Challenges for GPU Architecture

Michael Doggett
Graphics Architecture Group
April 2, 2008

Graphics Processing Unit

- Architecture
  - CPUs vs GPUs
  - AMD’s ATI RADEON 2900
- Programming
  - Brook+, CAL, ShaderAnalyzer
- Architecture Challenges
  - Accelerated Computing


ATI WhiteOut Demo [2007]
Architecture

Chip Design Focus Point

CPU	GPU
Lots of instructions little data Out of order exec Branch prediction	Few instructions lots of data SIMD Hardware threading
Reuse and locality	Little reuse
Task parallel	Data parallel
Needs OS	No OS
Complex sync	Simple sync
Latency machines	Throughput machines

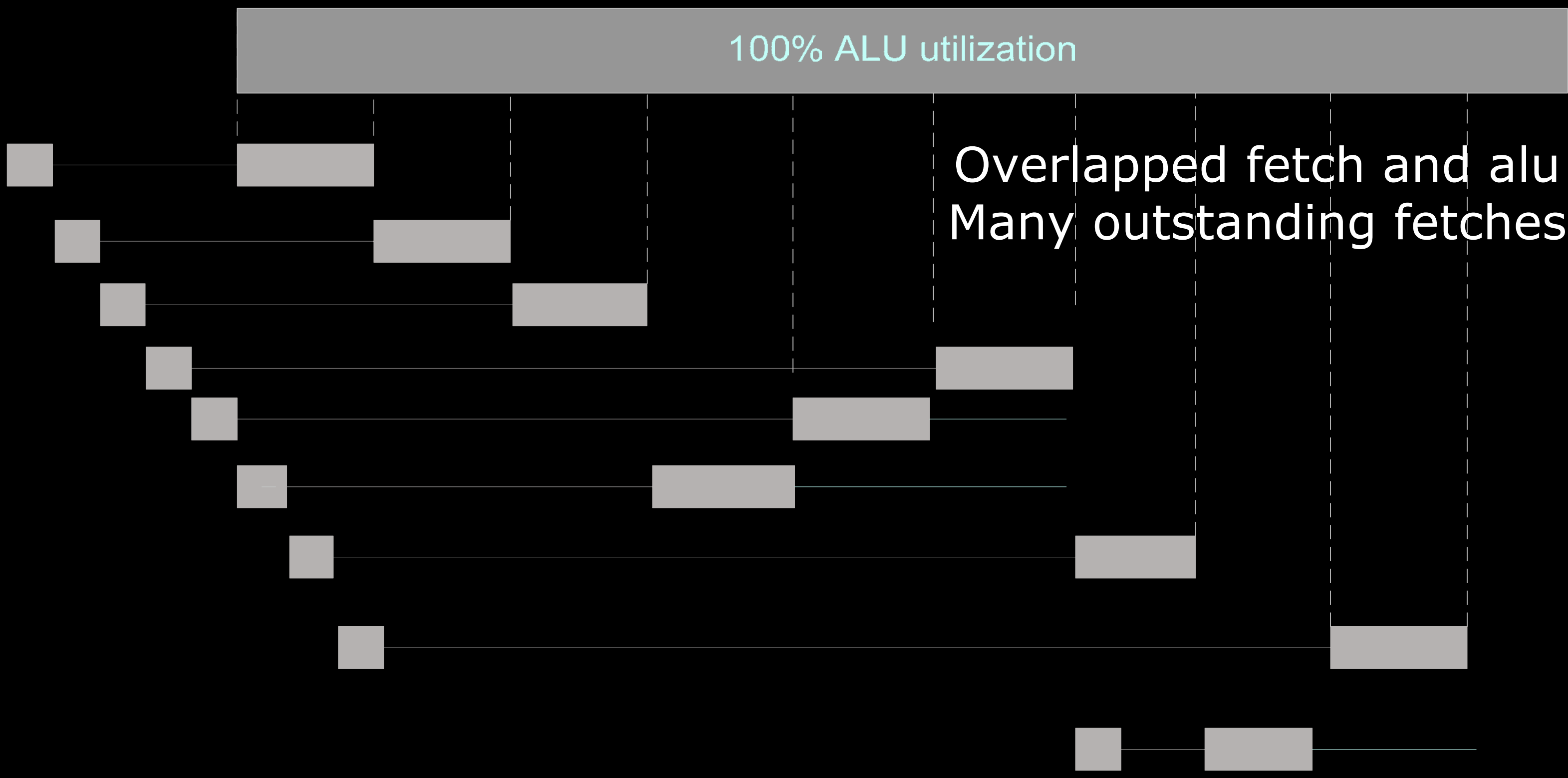
Typical CPU Operation

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

Hard to prefetch data
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)



