HPEC 2007

Norm Rubin
Fellow
AMD Graphics Products Group
norman.rubin at amd.com

Overview

- Why GPU
- What is the difference between GPU/CPU
- 2900 introduction
- AMD programming model

GPGPU

- GPGPU means not rendering (not visual output)
- GPUs are getting closer to CPU functionality, branches float operations etc
- GPUs have huge floating point performance (2 * 500 gflops)
- GPUs are cheaper per mflop (less then ¼ the cost)
- Once you need to recode for mult-core, any crazy idea starts to look promising, improved clock speed is over
- GPU has new programming models, new languages, new issues with optimization

GPU Programming Systems

- OpenGl or DirectX
  - CG/HLSL/OpenGl shading language
- GPGPU languages
  - Accelerator (MicroSoft research)
  - Brook (Stanford)
  - CTM (AMD)
  - CUDA (NVIDIA)
  - RapidMind et al

Published results (kernels)

- Large matrix/vector operations (BLAS)
- Protein Folding (Molecular Dynamics)
- Finance modeling
- FFT (SETI, signal processing)
- Raytracing
- Physics Simulation [cloth, fluid, collision,…]
- Sequence Matching (Hidden Markov Models)
- Speech/Image Recognition (Hidden Markov Models, Neural nets)
- Databases
- Sort/Search
- Medical Imaging (image segmentation, processing)
- And many, many, many more…

Speedup switching to GPU (kernel code)

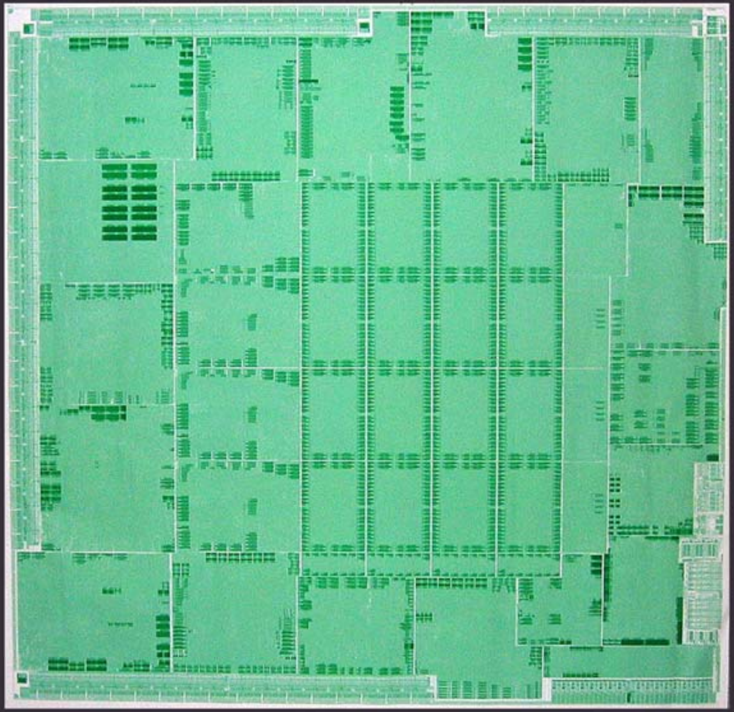
- Simple ports of data parallel programs get 5-10 times faster. Algorithm is data parallel to start, problem fits on machine

- Smart ports (make use of tiling/compression, recode understanding the GPU, gets 20-100 times faster) – change of algorithm.

