Modern GPU Architectures

Varun Sampath
University of Pennsylvania
CIS 565 - Spring 2012

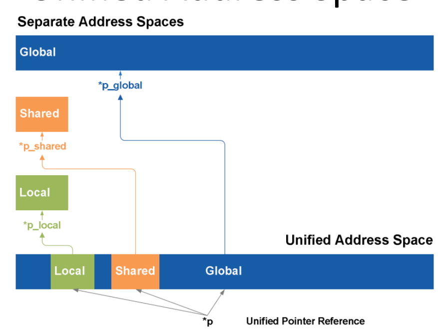
Agenda/GPU Decoder Ring

- Fermi / GF100 / GeForce GTX 480
  - "Fermi Refined" / GF110 / GeForce GTX 580
  - "Little Fermi" / GF104 / GeForce GTX 460
- Cypress / Evergreen / RV870 / Radeon HD 5870
  - Cayman / Northern Islands / Radeon HD 6970
- Tahiti / Southern Islands / GCN / Radeon HD 7970
- Kepler / GK104 / GeForce GTX 680
- Future
  - Project Denver
  - Heterogeneous System Architecture

From G80/GT200 to Fermi

- GPU Compute becomes a driver for innovation
  - Unified address space
  - Control flow advancements
  - Arithmetic performance
  - Atomics performance
  - Caching
  - ECC (is this seriously a graphics card?)
  - Concurrent kernel execution & fast context switching

Unified Address Space



- PTX 2.0 ISA supports 64-bit virtual addressing (40-bit in Fermi)
  - CUDA 4.0+: Address space shared with CPU
- Advantages?

Image from NVIDIA

Unified Address Space

```
cudaMemcpy(d_buf, h_buf, sizeof(h_buf),
           cudaMemcpyDefault)
```

- Runtime manages where buffers live
- Enables copies between different devices (not only GPUs) via DMA
  - Called GPUDirect
  - Useful for HPC clusters
- Pointers for global and shared memory are equivalent

Control Flow Advancements

- Predicated instructions
  - avoid branching stalls (no branch predictor)
- Indirect function calls:
  `call{.uni} fptr, flist;`
- What does this enable support for?

Control Flow Advancements

- Predicated instructions
  - avoid branching stalls (no branch predictor)
- Indirect function calls:
  `call{.uni} fptr, flist;`
- What does this enable support for?
  - Function pointers
  - Virtual functions
  - Exception handling
- Fermi gains support for recursion

Arithmetic

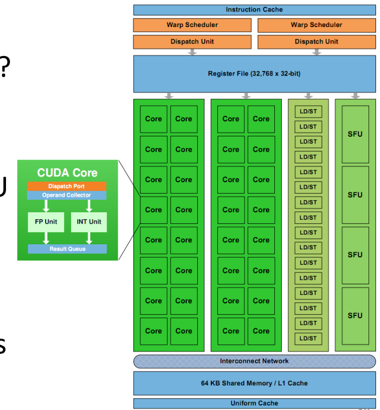
- Improved support for IEEE 754-2008 floating point standards
- Double precision performance at half-speed
- Native 32-bit integer arithmetic
- Does any of this help for graphics code?

Cache Hierarchy

- 64KB L1 cache per SM
  - Split into 16KB and 48KB pieces
  - Developer chooses whether shared memory or cache gets larger space
- 768KB L2 cache per GPU
  - Makes atomics really fast. Why?
  - 128B cache line
  - Loosens memory coalescing constraints

The Fermi SM

- Dual warp schedulers – why?
- Two banks of 16 CUDA cores, 16 LD/ST units, 4 SFU units
- A warp can now complete as quickly as 2 cycles



Fermi Streaming Multiprocessor (SM)

Image from NVIDIA

The Stats

GPU	G80	GT200	GF100
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Special Function Units / SM	2	2	4
Warp schedulers (per SM)	1	1	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

Image from Stanford CS193g

The Review in March 2010

- Compute performance unbelievable, gaming performance on the other hand…
  - "The GTX 480… it's hotter, it's noisier, and it's more power hungry, all for 10-15% more performance." – AnandTech (article titled "6 months late, was it worth the wait?")
- Massive 550mm^2 die
  - Only 14/16 SMs could be enabled (480/512 cores)

“Fermi Refined” – GTX 580

- All 32-core SMs enabled
- Clocks ~10% higher
- Transistor mix enables lower power consumption

“Little Fermi” – GTX 460

- Smaller memory bus (256-bit vs 384-bit)
- Much lower transistor count (1.95B)
- Superscalar execution: one scheduler dual-issues
- Reduce overhead per core?