ALU units reach 100% utilization
Hardware sync for final Output

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

RADEON 2900 Top Level

Red – Compute/

Fixed Function

Yellow – Cache

Unified shader

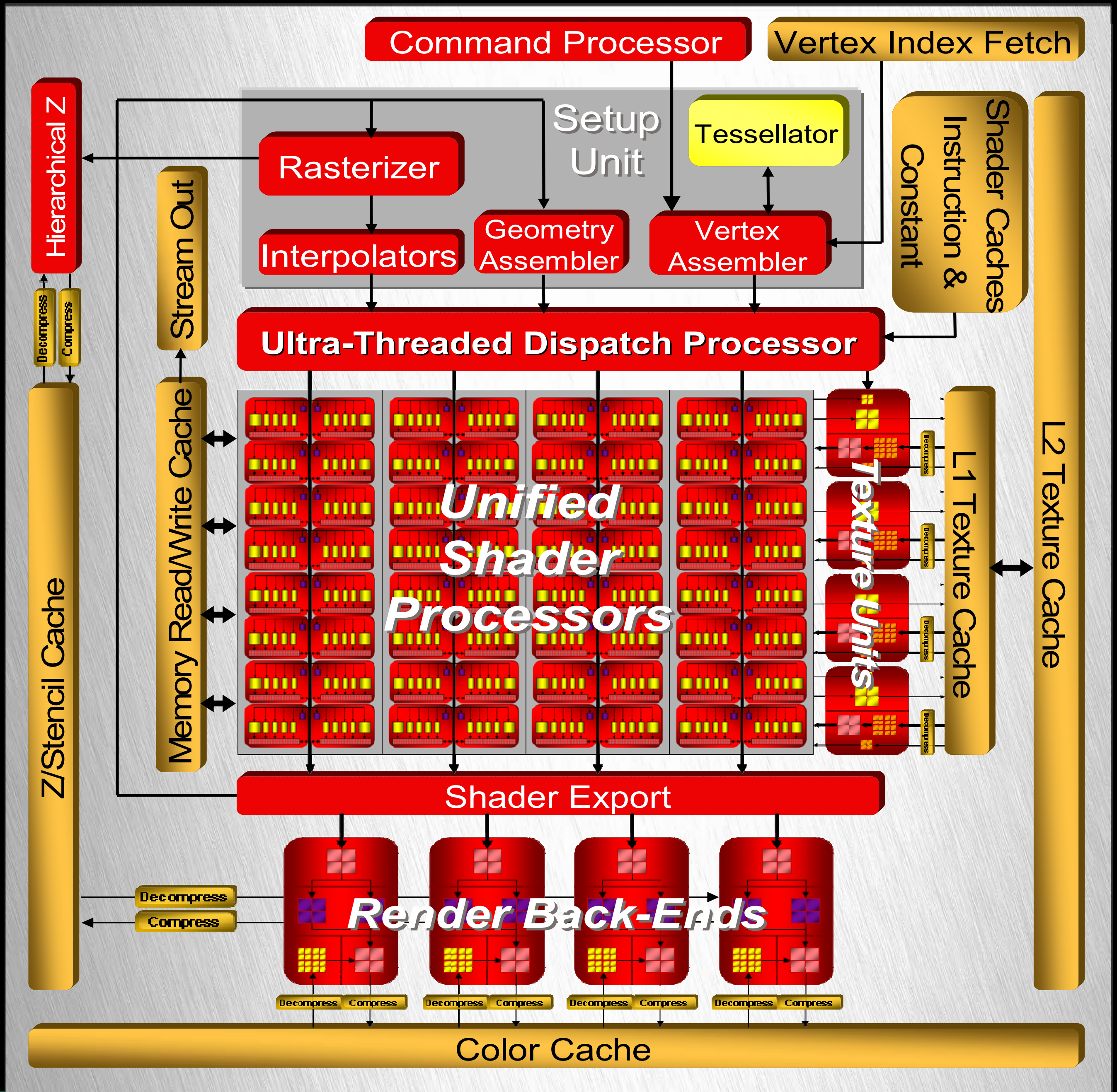
Shader R/W Cache

Instr./Const. cache

Unified texture cache

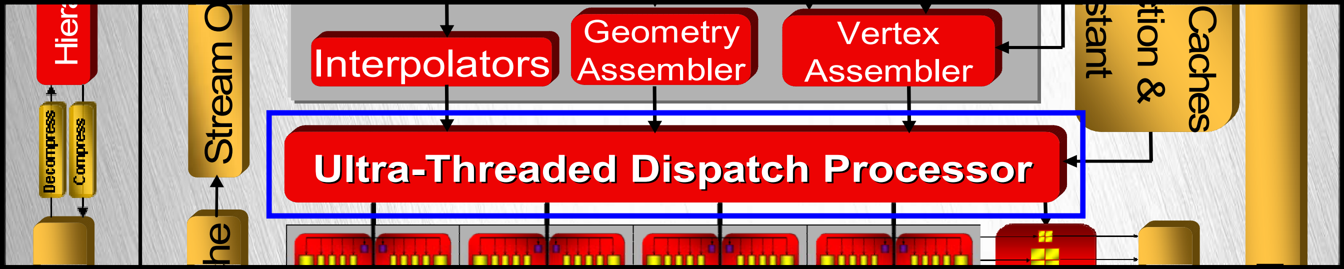
Compression

Tessellator