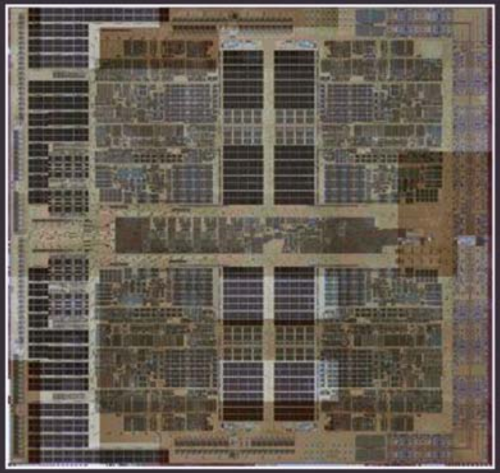
Why are GPU’s getting faster

- ***Why are GPUs getting faster so fast?***
- – Arithmetic intensity- The specialized nature of GPUs makes it easier to use additional transistors for computation
- – Economics - Multi-billion dollar video game market drives innovation
- - Games and images do not need the same kind of accurate results as GPGPU, so lots of fast but odd arithmetic, e.g. no denorms

GPU vs CPU – what is the difference?



HD 2900 - 80 nano



Barcelona 65 Nano

CPU vs GPU

CPU	GPU
5% of area is ALU	40 % of area is ALU
Mem – low latency (1/10) of GPU	Mem – high bandwidth (10 times CPU)
Big Cache (10 times GPU)	Small Cache
Full IEEE + doubles	Partial IEEE
4 cores	64+ cores
Just load stores	Fancy memory – tiling – arithmetic in memory
	10 Times the flops of CPU

Chip design point

- CPU

Lots of instructions, little data
- Out of order exec
- Branch prediction

- Reuse and locality
- Task parallel
- Needs OS
- Complex sync
- Backward compatible
- Little functional change
- Agreement on instructions

- GPU

Little instructions, lots of data
- SIMD
- Hardware threading

- Little reuse
- Data parallel
- No OS
- Simple sync
- Not backward compatible
- Fast/frequent functional change

CPU vs GPU performance

- Peak performance = 1 float op per cycle
- Program:
- series of loads (1-6)

```
r1 = load (index1)
r2 = load (index2)
r3 = r1 + r2
```

series of muladds (1-100)

```
r4 = r3 * r3 + r3
r5 = r4 * r4 + r4
```

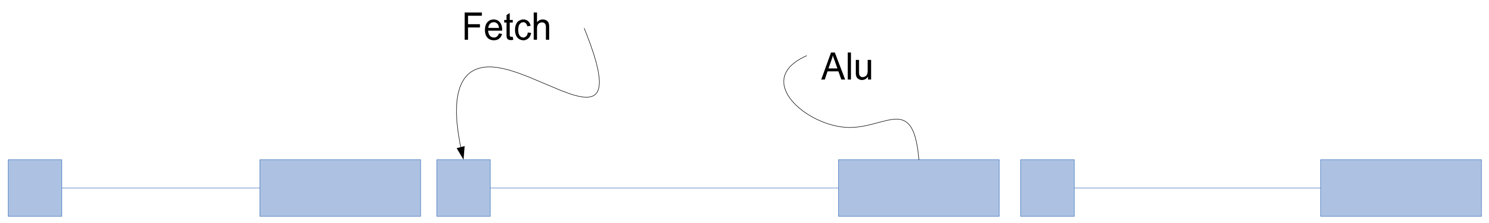
Run over a very large (out-of-cache) data set. (stream comp)

Can you get peak performance/multi-core/cluster?

