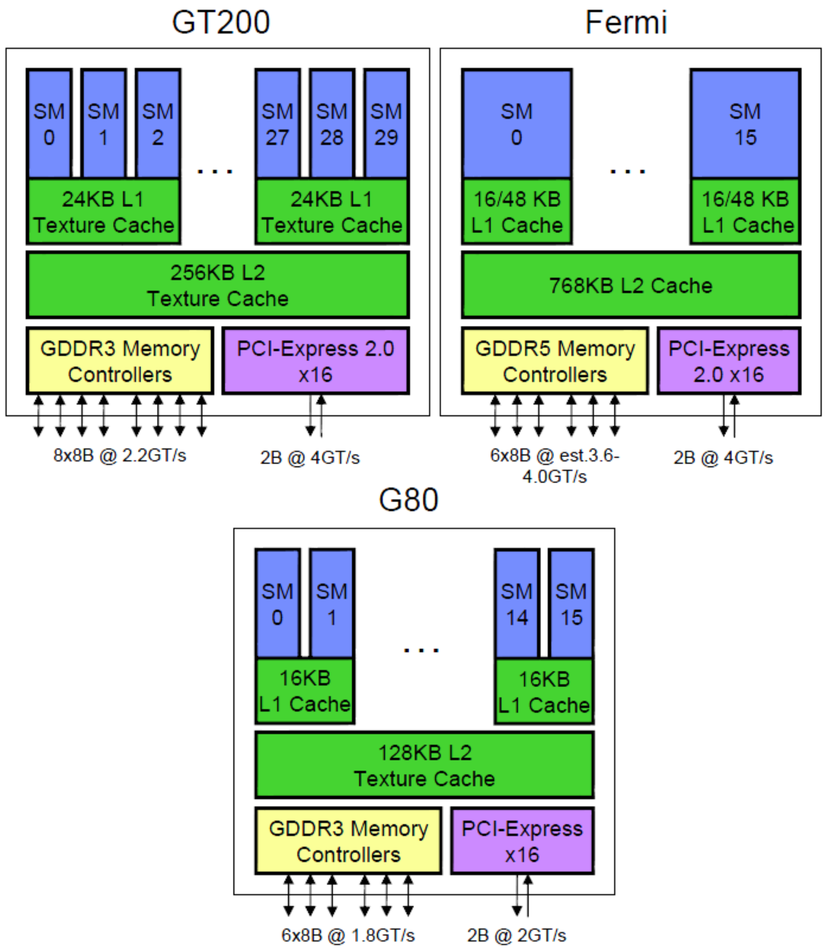
- We will discuss CUDA a little later on…first let’s look at a more modern GPU architecture
  - Why? Because we see GPUs start to look a little more like CPUs, and to see its transition toward the GPGPU (or General Purpose GPU)

NVIDIA Fermi – Introduction

- Released in 2010
- Wanted to address some key issues with their prior architectures:
  - Allows for a wider variety of work to be executed on a GPU, by providing constructs previously exclusive to CPUs
    - Exceptions
    - Multiple kernels
  - Clean up the mess that was their memory hierarchy, left over from years of prior GPU design

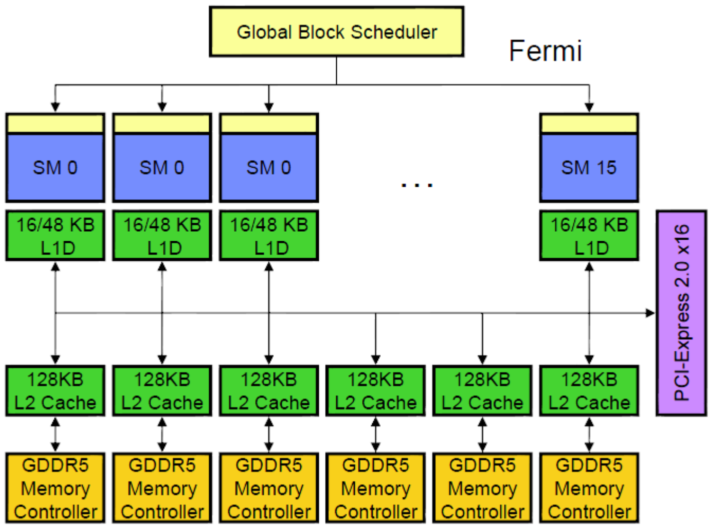
NVIDIA Fermi - Architecture

- Changes the memory hierarchy
  - Instead of 2 or 3 SMs sharing an L1 Texture cache, each SM has its own L1
- Memory improvement to GDDR5, capable of up to 144 GB/s
  - Means much faster memory bandwidth



