Challenges Using Heterogeneous Systems

Jeremy Meredith

Oak Ridge National Laboratory

March 9, 2010

A Historical GPU Perspective

Challenges (ca. 2005)

- Moving targets
  - Hardware capabilities (e.g. branching)
  - Language features (e.g. instruction limits)
  - Drivers are incomplete and unstable
  - Vendor-specific limitations (num texture fetches/program, FBOs)
  - Variation across operating systems (RenderTexture)
- Precision limitations
  - 32-bit floats in best case
  - Not full IEEE rounding

Challenges (ca. 2005)

- Memory limitations
  - 512MB is an upper limit
  - Array size limited to 4096×4096
- Bandwidth to/from the card
  - AGP common, PCIe typically a fraction of its 4GB/s max
  - Asymmetric – readback can be many times worse
- Capabilities change almost unpredictably
  - Example: alpha blending on NV40 series
    - Works with 8-bit buffers
    - Software emulated with 16-bit float buffers
    - Fails silently with 32-bit float buffers

Challenges (ca. 2005)

- No debugger, no profiler
- Fast paths are elusive
  - Pack to 4-component textures
  - Use BGRA, not RGBA
  - Use PBOs, not glTexImage2D
  - Use ping-pong FBOs, not PBuffers with glCopyTexSubImage2D
  - Use non-square textures to disable tiling
- Accurate timings are hard
  - Optimization happens at first render, not initialization
  - First readback might cause actual calculation

Improvements

- Moving targets
  - Hardware capabilities, language features, vendor- and OS-specific limitations are minimized by CUDA and OpenCL
    - But they exist, e.g. max # allowable local threads in OpenCL ranges from 1 to 1024 on various platforms.
  - Drivers are much better, but still too often beta (or worse)

- Limited precision
  - 32-bit floats are a minimum level
  - 64-bit floats now well supported, and gap is closing
  - IEEE compliant

Improvements

- Memory limitations
  - 512MB is still common, but 4GB upper limit (still smaller than equivalent host memory)
  - Array size limits many times larger, but still there
- Bandwidth to/from the card
  - >4GB/sec seen with PCIe gen2, mostly symmetric
- Changing capabilities
  - Much better as standards encompassing these uses

Improvements

- Debuggers, Profilers exist for some platforms
- Fast paths exist, but are better documented
  - Many old “fast paths” challenges were due to using graphics APIs
  - New ones are better documented and more representative of hardware
- Accurate timings are still hard
  - Asynchronicity is even more prominent
  - OpenCL includes event timers
  - Care still needed

So: Why is Using Heterogeneous Systems Hard?

- A new device type adds more parallel layers
  - One API/language to solve them all at once?
- The features that make them fast (e.g. dedicated wide memory bus) add programming complexity
- The devices we're using aren't that mature
  - At least not for how we're using them
  - As they become more mature (ECC, unified shaders), they do become easier to program
  - Compilers, tools, profilers need improvement

BACKUP SLIDES

Questions

- What makes it hard? Is it really that hard? How can we make it easier?
- Is it human aversion? Is it lack of experience or training?
- Is it the lack of a standard programming model?
- Is it the dearth of languages, compiler tools, software development environments and runtimes?
- How much of it is hardware limitations or specializations?

GPU PCB Layout

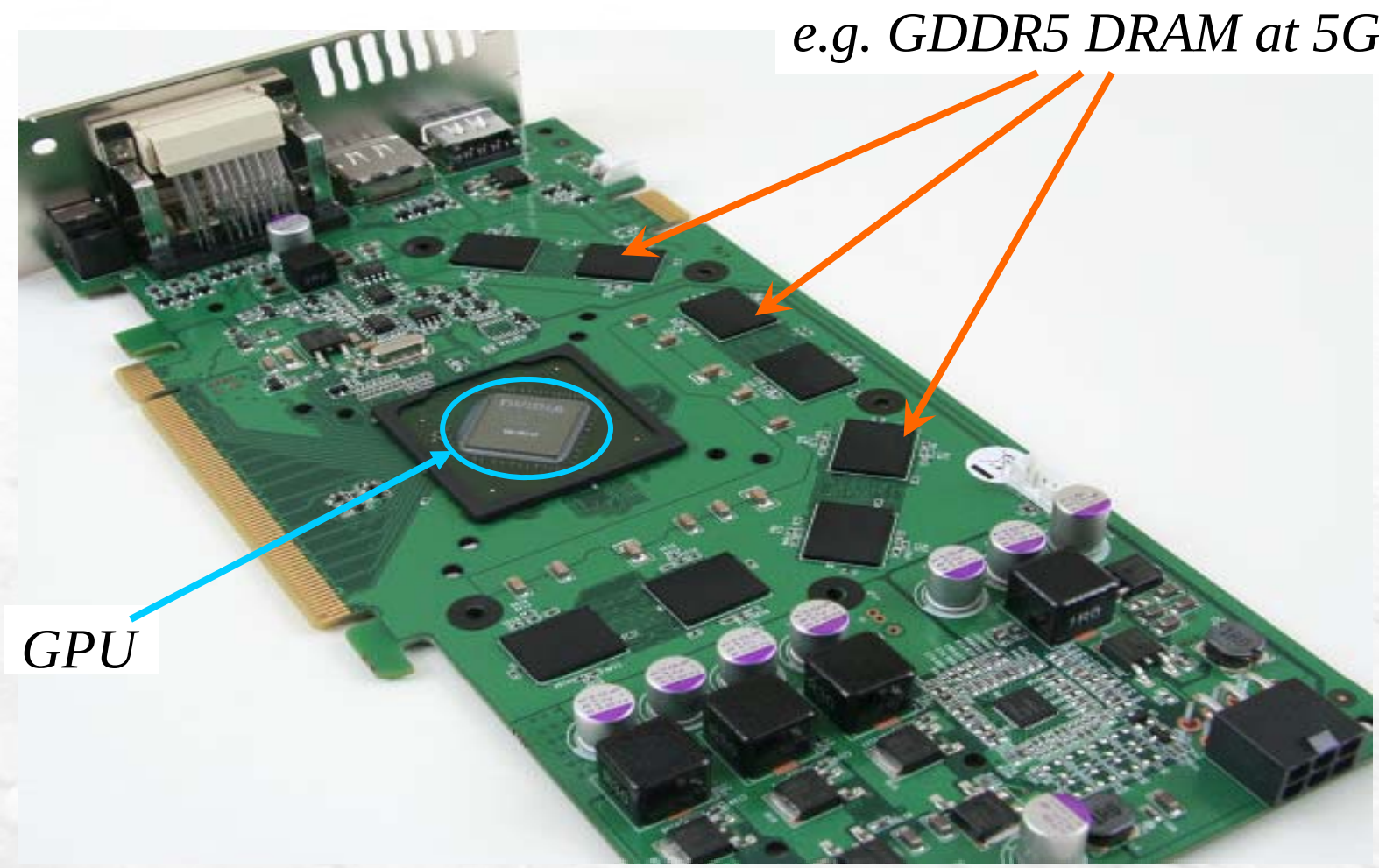

Image from http://techreport.com/articles.x/14168