



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



Architecture

Chip Design Focus Point

CPU

Lots of instructions little data

Out of order exec

Branch prediction

Reuse and locality

Task parallel

Needs OS

Complex sync

Latency machines

GPU

Few instructions lots of data

SIMD

Hardware threading

Little reuse

Data parallel

No OS

Simple sync

Throughput machines

Typical CPU Operation

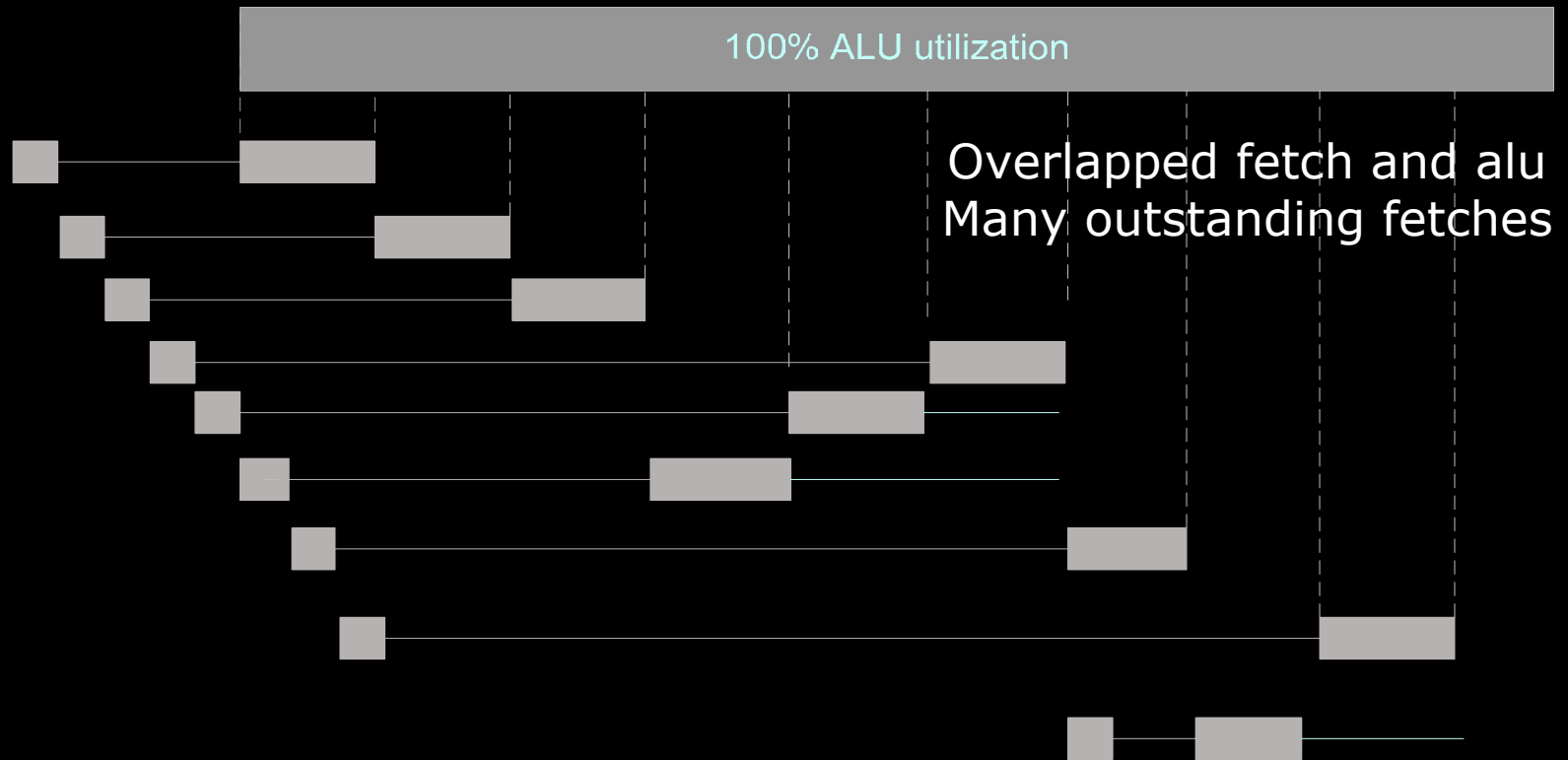
One iteration at a time
Single CPU unit
Cannot reach 100%