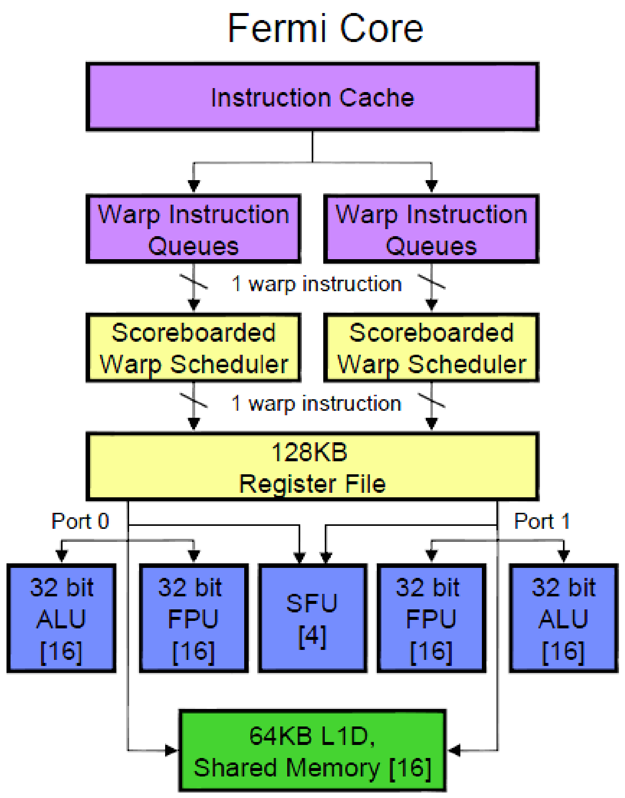
NVIDIA Fermi - Architecture

- On top of this, the Thread/processing cluster is eliminated, and each SM getting its own load store unit
- Increases number of threads for each Fermi core to 1536
  - Why does this matter?
    - Greater parallelism
    - Multiple, independent kernels



NVIDIA Fermi - Architecture

- Redesign of the SM unit
  - Each SM has two ports, which can access either the ALU or FPU but not both
  - Each port has 16 execution units, meaning a warp takes 2 fast cycles to complete
  - Warps are scoreboarded now, meaning structural hazards are tracked more carefully
    - Scoreboarding?
  - Inclusion of a "SFU", effectively the texture processor of the Fermi architecture



NVIDIA Fermi - Architecture

- L2 Cache
  - Shared global cache that acts like a early synchronization point (less work for programmer)
  - Write-back to global memory, most GPU-bound programs do not write over memory much
- ECC (Error Correction) Memory
  - Optional support in Fermi, unused before due to the performance hit it would cause
  - Some software designers like ECC, others think it's a waste
- Operand Collector
  - Hardware Units that read in many register values and buffer them for later consumption