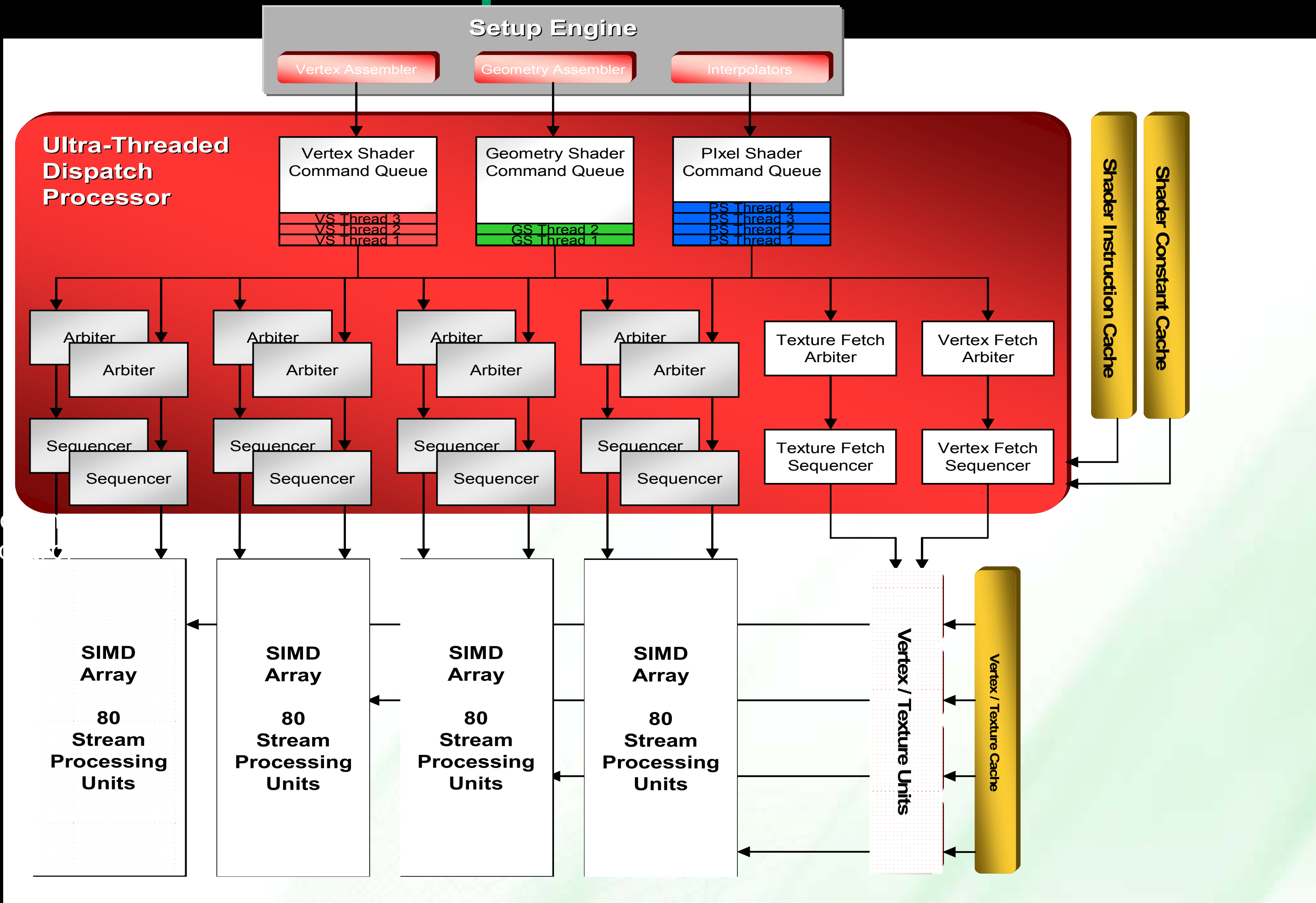


Ultra-Threaded Dispatch Processor/ Scheduler

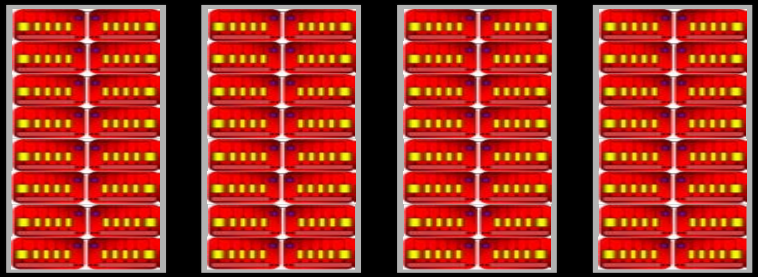
- Main control for the shader core
  - All workloads have threads of 64 elements
  - 100’s of threads in flight
  - Threads are put to sleep when they request a slow responding resource
- Arbitration policy
  - Age/Need/Availability
  - When in doubt favor pixels
  - Programmable



