GPU Programming and Computing

Johan Seland
johan.seland@sintef.no

Workshop on High-Performance and Parallel Computing
Simula Research Laboratory
October 24, 2007

Offline rendering five years ago

Shrek (Dreamworks/PGI)

Interactive Rendering five years ago

Quake 3 (id Software)

Offline Rendering today

Madagascar (Dreamworks/PGI)

Interactive Rendering today

Project Gotham Racing

Enter the GPU

This is because of the Graphical Processing Unit(GPU)

- Fall 2002: Intel Pentium IV
  - 3.06 GhZ
- Fall 2007: Intel Core 2 Ext.
  - 4 $\times$ 3.00GhZ
  - 582 million transistors (Follows Moores Law)

- Fall 2002: Nvidia GeForce 4
  - 250 MhZ
- Fall 2007: Nvidia Geforce 8
  - 1.35 GhZ
  - 680 million transistors (Follows Moores Law)

The PC of 2007

- 4 core CPU
  - 96 GFLOPS(peak)
  - 7750,- NOK
- GPU
  - 330 GFLOPS (observed)
  - 4299,- NOK
- Interconnects
  - 1 GiB/s CPU $\leftrightarrow$ GPU
  - 21 GiB/s CPU $\leftrightarrow$ system memory (peak)
  - 55.2 GiB/s GPU $\leftrightarrow$ graphics memory (observed)

It is quite clear where performance is located.

Outline

1. Capabilities of GPUs
2. GPU Programming
3. Successful Applications

Why are GPUs fast?

- Traditional CPU designs use $\approx$ 50% of transistors for cache and control logic, not computations
  - The nature of GPUs makes it easier to use additional transistors for computation
  - This comes at the cost of flexibility
- CPU industry is moving from "instructions per second" to "instructions per watt"
  - "Power wall" is now all important
  - We can not scale voltage like we used to
  - We can not scale clock as we used to
- Video game market drives innovation

Characteristics of a GPU

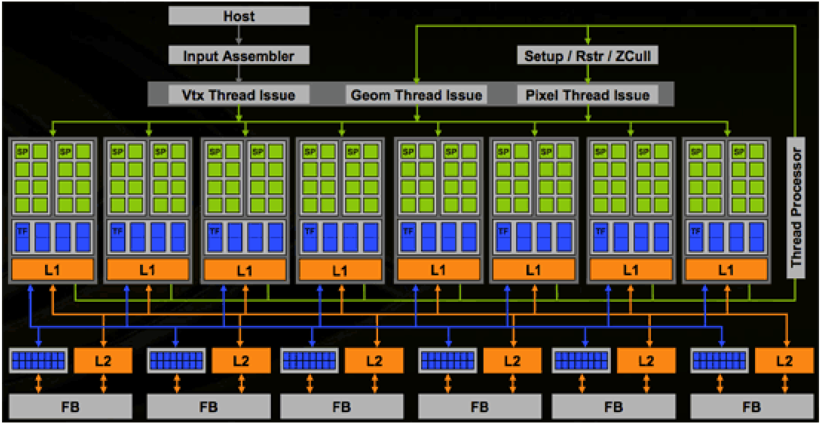
Nvidia G80 - Released fall 2006

- 128 Stream Processors
  - Fused Multiply And Add
  - Trigonometric functions in once cycle
  - (almost) IEEE 754 Single-precision (32 bit)
  - Scalar processor
- Core clock up to 1.35GhZ
- Up to 2GiB Memory
- 680 million transistors
- Two can be run in parallel
- $\approx$ 300 Watt (under load) $\rightarrow \approx$ 1.1 Flops/Watt
  - $\approx$ 0.7 Flops/Watt for Quad core CPU

Floating point on GPUs (as of 2007)

- Only 32-bit (single precision)
  - Announced 64-bit precision at half speed
  - Possible remedy: Correction steps in 64-bit precision
- Lacks denormalized numbers
- Lacks signalling of NaNs
- Rounding mode can not be changed
- Lower precision for division and square root
- Floating point $\longrightarrow$ integer conversion not fully IEEE-754 compliant

Nvidia G80 block diagram



- Very little of this is graphic specific
- ...but, assumes threads are independent

Limitations of GPUs

If the GPU is so great, why are we still using the CPU?

You can not simply “port” existing code and algorithms!