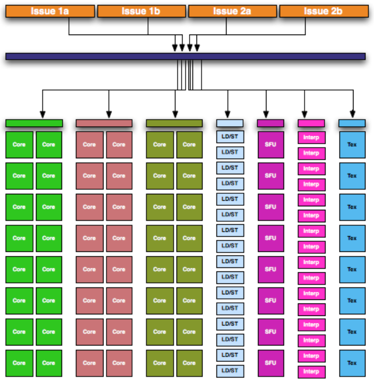


Image from AnandTech

A 2010 Comparison

NVIDIA GeForce GTX 480

- 480 cores
- **177.4 GB/s** memory bandwidth
- 1.34 TFLOPS single precision
- **3 billion** transistors

ATI Radeon HD 5870

- **1600** cores
- 153.6 GB/s memory bandwidth
- **2.72 TFLOPS** single precision
- 2.15 billion transistors

Over double the FLOPS for less transistors! What is going on here?

VLIW Architecture

- Very-Long-Instruction-Word
- Each instruction clause contains up to 5 instructions for the ALUs to execute in parallel

\+ Save on scheduling and interconnect (clause “packing” done by compiler)

\- Utilization

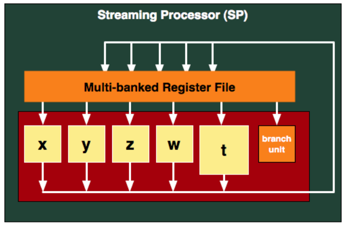
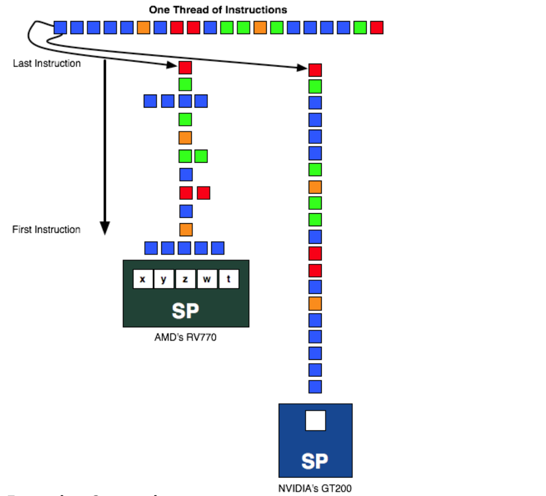


Image from AnandTech



Execution Comparison

Image from AnandTech

Assembly Example

AMD VLIW IL

```
0  t: MULLO_INT   ____,
1  z: ADD_INT     ____,
   t: MULLO_INT   ____,
2  x: ADD_INT     R0.x,
   w: ADD_INT     ____,
3  y: ADD_INT     R0.y,
   z: SETGT_UINT  T0.z,
4  y: SETGT_UINT  ____,
5  x: AND_INT     ____,
6  w: AND_INT     R0.w,
```

NVIDIA PTX

```
mov.u32    %r1, %ctaid.x;
mov.u32    %r2, %ntid.x;
mul.lo.u32  %r3, %r1, %r2;
mov.u32    %r4, %ctaid.y;
mov.u32    %r5, %ntid.y;
mul.lo.u32  %r6, %r4, %r5;
mov.u32    %r7, %tid.x;
add.u32    %r8, %r7, %r3;
mov.u32    %r9, %tid.y;
add.u32    %r10, %r9, %r6;
```

The Rest of Cypress

- 16 Streaming Processors packed into SIMD Core/ compute unit (CU)
  - Execute a “wavefront” (a 64-thread warp) over 4 cycles
- 20 CUs * 16 SPs * 5 ALUs = 1600 ALUs