Ultra-Threaded Dispatch Processor



Shader Core



- 4 parallel SIMD units
- Each unit receives independent ALU instruction
- Very Long Instruction Word (VLIW)
- ALU Instruction (1 to 7 64-bit words)
  - 5 scalar ops – 64 bits for src/dst/cntrls/op
  - 2 additional for literal constant(s)

Stream Processing Units

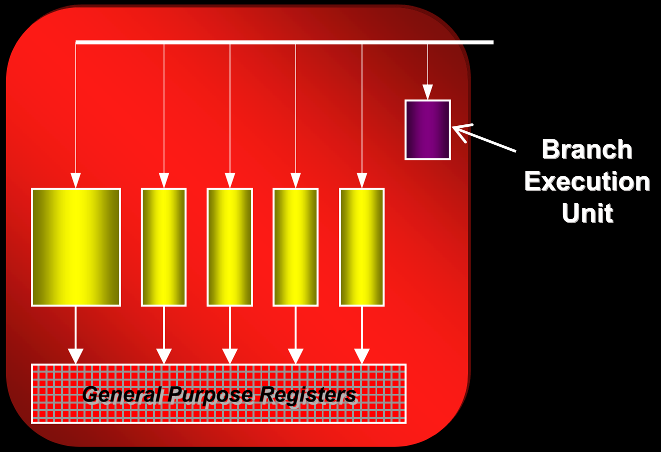
5 Scalar Units

- Each scalar unit does FP Multiply-Add (MAD) and integer operations
- One also handles transcendental instructions
- IEEE 32-bit floating point precision

Branch Execution Unit

- Flow control instructions

Up to 6 operations co-issued




Programming

Programming model

Vertex and pixel kernels (shaders)