CPU operation

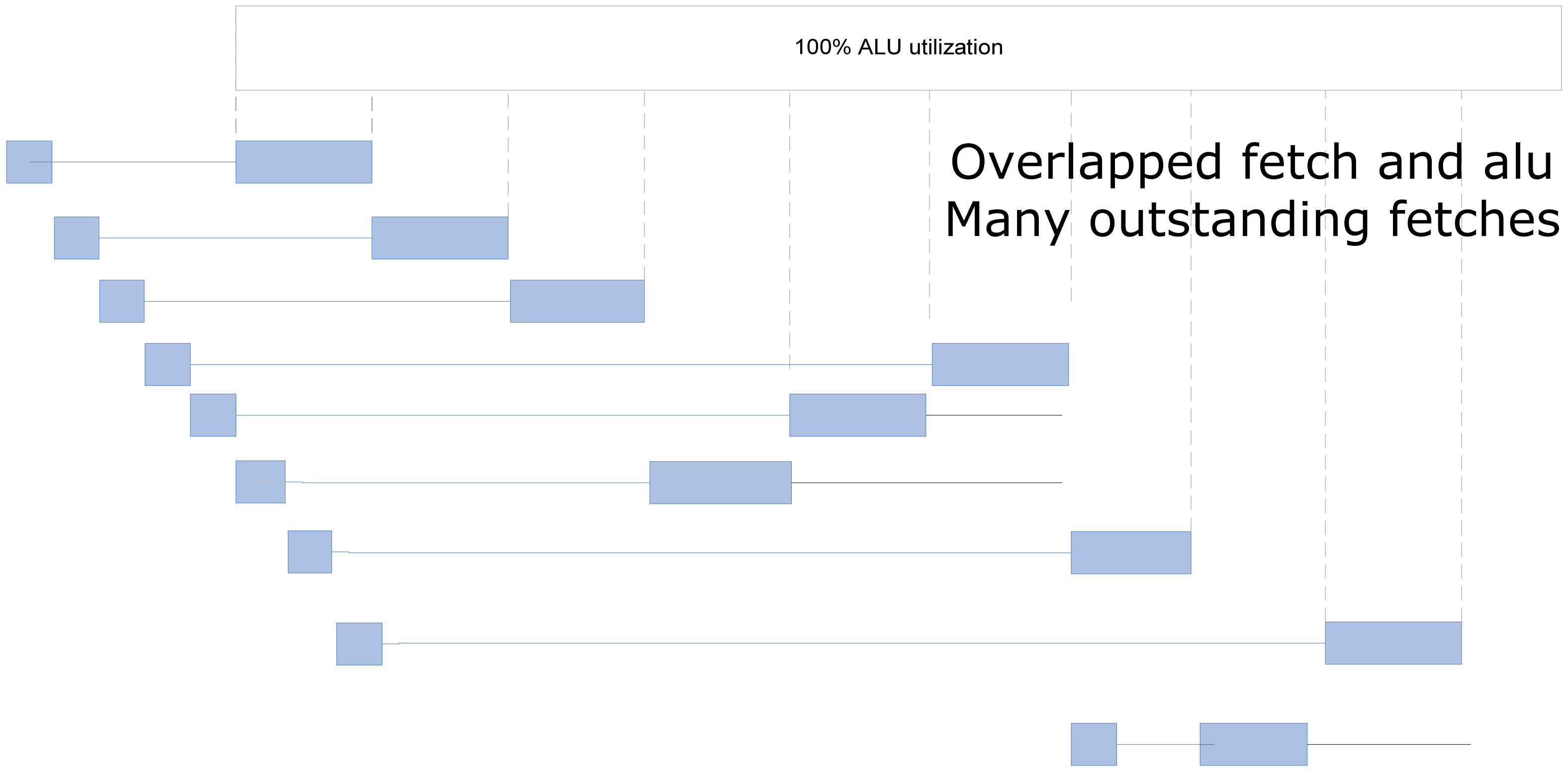
One iteration at a time
Single CPU unit
Cannot reach 100%

Hard to prefetch
Multi-core does not help
Cluster does not help
Limited number of outstanding fetches



Wait for memory, gaps prevent peak performance
Gap size varies dynamically
Hard to tolerate latency

GPU THREADS (lower clock – different scale)

