The Evolution of GPUs for General Purpose Computing

Ian Buck | Sr. Director GPU Computing Software

San Jose Convention Center, CA | September 20-23, 2010
Talk Outline

- History of early graphics hardware
- First GPU Computing
- When GPUs became programmable
- Creating GPU Computing
- Future Trends and directions
First Generation - Wireframe

- Vertex: transform, clip, and project
- Rasterization: lines only
- Pixel: no pixels! calligraphic display
- Dates: prior to 1987
Storage Tube Terminals

- CRTs with analog charge “persistence”
- Accumulate a detailed static image by writing points or line segments
Early Framebuffers

- By the mid-1970’s one could afford framebuffers with a few bits per pixel at modest resolution
  - “A Random Access Video Frame Buffer”, Kajiya, Sutherland, Cheadle, 1975

- Vector displays were still better for fine position detail

- Framebuffers were used to emulate storage tube vector terminals on a raster display
Second Generation - Shaded Solids

- Vertex: lighting
- Rasterization: filled polygons
- Pixel: depth buffer, color blending
- Dates: 1987 - 1992
Third Generation - Texture Mapping

- Vertex: more, faster
- Rasterization: more, faster
- Pixel: texture filtering, antialiasing
IRIS 3000 Graphics Cards

Geometry Engines & Rasterizer

4 bit / pixel Framebuffer (2 instances)
1990’s

- Desktop 3D workstations under $5000
  - Single-board, multi-chip graphics subsystems

- Rise of 3D on the PC
  - 40 company free-for-all until intense competition knocked out all but a few players
  - Many were “decelerators”, and easy to beat
  - Single-chip GPUs
  - Interesting hardware experimentation
  - PCs would take over the workstation business

- Interesting consoles
  - 3DO, Nintendo, Sega, Sony
Before Programmable Shading

- Computing though image processing circa. 1995
  - GL_ARB_imaging
Moving toward programmability
Programmable Shaders: GeForceFX (2002)

- Vertex and fragment operations specified in small (macro) assembly language
- User-specified mapping of input data to operations
- Limited ability to use intermediate computed values to index input data (textures and vertex uniforms)

```
ADDR R0.xyz, eyePosition.xyzx, -f[TEX0].xyzx;
DP3R R0.w, R0.xyzx, R0.xyzx;
RSQR R0.w, R0.w;
MULR R0.xyz, R0.w, R0.xyzx;
ADD R1.xyz, lightPosition.xyzx, -f[TEX0].xyzx;
DP3R R0.w, R1.xyzx, R1.xyzx;
RSQR R0.w, R0.w;
MADR R0.xyz, R0.w, R1.xyzx, R0.xyzx;
MULR R1.xyz, R0.w, R1.xyzx;
DP3R R0.w, R1.xyzx, f[TEX1].xyzx;
MAXR R0.w, R0.w, (0).x;
```
Stunning Graphics Realism

Lush, Rich Worlds

Incredible Physics Effects

Core of the Definitive Gaming Platform
recent trends

multiplies per second
(observed peak)

GFLOPS

July 01    Jan 02    July 02    Jan 03    July 03    Jan 04

NVIDIA NV30, 35, 40
ATI R300, 360, 420
Pentium 4
### GPU history

#### NVIDIA historicals

<table>
<thead>
<tr>
<th>Date</th>
<th>Product</th>
<th>Process</th>
<th>Trans</th>
<th>MHz</th>
<th>GFLOPS (MUL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug-02</td>
<td>GeForce FX5800</td>
<td>0.13</td>
<td>121M</td>
<td>500</td>
<td>8</td>
</tr>
<tr>
<td>Jan-03</td>
<td>GeForce FX5900</td>
<td>0.13</td>
<td>130M</td>
<td>475</td>
<td>20</td>
</tr>
<tr>
<td>Dec-03</td>
<td>GeForce 6800</td>
<td>0.13</td>
<td>222M</td>
<td>400</td>
<td>53</td>
</tr>
</tbody>
</table>

- 1.8x increase of transistors

- 20% decrease in clock rate

- 6.6x GFLOP speedup
Early GPGPU (2002)

- Ray Tracing on Programmable Graphics Hardware
  Purcell et al.
- PDEs in Graphics Hardware
  Strzodka, Rumpf
- Fast Matrix Multiplies using Graphics Hardware
  Larsen, McAllister
  Thompson et al.
Programming model challenge

• Demonstrate GPU performance
• PHD computer graphics to do this
• Financial companies hiring game programmers

• “GPU as a processor”
Bruno (2003)

C with streams

- streams
  - collection of records requiring similar computation
    - particle positions, voxels, FEM cell, ...
      
