Design Tradeoffs in the Kepler Architecture

June 26, 2012
Steve Molnar
Kepler GK110 Block Diagram

Architecture
- 7.1B Transistors
- 15 SMX units
- > 1 TFLOP FP64
- 1.5 MB L2 Cache
- 384-bit GDDR5
- PCI Express Gen3
The Kepler GPU architecture was designed for

- Performance
- Programmability
- Efficiency
The Kepler architecture family had many goals, but the key goal was *efficiency* (*perf/watt*)

- On Fermi we designed for max performance, but found ourselves power-limited in many cases:
  - Tesla high-performance compute parts were power limited and had to run at lower voltage and clocks than the design allowed
  - Dual-GPU systems had to run at lower clocks
  - Mobile parts required lower-than-desirable voltage and clocks to maintain battery life
On Kepler power was critical

- Below ~0.1µm, CMOS power no longer scales with feature size
  - Moving to a smaller process, a chip of given size burns more power than the previous one
  - You can lower voltage (and clock speed), but that’s no solution
- In Kepler’s 28nm process, we could be power limited by 50%

*Time for decisive action!*
Kepler took holistic view of power

- We had previously designed to reduce power, but on Kepler, we attacked power holistically
  - Tools to measure power consumed by each unit
  - Aggressive clock and power-gating
  - Redesigned shader core to greatly increase efficiency
  - Redesign of GDDR5 DRAM I/O for speed and power
  - Architectural enhancements to reduce work

- In the remainder of this talk I will discuss several of these – and some other system-level design tradeoffs
Fermi SM compared to Kepler SMX

- 3x the math units
- Greatly increased efficiency
SM redesigned with power efficiency in mind

- 2x hardware at $\frac{1}{2}$ clock frequency
  - Reduces power consumption
  - 40nm to 28 nm provides more area
- Overall result
  - SMX Performance is up
  - SMX Power is down
  - Perf/watt metric benefits from both
Optimizing for area vs. optimizing for power

<table>
<thead>
<tr>
<th></th>
<th>Logic</th>
<th>Clocking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>Power</td>
</tr>
<tr>
<td>Fermi 2x clock</td>
<td>1.0x</td>
<td>1.0x</td>
</tr>
<tr>
<td>Kepler 1x clock</td>
<td>1.8x</td>
<td>0.9x</td>
</tr>
</tbody>
</table>
Scheduling complexity moved from hardware to compiler

Fermi
HW Dependency Check

Inst. Cache → Decoder → Multi-port Post-decode Queue → Multi-port Register Scoreboard → Dependency Check → Reorder → Issue

Kepler
SW Pre-decode

Inst. Cache → Sched. info → Select → Decoder → Issue
New GDDR5 DRAM controller

- Clean slate design to:
  - Achieve peak GDDR5 speed
  - Minimize power

- World’s first 6Gbps GDDR5
Texture improvements

- SMX vs Fermi SM:
  - 4x filter ops per clock
  - 4x cache capacity

- In most texture-heavy regimes, shader is not limited by texture
Groundbreaking Power Efficiency

![Bar chart comparing Perf / Watt of GTX 580 and GTX 680 across various games]

All games run at 19x10. See Appendix for detail settings.
Other Kepler improvements
Bindless Textures

- Dramatic increase in the number of unique textures available to shaders at run-time
- More different materials and richer texture detail in a scene

Pre-Kepler texture binding model
128 simultaneous textures

Kepler bindless textures
over 1M simultaneous textures
Atomic instruction enhancements

- Shorter processing pipeline
- More atomic processors
- Slowest 10x faster
- Fastest 2x faster
- Added int64 functions to match existing int32

2x-10x performance increase
High speed atomics enable new uses

Atomics are now fast enough to use within inner loops

Example: Data reduction (sum of all values)

Without Atomics

1. Divide input data array into N sections
2. Launch N blocks, each reduces one section
3. Output is N values
4. Second launch of N threads, reduces outputs to single value
High speed atomics enable new uses