Parallel loops are implicit

Performance aware code does not know how many cores or how many threads

All sorts of queues maintained under covers

All kinds of sync done implicitly

Programs are very small

Parallelism Model

All parallel operations are hidden via domain specific API calls

Developers write sequential code + kernels

Kernel operate on one vertex or pixel

Developers never deal with parallelism directly

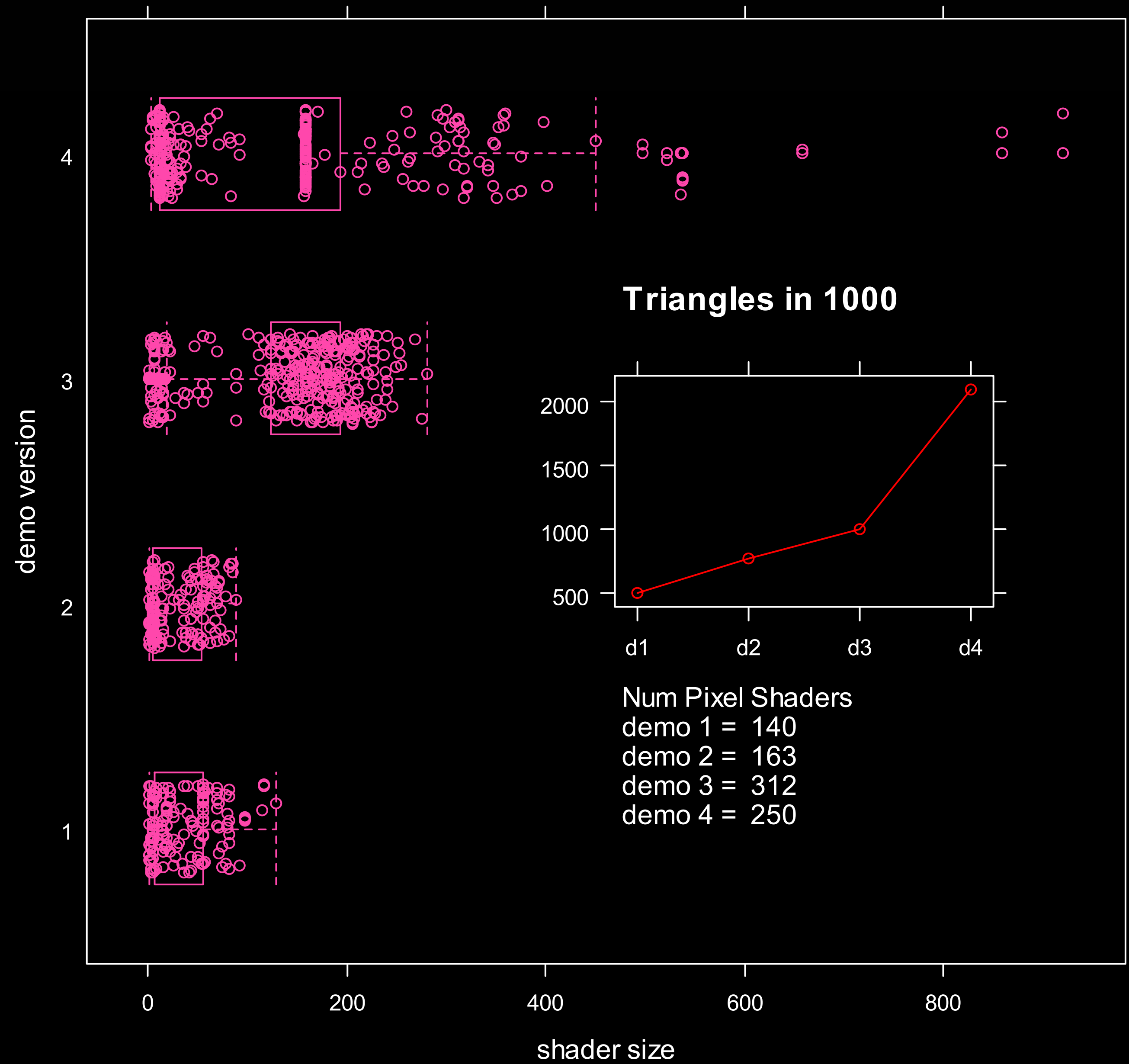
No need for auto parallel compilers

Ruby Demo Series

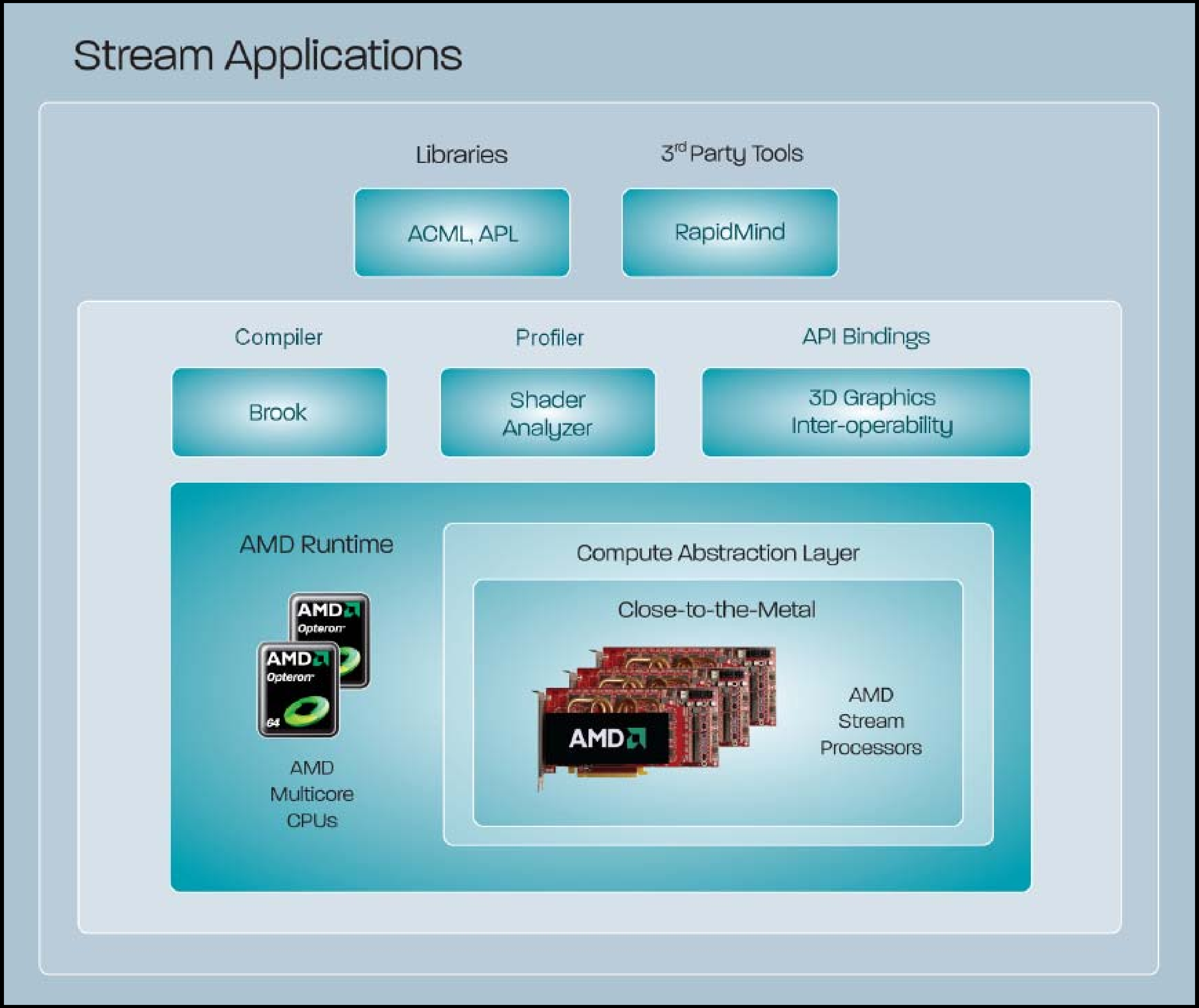
Four versions – each done by experts to show of features of the chip as well as develop novel forward-looking graphics techniques