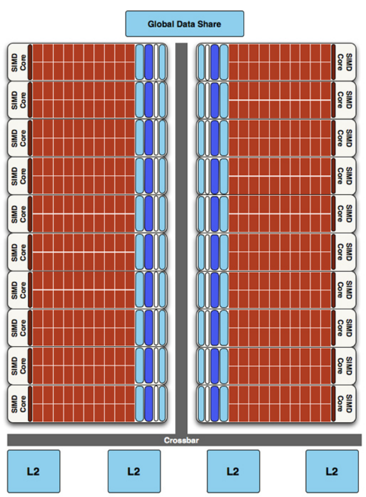
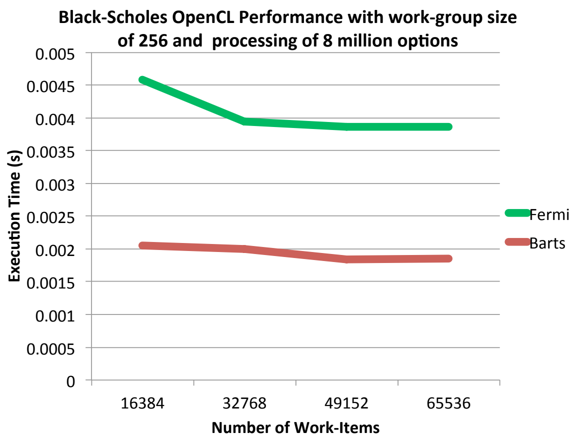
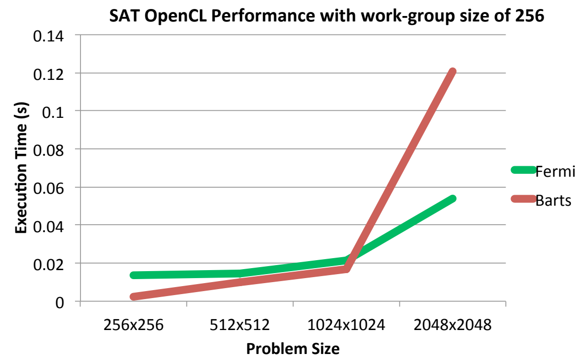


Image from AnandTech

Performance Notes

- VLIW architecture relies on instruction-level parallelism
- Excels at heavy floating-point workloads with low register dependencies
  - Shaders?
- Memory hierarchy not as aggressive as Fermi
  - Read-only texture caches
  - Can’t feed all of the ALUs in an SP in parallel
    - Fewer registers per ALU
    - Lower LDS capacity and bandwidth per ALU





Optimizing for AMD Architectures

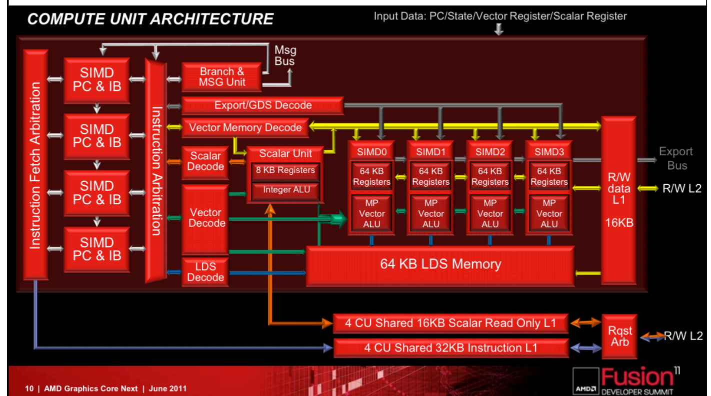
- Many of the ideas are the same – the constants & names just change
  - Staggered Offsets (Partition camping)
  - Local Data Share (shared memory) bank conflicts
  - Memory coalescing
  - Mapped and pinned memory
  - NDRange (grid) and work-group (block)sizing
  - Loop Unrolling
- Big change: be aware of VLIW utilization
- Consult the OpenCL Programming Guide

AMD's Cayman Architecture – A Shift

- AMD found average VLIW utilization in games was 3.4/5
- Shift to VLIW4 architecture
- Increased SIMD core count at expense of VLIW width
- Found in Radeon HD 6970 and 6950

Paradigm Shift – Graphics Core Next

- Switch to SIMD-based instruction set architecture (no VLIW)
  - 16-wide SIMD units executing a wavefront
  - 4 SIMD units + 1 scalar unit per compute unit
  - Hardware scheduling
- Memory hierarchy improvements
  - Read/write L1 & L2 caches, larger LDS
- Programming goodies
  - Unified address space, exceptions, functions, recursion, fast context switching
- Sound Familiar?