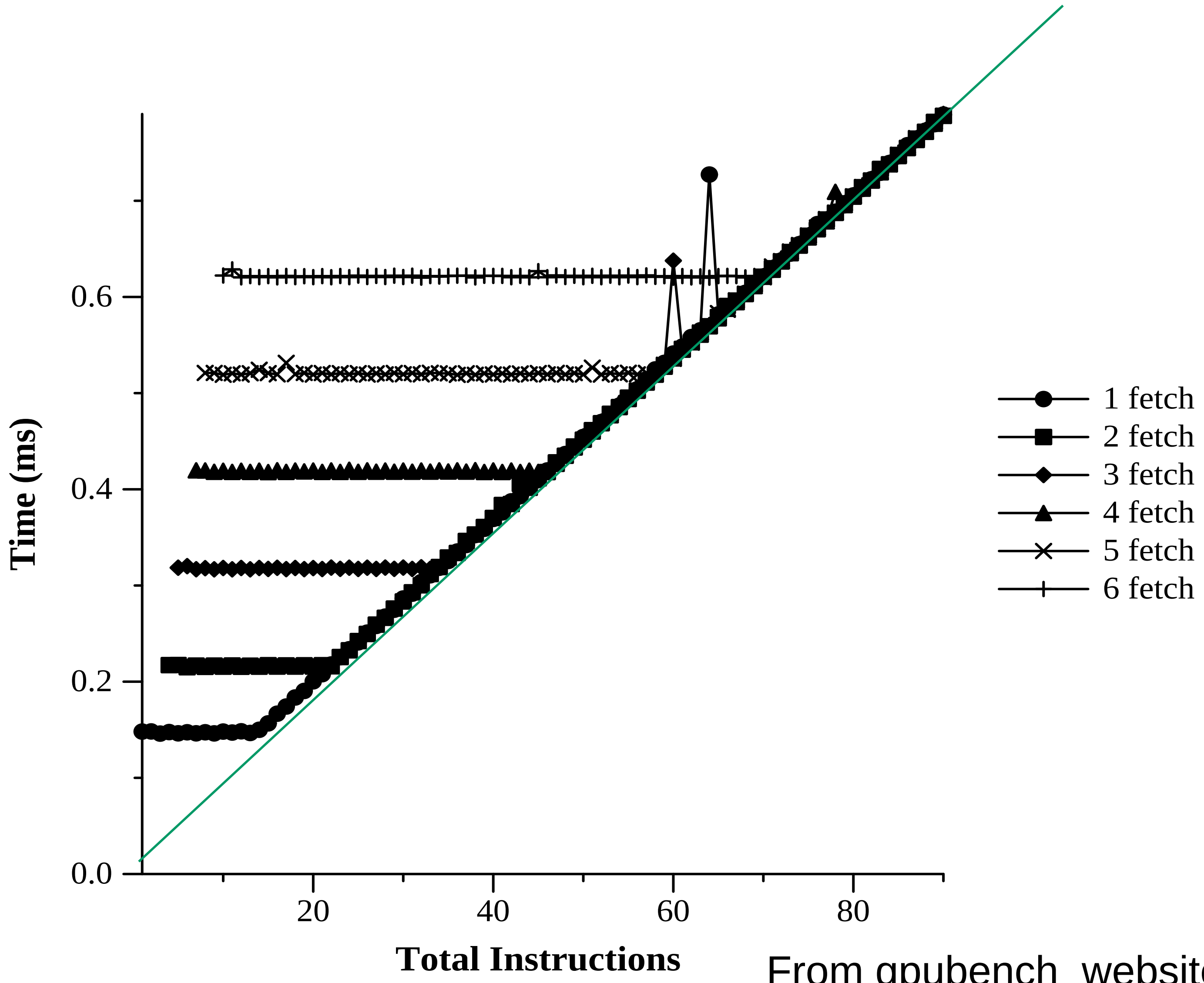


Overlapped fetch and alu
Many outstanding fetches

ALU units reach 100% utilization
Hardware sync for final Output

Lots of threads
Fetch unit + ALU unit
Fast thread switch
In-order finish

GPU performance (reaches peak)



From gpubench website

Implications

- Compute is cheap but you need lots of parallelism to keep all those alu’s busy. (graphics shading is highly parallel)
- Bandwidth can be expensive (500 cycle memory latency)
- Compute goes up by 70% a year but bandwidth goes up by 25% a year, latency goes down by 5% a year (arithmetic intensity – lots of alu ops per read)
- GPU wins when arithmetic intensity is high
- GPU wins when streaming (little reuse – lots of data) saxpy fft – sequential performance

What does a real machine look like?