- Data-stream mindset required
  - Parallel algorithms
  - New data structures (dynamic data structures are troublesome)
- Not suitable to all problems
  - Pointer chasing impossible or inefficient
  - Recursion
- Debugging is hard
  - Hardware is designed without debug bus
  - Driver is closed
- Huge performance cliffs
- No standard API
  - More about this later...

“Hyper-Core” computers

Speculation about the computer of the next decade:

- 10s of CPU cores
  - Use for scheduling
  - Use for “irregular” part of problem
  - Maybe higher precision (correction steps)
- 100s of GPU cores
  - Use for “regular” part of problem
- NUMA (Non-Uniform Memory Access) for both
  - Programming languages must expose this
  - Runtime systems?
  - Always out-of-(some)-core
- Clusters of these?
  - OpenMP/MPI not sufficient

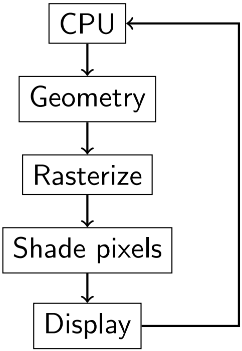
Outline

1. Capabilities of GPUs
2. GPU Programming
3. Successful Applications

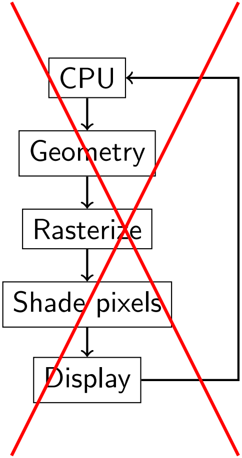
GPU Programming

- GPUs have traditionally been closed architectures.
  - Must program them through closed-source graphics driver
  - Driver is like an OS (threads, scheduling, protected memory)
- OpenGL/DirectX are standard, but
  - Designed for graphics, not general purpose computations
  - Many revisions of each standard
    - New revisions for each HW-generation
  - Allows for "capabilities"
  - Large variations between vendors
- Both vendors now have dedicated GPGPU APIs
  - Nvidia CUDA (Compute Unified Device Architecture)
  - AMD CTM (Close To Metal)
- GPGPU "version" of hardware as well

Computer graphics 101



- Pre-2007: Hardware mimicked graphics APIs
- It is possible to formulate many problems in this framework
  - Uses graphics APIs
  - “Classical GPGPU”



- Pre-2007: Hardware mimicked graphics APIs
- It is possible to formulate many problems in this framework
  - Uses graphics APIs
  - "Classical GPGPU"

DO NOT DO THIS ANYMORE!

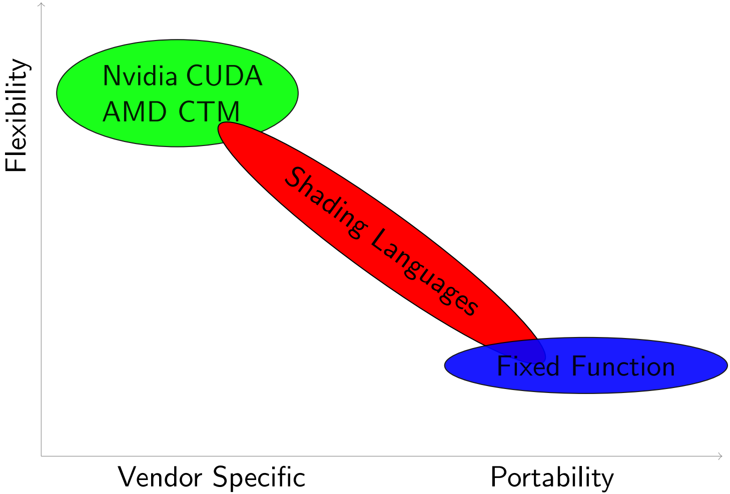
(Unless for graphics)

Nvidia CUDA (Compute Unified Device Architechure)

C-like API for programming newer Nvidia GPUs

- Computation kernels are written in C
  - Compiles with dedicated compiler, `nvcc`
- Kernels are executed as threads, threads organized into blocks
  - Programmer decides #threads, #threads/block, and mem/block
- Exposes different kinds of memory
  - Thread-local (register)
  - Shared per block
  - Global (not cached, write everywhere)
  - Texture (cached read only memory)
  - Constant(cached read only memory)
- Some synchronization primitives
- `cudaMalloc`, `cudaMemcpy` for allocating and copying memory