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

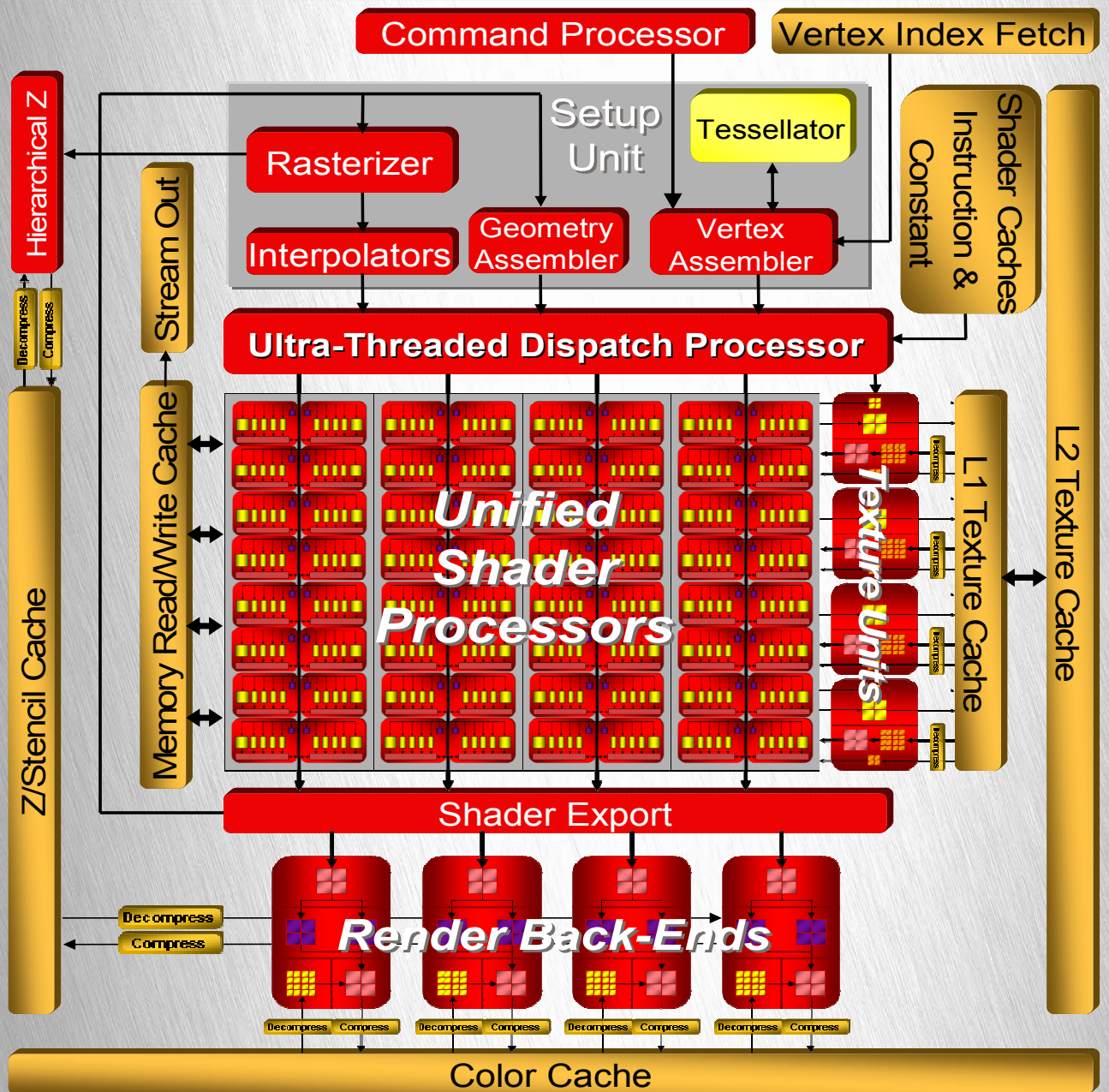
Shader R/W Cache

Instr./Const. cache

Unified texture cache