AMD Radeon HD™ 2900 Highlights

Technology leadership

- Clock speeds – 742 MHz
- Transistor – 700 million
- Technology Process - TSMC 80nm HS
- Power ~215 W, Pin Count - 2140
- Die Size 420mm^2 (20mm x 21mm)

2nd generation unified architecture

- Scalar ALU design with 320 stream processing units
- 475 GigaFLOPS of (MulAdd) compute
- 47.5 GigaPixels/Sec & 742 Mtri/sec
- 106 GB/sec Bandwidth
- Optimized for Dynamic Game Computing and Accelerated Stream Processing

DirectX® 10

- Massive shader and geometry processing performance
- Shader Model 4.0 with Integer support
- Enabling the next generation of visual effects

Cutting-edge image quality features

- Advanced anti-aliasing and texture filtering capabilities
- Fast High Dynamic Range rendering
- Programmable Tessellation Unit

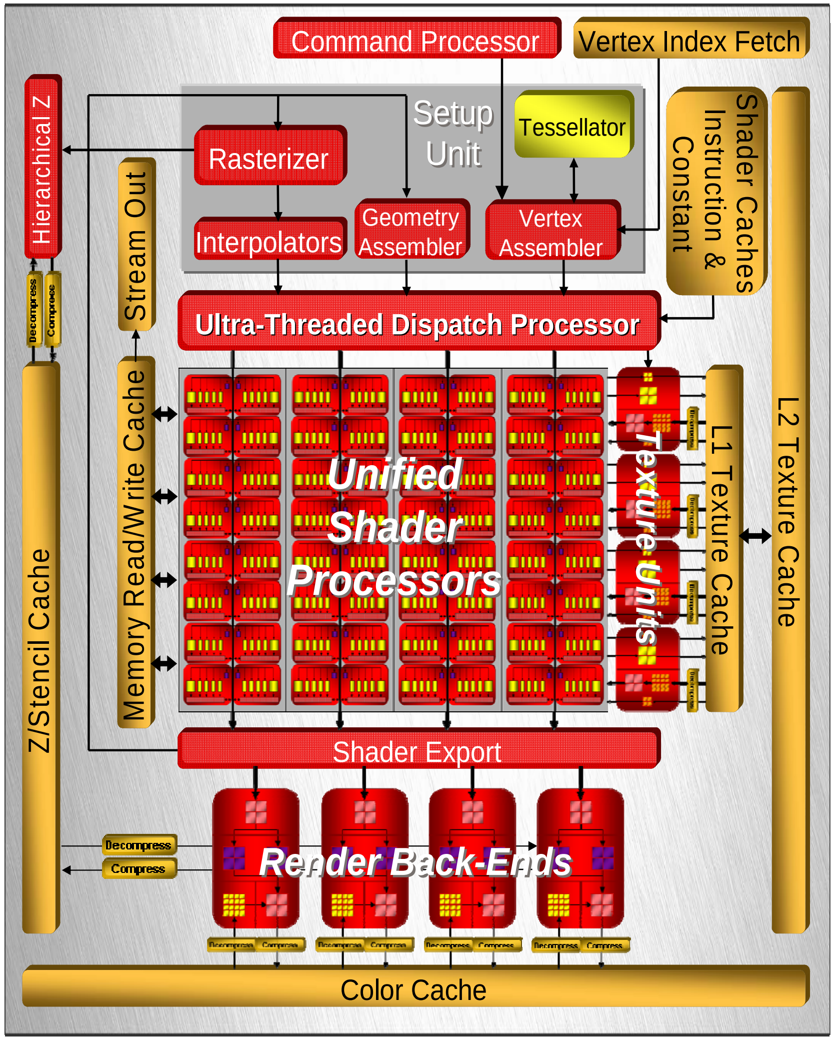
ATI Avivo™ HD technology

- Delivering The Ultimate Visual Experience™ For HD video
- HD display and audio connectivity
- HD DVD and Blu-Ray capable