First 3 written in DirectX9, fourth in DirectX10

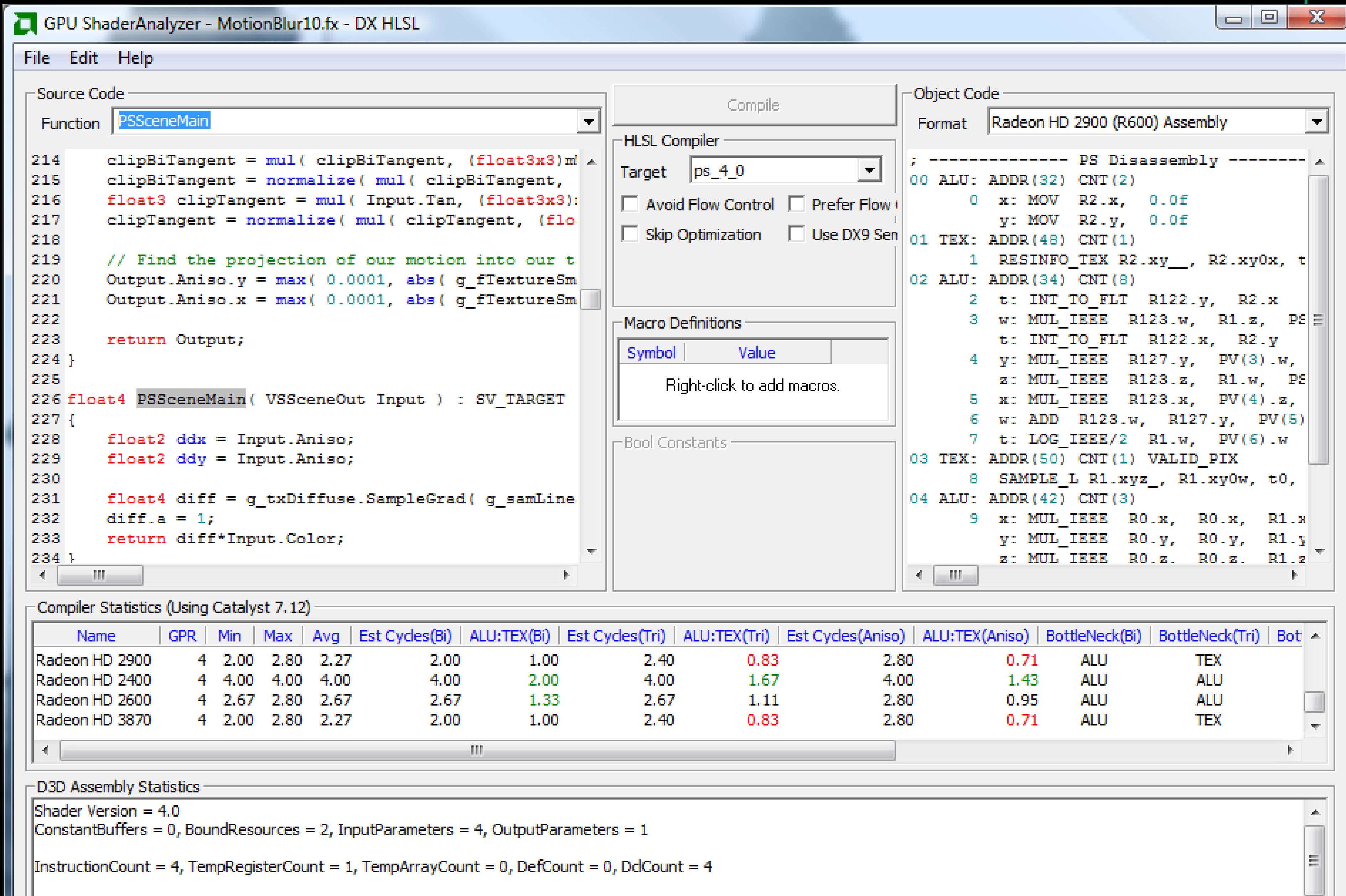
DX Pixel Shader Length



AMD Stream Computing



GPU ShaderAnalyzer



Architecture Challenges

GPU Directions

- Continued improvement and evolution in Graphics
  - RADEON 2900 Tessellator
- Continued conversion/removal of fixed function
  - Which fixed function ?
- Increasing importance of GPU as an accelerator
  - Generalization of GPU architecture
    - More aggressive GPU thread scheduler
    - Shader read-only via texture, write-only via color exports
    - Need new GPU shader architectures to meet much wider requirements
      - Already has a range of requirements across Graphics and Compute

Accelerated Computing

The Accelerated Computing Imperative

- Homogeneous multi-core becomes increasingly inadequate
- Targeted special purpose HW solutions can offer substantially better **power efficiency and throughput** than traditional microprocessors when operating on some of these new data types.
- Power constraints will force applications to be performance heterogeneous
  - Applications can target the HW device to get this power benefit
- GPUs - high power efficiency, more than 2 GFLOPs/watt
  - 20x > than dual-core CPU