Radeon HD 7970: 32 CUs * 4 SIMDs/CU * 16 ALUs/SIMD = 2048 ALUs

Image from AMD

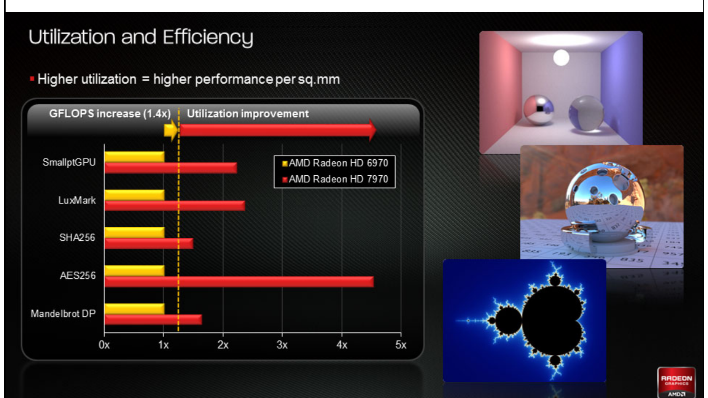


Image from AMD

NVIDIA's Kepler

NVIDIA GeForce GTX 680

- 1536 SPs
- 28nm process
- 192.2 GB/s memory bandwidth
- 195W TDP
- 1/24 double performance
- 3.5 billion transistors

NVIDIA GeForce GTX 580

- 512 SPs
- 40nm process
- 192.4 GB/s memory bandwidth
- 244W TDP
- 1/8 double performance
- 3 billion transistors

- A focus on efficiency
  - Transistor scaling not enough to account for massive core count increase or power consumption
  - Kepler die size 56% of GF110's

Kepler's Extended SM (SMX)

- Removal of shader clock means warp executes in 1 GPU clock cycle
  - Need 32 SPs per clock
- Kepler then doubles resources again of GF104 SM
  - 96 SPs, 32 LD/ST, 32 SFUs
  - New FP64 block
- Note: from Fermi, register file size has only doubled and shared memory/L1 size is the same

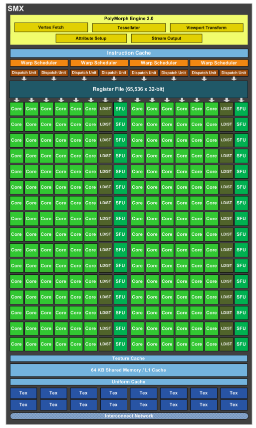
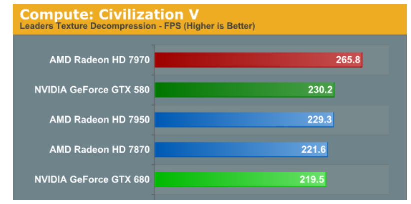
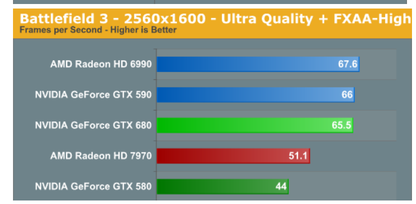


Image from AnandTech

Performance

- GTX 680 may be compute regression but gaming leap
- "Big Kepler" expected to remedy gap
  - Double performance necessary





Images from AnandTech

Future: Integration

- Hardware benefits from merging CPU & GPU
  - Mobile (smartphone / laptop): lower energy consumption, area consumption
  - Desktop / HPC: higher density, interconnect bandwidth
- Software benefits
  - Mapped pointers, unified addressing, consistency rules, programming languages point toward GPU as vector co-processor

The Contenders

- AMD – Heterogeneous System Architecture
  - Virtual ISA, make use of CPU or GPU transparent
  - Enabled by Fusion APUs, blending x86 and AMD GPUs
- NVIDIA – Project Denver
  - Desktop-class ARM processor effort
  - Target server/HPC market?
- Intel
  - ~~Larrabee~~ Intel MIC

References

- NVIDIA Fermi Compute Architecture Whitepaper. Link
- Schroeder, Tim C. "Peer-to-Peer & Unified Virtual Addressing" CUDA Webinar. Slides
- AnandTech Review of the GTX 460. Link
- AnandTech Review of the Radeon HD 5870. Link
- AnandTech Review of the GTX 680. Link
- AMD OpenCL Programming Guide (v 1.3f). Link

Bibliography

- Beyond3D's Fermi GPU and Architecture Analysis. Link
- RWT's article on Fermi. Link
- AMD Financial Analyst Day Presentations. Link