Native CrossFire™ technology

- Superior multi-GPU support
- Scales up rendering performance and image quality with 2 or more GPUs

AMD Radeon HD2900 Graphics Unit
2nd Generation Unified Shader Architecture



- Development from proven and successful XBOX 360 graphics
- New dispatch processor handling thousands of simultaneous threads
- Instruction Cache and Constant Cache for unlimited program size
- Up to 320 discrete, independent stream processing units
- Vliw ALU implementation
- Dedicated branch execution units
- Three dedicated fetch units
  - Texture Cache
  - Vertex Cache
  - Load/Store Cache
- Full support for DirectX 10.0, Shader Model 4.0

Shader Processing Units (SPU)

Arranged as 5-way vliw stream processors

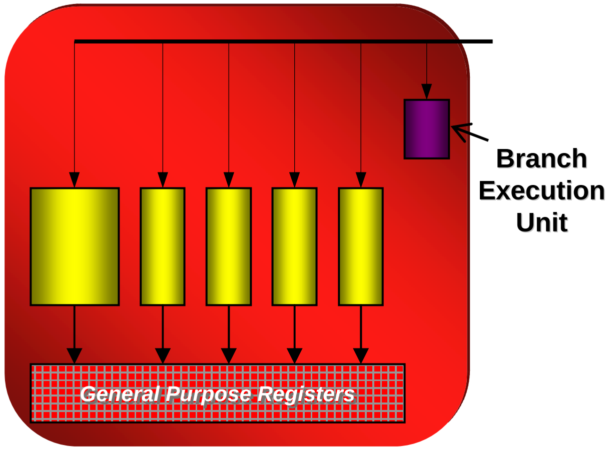
- Co-issue up to 5 scalar FP MAD (Multiply-Add)
- Up to 5 integer operations supported (cmp, logical, add)
- One of the 5 stream processing units additionally handles
  - * transcendental instructions (SIN, COS, LOG, EXP, RCP, RSQ)
  - * integer multiply and shift operations
- 32-bit floating point precision (round to nearest even)

Branch execution units handle flow control and conditional operations

- Condition code generation for full branching
- Predication supported directly in ALU