      Ray r<200>
      float3 velocityfield<100,100,100>
    - similar to arrays, but...
      - index operations disallowed: position[i]
      - read/write stream operators:
        streamRead (positions, p_ptr);
        streamWrite (velocityfield, v_ptr);
kernels

• functions applied to streams
  – similar to for_all construct

```c
    kernel void add (float a<>, float b<>,
                     out float result<>) {
      result = a + b;
    }
```

```c
    float a<100>;
    float b<100>;
    float c<100>;

    add(a, b, c);
```

for (i=0; i<100; i++)
  c[i] = a[i] + b[i];
Challenges

Hardware
- Addressing modes
  - Limited texture size/dimension
- Shader capabilities
  - Limited outputs
- Instruction sets
  - Integer & bit ops
- Communication limited
  - Between pixels
  - Scatter \(a[i] = p\)

Software
- Building the GPU Computing Ecosystem
GeForce 7800 Pixel

- Input Registers
- Fragment Program
- Texture
- Constants
- Registers
- Output Registers
Thread Programs

Features
- Millions of instructions
- Full Integer and Bit instructions
- No limits on branching, looping
- 1D, 2D, or 3D thread ID allocation

Thread Number
Thread Program
Texture
Constants
Registers
Output Registers
Global Memory

Features
- Fully general load/store to GPU memory: Scatter/Gather
- Programmer flexibility on how memory is accessed
- Untyped, not limited to fixed texture types
- Pointer support
Shared Memory

Features

- Dedicated on-chip memory
- Shared between threads for inter-thread communication
- Explicitly managed
- As fast as registers
Managing Communication with Shared

Single thread out of cache

Multiple passes through video memory

GPU Computing

Thread Execution Manager

Shared Data

Data/Computation

Program/Control

Control

Cache

DRAM

P_1

P_2

P_3

P_4

P_n' = P_1 + P_2 + P_3 + P_4

ALU

Video Memory

P_1, P_2

P_3, P_4

P_n' = P_1 + P_2 + P_3 + P_4

P_1, P_2

P_3, P_4

P_n' = P_1 + P_2 + P_3 + P_4

P_1

P_2

P_3

P_4

P_5

P_n' = P_1 + P_2 + P_3 + P_4

ALU

Control

DRAM

Control

ALU

Control

ALU

Control

ALU

Control

ALU

Control

ALU

Control

ALU

Control

ALU
**GeForce 8800**

Build the architecture around the processor
GeForce 8800 GPU Computing

Next step: Expose the GPU as massively parallel processors
Building GPU Computing Ecosystem

- Convince the world to program an entirely new kind of processor
- Tradeoffs between functional vs. performance requirements
- Deliver HPC feature parity
- Seed larger ecosystem with foundational components
CUDA: C on the GPU

- A simple, explicit programming language solution
- Extend only where necessary
  ```
  __global__ void KernelFunc(...);
  __shared__ int SharedVar;
  KernelFunc<<< 500, 128 >>>(...);
  ```
- Explicit GPU memory allocation
  - cudaMalloc(), cudaMemcpy()
- Memory copy from host to device, etc.
  - cudaMemcpy(), cudaMemcpy2D(),...
CUDA: Threading in Data Parallel

- Threading in a data parallel world
  - Operations drive execution, not data

- Users simply given thread id
  - They decide what thread access which data element
  - One thread = single data element or block or variable or nothing....
  - No need for accessors, views, or built-ins

- Flexibility
  - Not requiring the data layout to force the algorithm
  - Blocking computation for the memory hierarchy (shared)
  - Think about the algorithm, not the data
Divergence in Parallel Computing

- Removing divergence pain from parallel programming

- SIMD Pain
  - User required to SIMD-ify
  - User suffers when computation goes divergent

- GPUs: Decouple execution width from programming model
  - Threads can diverge freely
  - Inefficiency only when granularity exceeds native machine width
  - Hardware managed
  - Managing divergence becomes performance optimization
  - Scalable
Foundations

- Baseline HPC solution
- Ubiquity: CUDA Everywhere
- Software
  - C99 Math.h
  - BLAS & FFT
  - GPU co-processor
- Hardware
  - IEEE math (G80)
  - Double Precision (GT200)
  - ECC (Fermi)
Customizing Solutions

Ported Applications

Domain Libraries

Domain specific lang

C

Driver API

PTX

HW

Ease of Adoption

Generality
PTX Virtual Machine and ISA

- **PTX Virtual Machine**
  - Programming model
  - Execution resources and state
  - Abstract and unify target details

- **PTX ISA – Instruction Set Architecture**
  - Variable declarations
  - Data initialization
  - Instructions and operands

- **PTX Translator (OCG)**
  - Translate PTX code to Target code
  - At program build time
  - At program install time
  - Or JIT at program run time

- **Driver implements PTX VM runtime**
  - Coupled with Translator
GPU Computing Software Libraries and Engines

Application Acceleration Engines (AXEs)
SceniX, CompleX, Optix, PhysX

Foundation Libraries
CUBLAS, CUFFT, CULA, NVCUVID/VENC, NVPP, Magma

Development Environment
C, C++, Fortran, Python, Java, OpenCL, Direct Compute, ...

CUDA Compute Architecture
Directions

- Hardware and Software are one
- Within the Node
  - OS integration: Scheduling, Preemption, Virtual Memory
  - Results: Programming model simplification
- Expanding the cluster
  - Cluster wide communication and synchronization
- GPU on-load
  - Enhance the programming model to keep more of the computation (less cpu interaction) and more of the data (less host side shadowing).
Thank you!

Additional slide credits: John Montrym & David Kirk