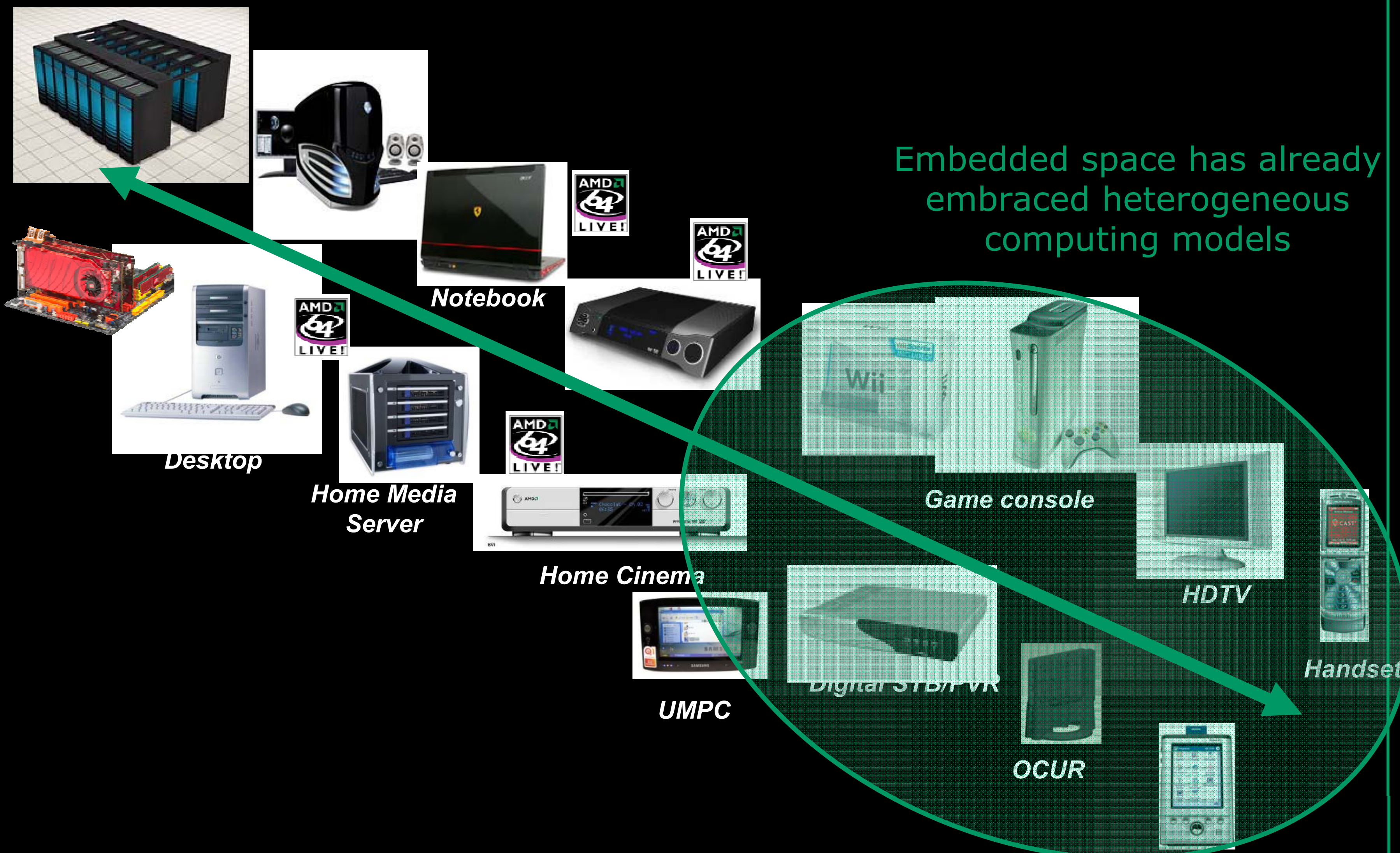
AMD's Accelerated Computing Initiative

- Discrete CPUs + GPUs
- Full Integration - Fusion - Mutli-core CPU + GPU Accelerator
- Compatibility will continue to be a key enabler in our industry
  - Need SW for new HW
    - How should existing Compute APIs evolve ?
    - Do we need new API models ?
    - GPU parallelism is so successful because graphics APIs are sequential
    - New APIs can’t tie down GPU progress
      - Need to replicate success of DX, need IHV input
    - Can only use GPGPU APIs when performance is necessary and programmer understands machine
  - Layers of computation
    - Compilers that can target workloads at appropriate processors

Future GPU/Accelerated Computing Applications

- Which applications benefit from combined CPU+GPU and how ?
- What type of architectural workload coupling between CPU+GPU do these applications require ?
- What are the new data and communication requirements ?
- What are the costs to existing performance to broaden what we do well ?