General Purpose Registers

- 1 MByte of GPR space for fast register access



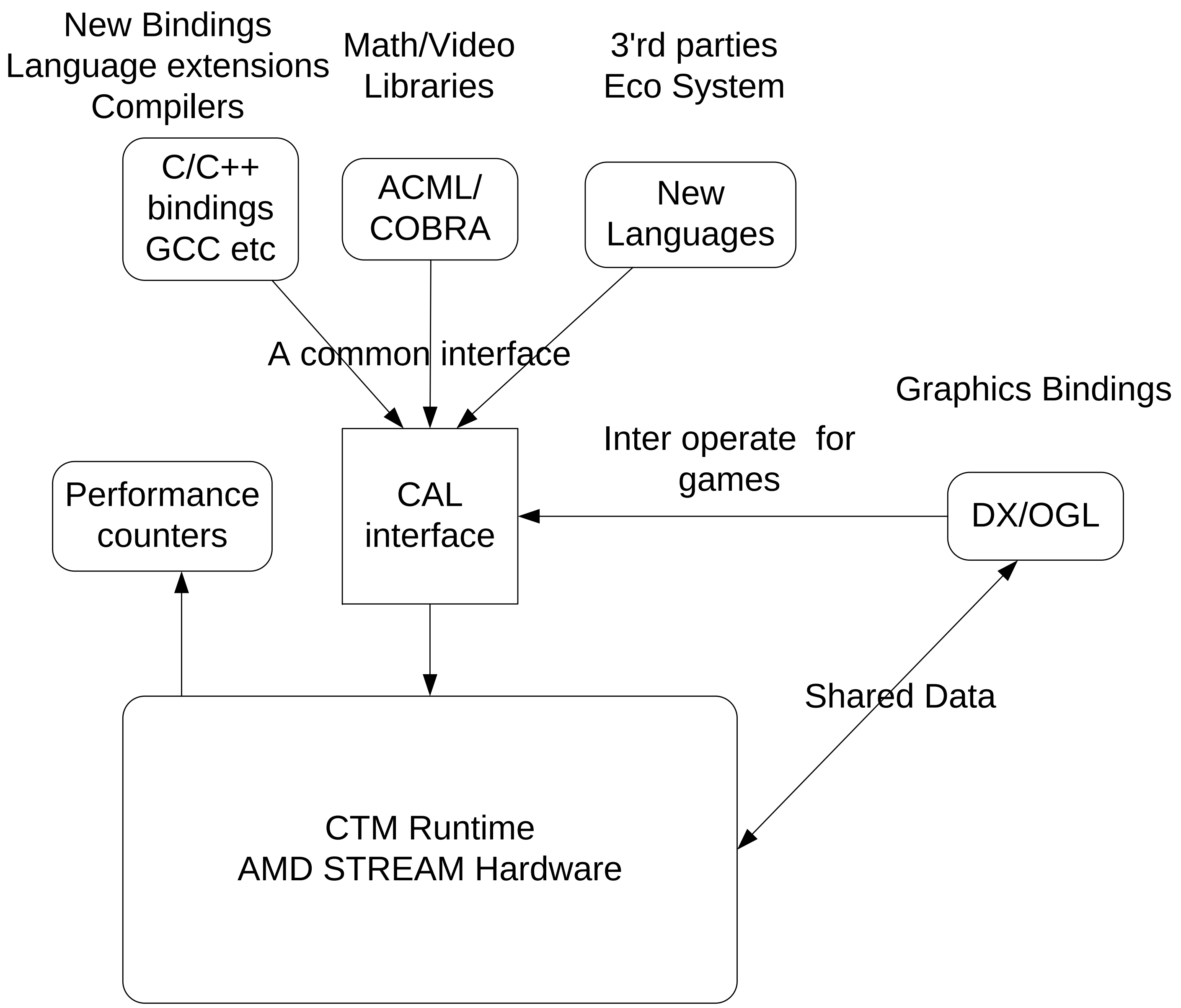
Active Thread count

- Assume a thread needs 5 registers
- Each simd has 256 sets of 64 vector registers
- 256/5 = 51
- 51*64 threads per simd
- 4 simd engines so 51*64*4 active threads = 13056
- Each register holds 4 – 32 bit values

CTM Introduction

- The Close-To-the-Metal interface
- CTM lets developers see a data parallel machine, not a vertex/pixel processor
- CTM removes all unnecessary driver overhead, but exposes all the special features of the processor
- CTM is a system developer interface, not an end user view
- AMD is building a application interface on top of CTM

AMD Accelerated Computing Software



Random number generation

- Pseudorandom number generation on the GPU, Sussman et. al. Graphics Hardware 2006
- Combined Explicit Inverse Congruent Generator
- Some issues
  - Explicit so no state needs to be saved
  - Slow to compute
  - Not proven that parallel random generators are uncorrelated.

10 times CPU speed (but slower then a CPU using a more standard algorithm)

Hardware issues that determined the algorithm

- – 16 outputs per thread (removed)
- – No integer operations (removed)
- – Inexact division (partially removed – no denorms)

One year later

- Can implement Mersenne twister algorithm
  - Requires integer ops + saved state
- parallel version can be proved uncorrelated
  - 16k parallel generators
- Approx 100 times faster (CPU/GPU) on current hardware
- Current limitation – pci bus – getting data into and out of the GPU -

Observations

- Random number generation moved from Odd-ball technique to what might be overkill for simulation.
- In one generation the GPU got both faster and better!
- Unlike CPU, GPU systems are rapidly changing.
- You may need to recode algorithms each rev.
- What remained constant: you need:
  - Massively parallel + lots of arithmetic intensity.

Disclaimer & Attribution