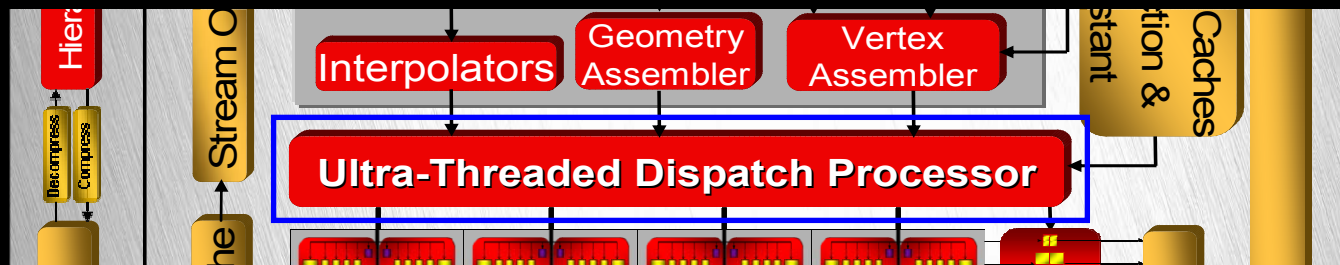
Compression

Tessellator



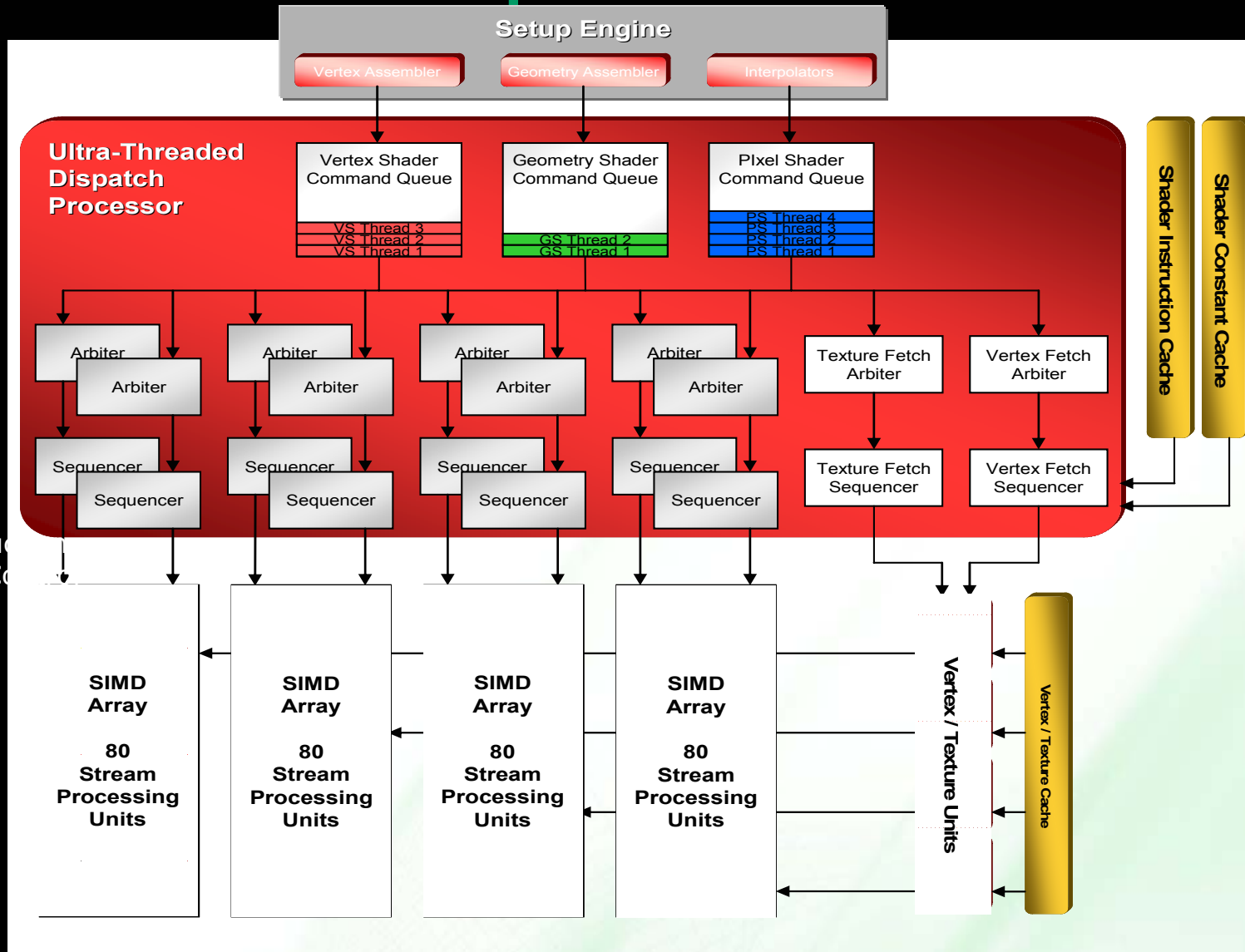
Ultra-Threaded Dispatch Processor/ Scheduler