Atomics are now fast enough to use within inner loops

- Example: Data reduction (sum of all values)

With Atomics

1. Divide input data array into N sections
2. Launch N blocks, each reduces one section
3. Write output directly via atomic. No need for second kernel launch.
GPU virtualization enables cloud gaming

- Kepler host and memory virtualization allow multiple virtual GPUs to be hosted on a single physical GPU
- Other critical pieces:
  - Fast hardware encoder works directly from render target
  - Cloud servers
  - Fast, low-latency WAN
- Used by Gaikai (www.gaikai.com)

GeForce VGX Cloud GPU
Game Latency
in Milliseconds

<table>
<thead>
<tr>
<th>Source</th>
<th>Game pipeline</th>
<th>Capture/Encode</th>
<th>Network</th>
<th>Decode</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaikai powered by GeForce Grid</td>
<td>50</td>
<td>10</td>
<td>30</td>
<td>5</td>
<td>66</td>
</tr>
<tr>
<td>Cloud Gen I</td>
<td>100</td>
<td>30</td>
<td>75</td>
<td>15</td>
<td>66</td>
</tr>
<tr>
<td>Console + TV</td>
<td>100</td>
<td>66</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The fastest, most powerful, lowest-latency cloud network on the market

Gaikai offers a bleeding-edge open platform technology that allows the most demanding games and applications to be seamlessly streamed via any connected video capable device including PCs, digital TVs, tablets and smart mobile devices. Having the fastest, lowest-latency, most sophisticated cloud network in the world enables users to experience rich media content as if they were running it locally.

Contact Us
Dynamic Parallelism (GK110+)
What is Dynamic Parallelism?

The ability to launch new work from the GPU

- Dynamically
- Simultaneously
- Independently

Fermi: Only CPU can generate GPU work

Kepler: GPU can generate work for itself
What Does It Mean?

GPU as Co-Processor

Autonomous, Dynamic Parallelism
Dynamic Work Generation

Initial Grid

Statically assign conservative worst-case grid

Dynamically assign resources where accuracy is required

Fixed Grid

Dynamic Grid
Batched & Nested Parallelism

CPU-Controlled Work Batching

- CPU programs limited by single point of control
- Can run at most 10s of threads
- CPU is fully consumed with controlling launches

Algorithm flow simplified for illustrative purposes

Multiple LU-Decomposition, Pre-Kepler
Batched & Nested Parallelism

**Batching via Dynamic Parallelism**
- Move top-level loops to GPU
- Run thousands of independent tasks
- Release CPU for other work

Algorithm flow simplified for illustrative purposes
Supporting an Architecture Family

gk107
gk104
gk110
Kepler isn’t a chip – it’s an architecture family

- Same base architecture must scale over wide range, diverse markets
  - Mobile graphics
  - Consumer desktop and enthusiast graphics
  - Workstation / professional graphics
  - High-performance computing (gk110)
Scaling and feature parameters

- Major configuration parameters (there are many more)
  - Note some are non-power-of-2

<table>
<thead>
<tr>
<th>Chip</th>
<th>GPCs</th>
<th>SMX per GPC</th>
<th>FBs</th>
<th>L2 size</th>
<th>ECC</th>
<th>Fast FP64</th>
</tr>
</thead>
<tbody>
<tr>
<td>gk104</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>512K</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>gk107</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>256K</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>gk110</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>1536K</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Market requirements determine configuration
  - Resource balance not the same at each level
  - Can change during project development
  - Arch model is quick; a lot of work remains for physical + pad design
Summary: World’s fastest and most efficient GPUs

- Performance
- Programmability
- Efficiency
Lots more

- GPU Boost
- TXAA
- Adaptive Vsync
- New shader instructions
For more information:

- **Kepler whitepaper:**

- **GeForce GRID (cloud gaming):**

- **GPU Technology Conference presentations:**
  - [www.gputechconf.com](http://www.gputechconf.com)

Acknowledgments: thanks to Lars Nyland, James Wang, and Jonah Alben for many of the slides used here and to the entire Kepler team.