GPU Computing Past, Present, Future

Ian Buck, GM GPU Computing Sw
History....

Stream Computing on Graphics Hardware

Ian Buck

GPGPU in 2004
recent trends

( observed peak )

- NVIDIA NV30, 35, 40
- ATI R300, 360, 420
- Pentium 4

GFLOPS

months:
- July 01, Jan 02, July 02, Jan 03, July 03, Jan 04
GPU history

<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>

translating transistors into performance

- 1.8x increase of transistors
- 20% decrease in clock rate
- 6.6x GFLOP speedup
Stunning Graphics Realism

Lush, Rich Worlds

Incredible Physics Effects

Core of the Definitive Gaming Platform
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”
C with streams

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

- Graphics API
- Addressing modes
  - Limited texture size/dimension
- Shader capabilities
  - Limited outputs
- Instruction sets
  - Integer & bit ops
- Communication limited
  - Between pixels
  - Scatter $a[i] = p$
GeForce 7800 Pixel

Input Registers

Fragment Program

Output Registers

Texture

Constants

Registers
Thread Programs

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

CPU

GPGPU

CUDA

GPU Computing

Single thread out of cache

Multiple passes through video memory

Data/Computation

Program/Control
CUDA: C on the GPU

- A simple, explicit programming language solution
- Extend only where necessary
  ```c
  __global__ void KernelFunc(...);
  __shared__ int SharedVar;
  KernelFunc<<< 500, 128 >>>(...);
  ```
- Explicit GPU memory allocation
  - `cudaMalloc()`, `cudaFree()`
- 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
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
GPU Computing By the Numbers:

- >350,000,000 Compute Capable GPUs
- >1,000,000 Toolkit Downloads
- >120,000 Active CUDA Developers
- >450 Universities Teaching CUDA
- 100% OEMs offer CUDA GPU PCs
Customizing Solutions

Ease of Adoption

Ported Applications

Domain Libraries

Domain specific lang

C

Driver API

PTX

HW

Generality
CUDA Math Libraries

High performance math routines for your applications:

- cuFFT - Fast Fourier Transforms Library
- cuBLAS - Complete BLAS Library
- cuSPARSE - Sparse Matrix Library
- cuRAND - Random Number Generation (RNG) Library
- NPP - Performance Primitives for Image & Video Processing
- Thrust - Templated Parallel Algorithms & Data Structures
- math.h - C99 floating-point Library

Included in the CUDA Toolkit  Free download @ www.nvidia.com/getcuda

More information on CUDA libraries:

http://www.nvidia.com/object/gtc2010-presentation-archive.html#session2216
cuFFT: Multi-dimensional FFTs

- New in CUDA 4.1
  - Flexible input & output data layouts for all transform types
    - Similar to the FFTW “Advanced Interface”
    - Eliminates extra data transposes and copies
  - API is now thread-safe & callable from multiple host threads
  - Restructured documentation to clarify data layouts

\[ F(x) = \sum_{n=0}^{N-1} f(n)e^{-j2\pi(x \frac{n}{N})} \]
\[ f(n) = \frac{1}{N} \sum_{n=0}^{N-1} F(x)e^{j2\pi(x \frac{n}{N})} \]
cuSPARSE: Sparse linear algebra routines

- Sparse matrix-vector multiplication & triangular solve
  - APIs optimized for iterative methods
- New in 4.1
  - Tri-diagonal solver with speedups up to 10x over Intel MKL
  - ELL-HYB format offers 2x faster matrix-vector multiplication

\[
\begin{bmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  y_4
\end{bmatrix} = \alpha \begin{bmatrix}
  1.0 & & & \\
  2.0 & 3.0 & & \\
  & & 4.0 & \\
  5.0 & 6.0 & 7.0 &
\end{bmatrix} \begin{bmatrix}
  1.0 \\
  2.0 \\
  3.0 \\
  4.0
\end{bmatrix} + \beta \begin{bmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  y_4
\end{bmatrix}
\]
cuRAND: Random Number Generation