Properties of APIs



Outline

1 Capabilities of GPUs

2 GPU Programming

3 Successful Applications

Hyperbolic Conservations Laws

- Most high-resolution schemes for conservations laws are explicit
- Explicit schemes are embarrassingly (pleasantly?) parallel
- Algorithm is numerically stable, suitable for single-precision
- Complex schemes $\longrightarrow$ High number of arithmetic operations per memory operation
- Finite speed of wave propagation $\longrightarrow$ Easy to decompose computational domain into subdomains
  - Overcomes lack of memory on GPUs
  - Obvious potential for cluster implementations

Hyperbolic Conservations Laws Cont.

- Scheme with low arithmetic intensity

Grid size	CPU ms[1]	GPU ms	Speedup
128×128	2.22	0.23	9.5
256×256	9.09	0.46	19.8
512×512	37.10	1.47	25.2
1024×1024	1248.00	5.54	26.7

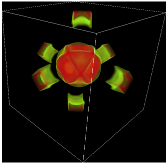
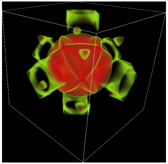
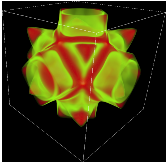
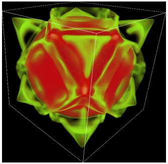
- Scheme with high arithmetic intensity

Grid size	CPU ms	GPU ms	Speedup
128×128	30.6	1.27	24.2
256×256	122.0	4.19	29.1
512×512	486.0	16.80	28.9
1024×1024	2050.0	68.30	30.0

[1] Per time step

3D Euler Equations

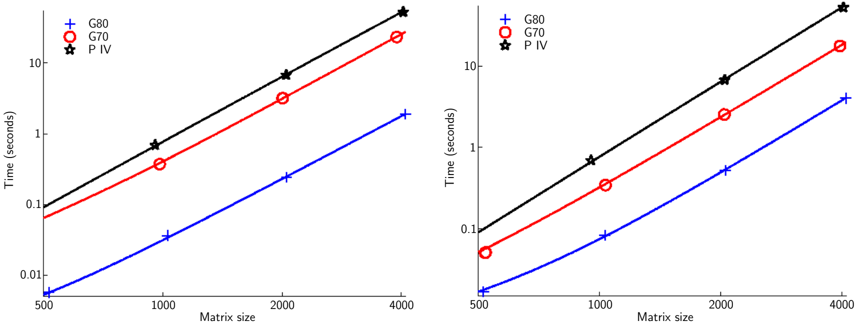
- Images show a circular explosion inside a cubic container

- Runtimes per time step in seconds of the Rayleigh-Taylor instability on a $N \times N \times N$ grid.

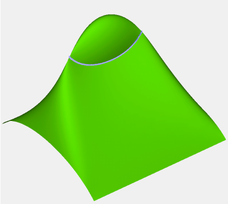
Grid Size	CPU ms	GPU ms	Speedup
49^3	5.23e-1	4.16e-2	12.6
64^3	1.14e-0	8.2e-2	13.9
81^3	1.98e-0	1.72e-1	11.5

Matrix Multiplication

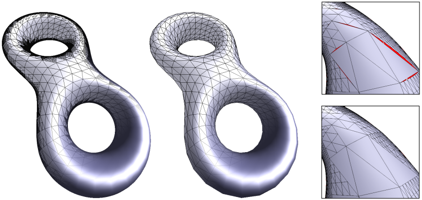


Single Pass　　　　Multi Pass

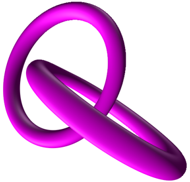
- Speedup is around 30× for dense matrix multiply
- 7× for PLU factorization



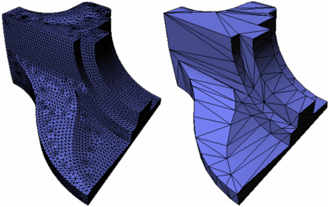
Self intersections



Dynamic silhouette refinement



Algebraic Geometry



Preparation of FEM grids

Conclusion

- The GPU is the only parallel processor that has seen widespread success
- Allows us to experiment with 100s of cores today
- Not just a toy anymore
- Future is definitively parallel, but what kind of parallel?
- Memory management is a very open problem