Form Factors for Accelerated Computing

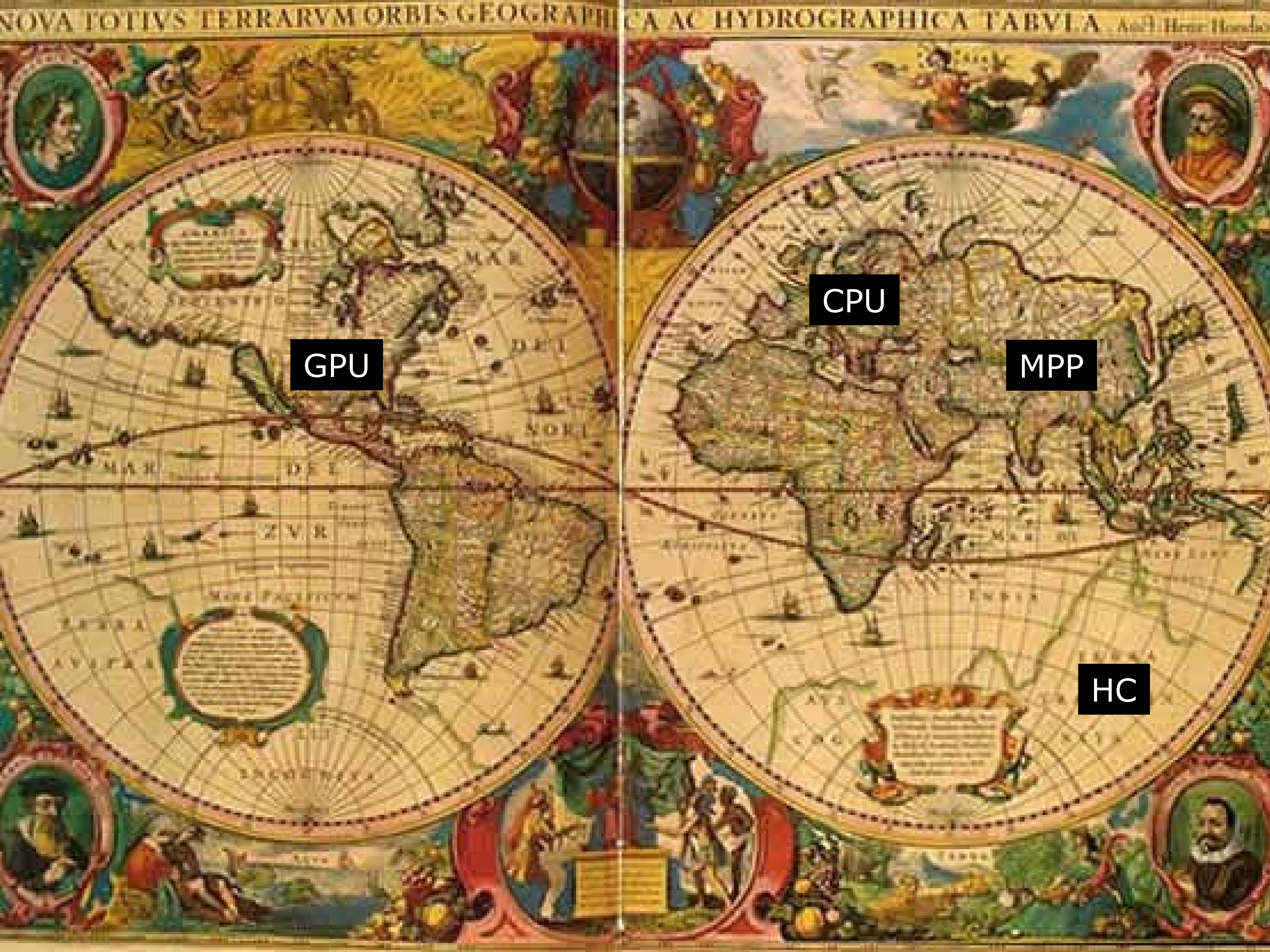


Accelerated Computing

- GPGPU APIs enable offloading compute to GPUs
  - Without heroic programming efforts
  - API compatibility enables
    - Hardware innovation without new SW
    - Improving performance on new HW without existing apps
- Broad range of possible accelerator designs
  - We need both CPUs and GPUs
    - Amdahl's law

Lots of Challenges …

- Managing context state and exceptions
  - This includes the program-visible state in the compute offload engine!
  - *Virtualizing* the context state
- Communications/Messaging
  - Simplified & fabric independent producer-consumer model
  - Optimized communications is a key enabler
  - *It's the synchronization, stupid*
- Memory BW and Data Movement
  - Keeping up with the computation rates will require increasingly capable memory systems
- New and appropriate APIs
  - Must offer a programming model that is actually easier than today's multi-core models
  - Use abstraction to trade some performance for programmer productivity
  - Live within bounds set by OS and Concurrent Runtimes


NOVA TOTIVS TERRARVM ORBIS GEOGRAPHICA AC HYDROGRAPHICA TABVLA
GPU
CPU
MPP
HC

Questions ?

Internships

ATI Fellowships

Michael.doggett ‘at’ amd.com

References

"Issues and Challenges in Compiling for Graphics Processors", Norm Rubin, Code Generation and Optimization 8 April, 2008

"The Role of Accelerated Computing in the Multi-Core Era", Chuck Moore, March 2008