Comparison

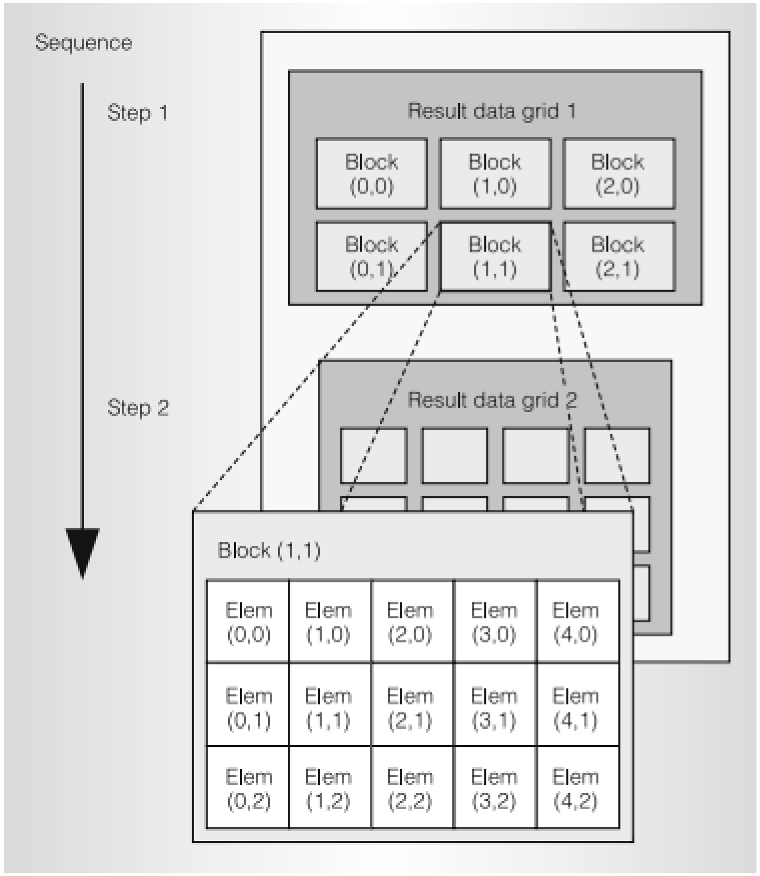
	GeForce 256 (DDR Version)	Tesla Arch. (GeForce 8800)	Fermi Arch. (Tesla c2050)
Fab Process	220 nm	90 nm	40 nm
Cores	4	128	448
Clock Rate	120 MHz	1.5 GHz	1.15 GHz
Peak GFLOPS	0.48 GF	576 GF	1030 GF
Memory Type	DDR	GDDR3	GDDR5
MBs of Memory	64	768	3000
Memory Clock	300 MHz	1.08 GHz	3 GHz
Bandwidth	4.8 GB/s	104 GB/s	144 GB/s
TDP	~20 W	150 W	225 W

CUDA - Introduction

- Developed by NVIDIA to help code for GPUs (specifically their GPUs)
- An extension of C and C++
- Programmer writes a serial program that calls parallel kernels
    - Can be as simple as a math equation
    - Or as complex as a full program

CUDA - Partitioning

- Generally the work is split into same-sized blocks
- Idea is to split the program into many equal parallel chunks
- Data is normally in some array format, so splitting these blocks is a relatively simple task



CUDA - Kernels

- CUDA is executed using kernels
  - Very C like code with a couple of exceptions
    - Elimination of first level for loops
    - Keep conditions and branches to a minimum
      - Why? Because branches mean that the code might make different decisions, leading to a concept called “thread divergence”

CUDA – Kernels

- Thread divergence
  - The concept that certain threads will take longer than others, forcing you to run as fast as the “slowest” runner.
  - You want to avoid this in parallel programming, as you get closer to single thread performance
  - Kind of the same thing as the “critical path” of a program

CUDA – Dealing With Memory

- Local Memory is at the Thread Level
- Shared Memory is at the Block Level
- Global Memory is at the Kernel Level