- 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

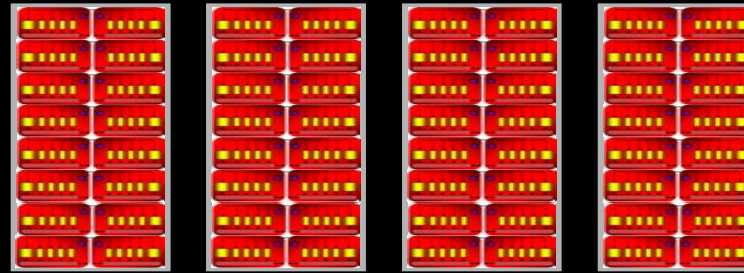


Ultra-Threaded Dispatch Processor

Instru
and Co



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

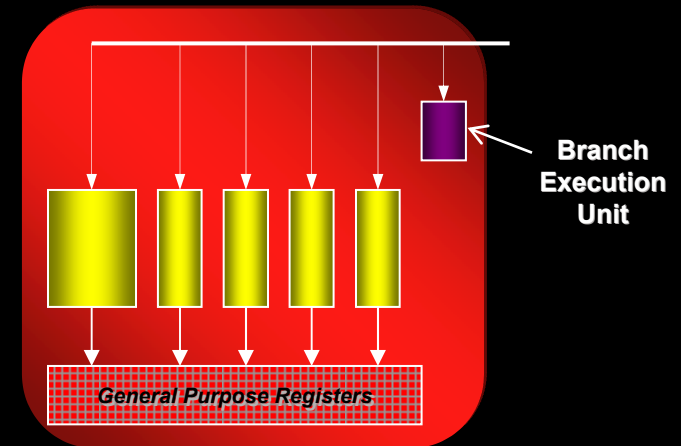
5 Scalar Units

- 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

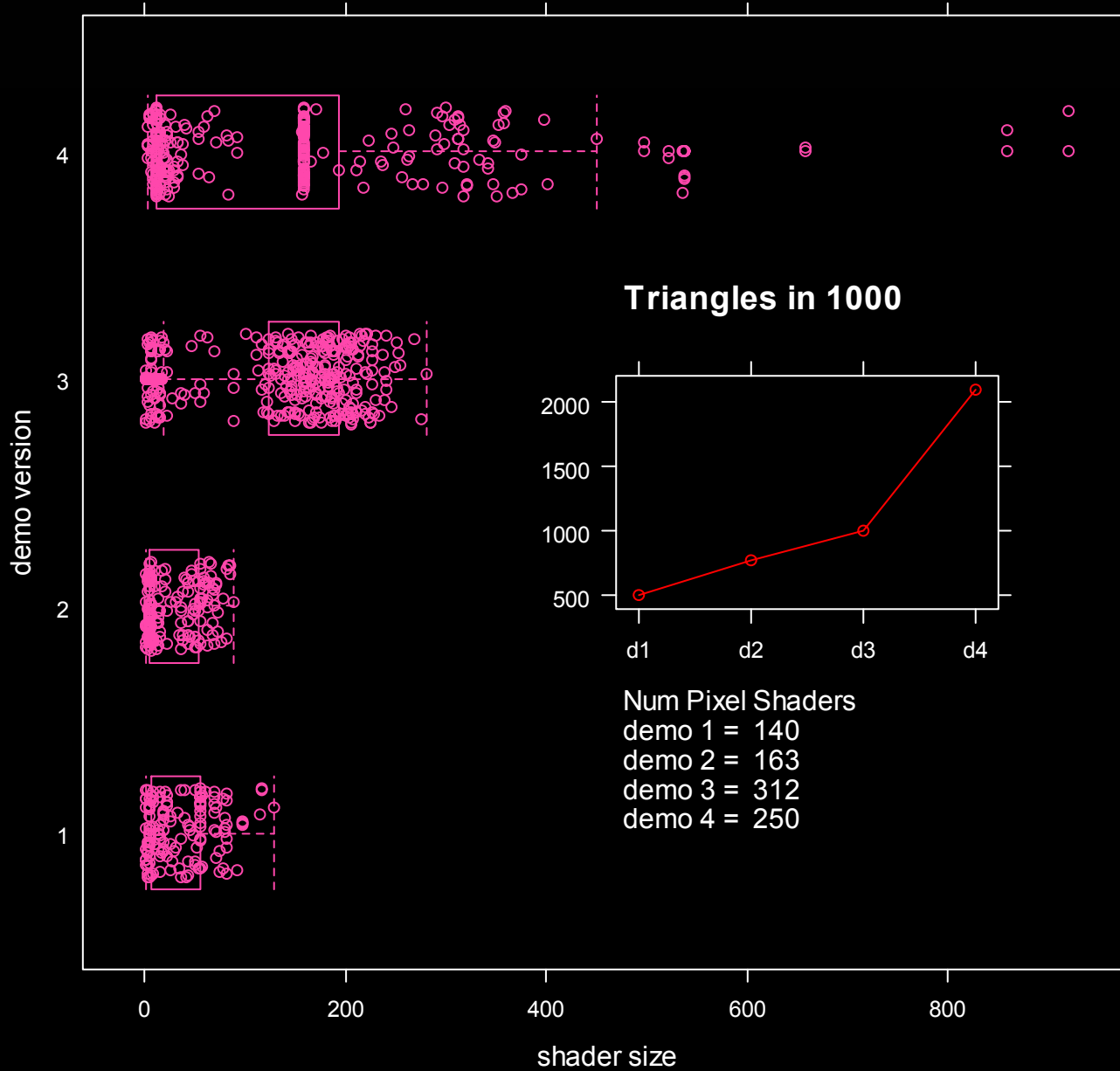
No need for auto parallel compilers

Ruby Demo Series

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

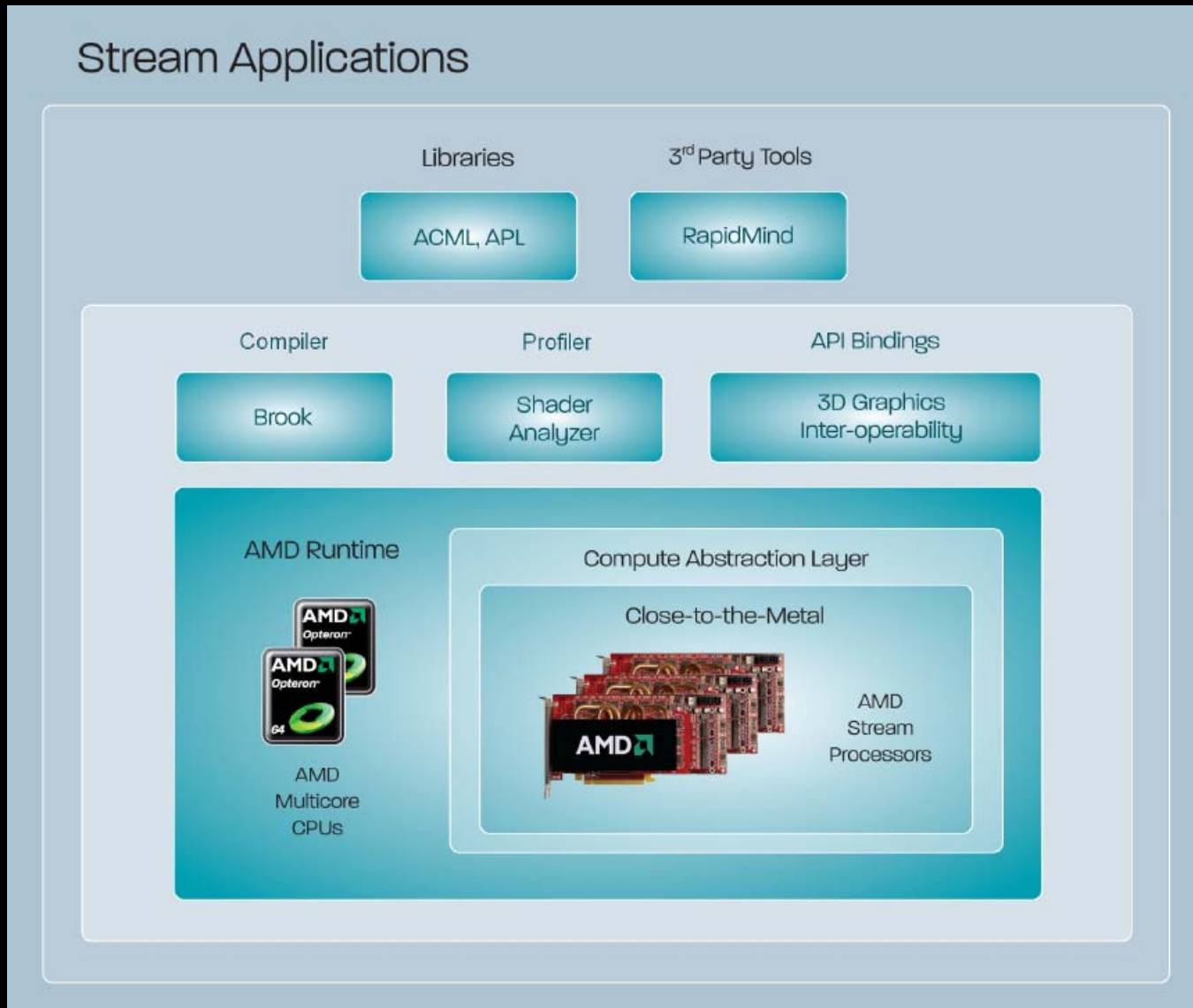
First 3 written in DirectX9, fourth in DirectX10

DX Pixel Shader Length



AMD Stream Computing

Stream Applications