- Pseudo- and Quasi-RNGs
- Supports several output distributions
- Statistical test results reported in documentation

- New commonly used RNGs in CUDA 4.1
  - MRG32k3a RNG
  - MTGP11213 Mersenne Twister RNG
Developer ecosystem enables the application growth
Directives: Simple Hints for the Compiler

Your original C/Fortran code

main() {
  ...
  <serial code>
  ...
  #pragma acc region
  {
    <compute intensive code>
  }
  ...
}

Compiler Hint

Add hints to code

On-ramp to parallel computing

Compiler does heavy lifting of parallelizing code

Works on multicore CPUs & many core GPUs
OpenACC: Open Programming Standard for Parallel Computing

http://www.openacc-standard.org
2x in 4 Weeks. Guaranteed.

Free 30 day trial license
to PGI Accelerator*

Tools for quick ramp

www.nvidia.com/2xin4weeks

*Limit 1000 developers
Directives Program Hugely Successful

Real-Time Object Detection
*Global Manufacturer of Navigation Systems*

Valuation of Stock Portfolios using Monte Carlo
*Global Technology Consulting Company*

Interaction of Solvents and Biomolecules
*University of Texas at San Antonio*

---

Optimizing code with directives is quite easy, especially compared to CPU threads or writing CUDA kernels. The most important thing is avoiding restructuring of existing code for production applications.

-- Developer at the Global Manufacturer of Navigation Systems
NVIDIA Opens Up CUDA Platform

CUDA Compiler Source for Researchers & Tools Vendors

Enables
New Language Support

New Processor Support

Apply for early access at http://developer.nvidia.com/cuda-source
New Opportunities for Developer Tools

Guided Workflow

Automated Analysis

Drill Down to Expert Guidance

New CUDA 4.1 Release
New CPU Architectures emerging

CUDA for ARM Development Kit

CUDA GPU  Tegra ARM CPU

Available: 1H 2012

256 Tegra (ARM) CPUs
+ 256 CUDA GPUs
Building blocks for Exascale

## GPU Direct

- **CPU**
- **Chip set**
- **InfiniBand**
- **GPU**
- **GPU Memory**

## Atomic Ops

Atomic operations for thread-to-thread communication

```plaintext
atom{.space}.op.type d, [a], b;
atom{.space}.op.type d, [a], b, c;
.space = { .global, .shared };
.op = { .and, .or, .xor, //
   .cas, .exch, //
   .add, //
   .inc, .dec, //
   .min, .max }; //
.type = { .b32, .b64,
   .u32, .u64,
   .s32,
   .f32 };
```

## Dynamic Parallelism
GPUDirect v3

InfiniBand

Direct transfers between GPU and IB

© NVIDIA Corporation 2011

NVIDIA Confidential
Unified Virtual Addressing

*Easier to Program with Single Address Space*

**No UVA: Multiple Memory Spaces**

- **System Memory**
  - 0x0000 - 0xFFFF

- **GPU0 Memory**
  - 0x0000 - 0xFFFF

- **GPU1 Memory**
  - 0x0000 - 0xFFFF

**UVA: Single Address Space**

- **System Memory**
  - 0x0000 - 0xFFFF

- **GPU0 Memory**
  - 0x0000 - 0xFFFF

- **GPU1 Memory**
  - 0x0000 - 0xFFFF

NVIDIA Confidential
Better Load Balancing

Dynamic workloads are hard to balance

- when work-per-block is data-dependent
- e.g. Adaptive Mesh CFD

For this example

- GPU occupancy improves by ~1.4x
- Runtime ~1.4x faster

(shading indicates compute loading)
Building A Massively Parallel World

- The Future is Heterogeneous
- Many solutions build a heterogeneous world
  - General Purpose Languages
  - Domain Specific Languages
  - Directives & Open Compiler Platform
  - New CPUs enable new opportunities
  - Platform for Exascale
  - Discovering and Sharing
- Education & Research
  - Where are the world’s parallel programmers?