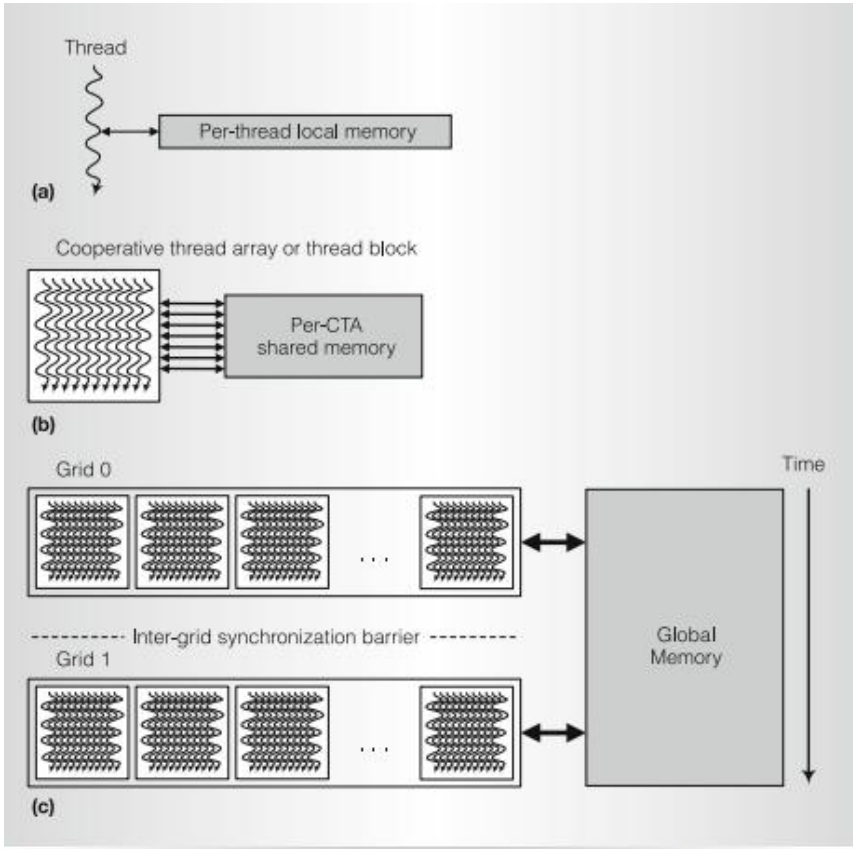


Figure 6. Nested granularity levels: thread (a), cooperative thread array (b), and grid (c). These have corresponding memory-sharing levels: local per-thread, shared per-CTA, and global per-application.

CUDA - Miscellaneous

- Synchronization ensures memory is safe, and functions are implied to explicitly require threads are synchronized before continuing
  - __syncthreads()
- Inter-kernel synchronization is implied, global synchronization occurs at kernel launch
- Recursive function calls are not allowed in CUDA (thread divergence occurs too easily, so you aren't running fib on a GPU)

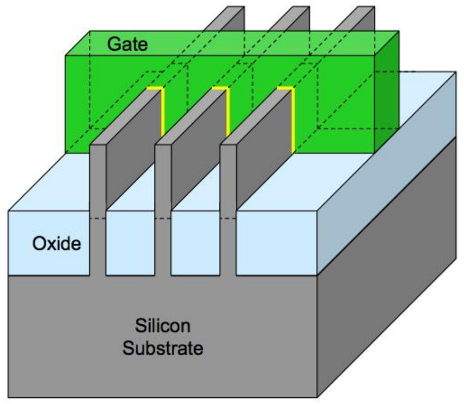
Related Work - GPUs

- AMD Fusion APU (2011-)
  - 1-4 core AMD CPU with Radeon Evergreen series GPU on the processor die
  - Designed to eliminate communication latency between CPU and GPU
  - In theory, the system is trying to obviate the need for discrete off-chip GPUs by providing an option that is lower power for the same GF/watt performance

Related Work - GPUs

- Intel Ivy Bridge (2012)
  - Intel's newest processor plans on having Intel HD Graphics on-chip, similar fashion to AMD
  - Will come coupled with a high-speed 2-8 core processor as well
  - Tri-gate transistors present on board, which allows for greater surface area and thus reduces leakage
  - Helps chip compete on power

22 nm Tri-Gate Transistor



Tri-Gate transistors can have multiple fins connected together to increase total drive strength for higher performance

Related Work – GPU Language Extensions

- OpenCL – Open Compute Language
  - Developed by Khronos
    - Industry Consortium that includes: AMD, ARM, Intel, and NVIDIA
  - Designed as an open standard for cross-platform parallel programming
  - Allows for more general programming across multiple GPUs/CPUs
  - Problems: Not as well developed/specialized as CUDA, companies are only just now adopting the standard

Related Work – GPU Language Extensions

	OpenCL	CUDA
Programming Language	C	C/C++
Supported GPUs	AMD, NVIDIA	NVIDIA
Supported CPUs	AMD, Intel, ARM	None
Method of Creating GPU Work	Kernel	Kernel
Run-time compilation of kernels	Yes	No
Multiple Kernel Execution	Yes (in certain hardware)	Yes (in certain hardware)
Execution Across Multiple Components	Yes	Yes – only GPUs
Need to Optimize for Best Performance	High	High
Coding Complexity	High	Medium

Conclusion

- NVIDIA GPUs are pushing slightly away from graphics processors and more toward a more general purpose high performance architecture
- While still competing in the pure GPU space, they have conceded ground to competitors in order to get the jump on HPC
- CUDA provides the framework to program these GPUs for general programming, and while other languages exist, CUDA is by far the most mature