GPU ShaderAnalyzer



GPU ShaderAnalyzer - MotionBlur10.fx - DX HLSL

File Edit Help

Source Code

```

Function PSSceneMain
214 clipBiTangent = mul( clipBiTangent, (float3x3)m
215 clipBiTangent = normalize( mul( clipBiTangent,
216 float3 clipTangent = mul( Input.Tan, (float3x3):
217 clipTangent = normalize( mul( clipTangent, (flo
218
219 // Find the projection of our motion into our t
220 Output.Aniso.y = max( 0.0001, abs( g_fTextureSm
221 Output.Aniso.x = max( 0.0001, abs( g_fTextureSm
222
223 return Output;
224 }
225
226 float4 PSSceneMain( VSSceneOut Input ) : SV_TARGET
227 {
228 float2 ddx = Input.Aniso;
229 float2 ddy = Input.Aniso;
230
231 float4 diff = g_txDiffuse.SampleGrad( g_samLine
232 diff.a = 1;
233 return diff*Input.Color;
234 }
    
```

Compile

HLSL Compiler

Target ps_4_0

Avoid Flow Control Prefer Flow Control

Skip Optimization Use DX9 Semantics

Macro Definitions

Symbol	Value
Right-click to add macros.	

Bool Constants

Object Code

Format Radeon HD 2900 (R600) Assembly

```

; ----- PS Disassembly -----
00 ALU: ADDR(32) CNT(2)
   0 x: MOV R2.x, 0.0f
   1 y: MOV R2.y, 0.0f
01 TEX: ADDR(48) CNT(1)
   1 RESINFO_TEX R2.xy_, R2.xy0x, t
02 ALU: ADDR(34) CNT(8)
   2 t: INT_TO_FLT R122.y, R2.x
   3 w: MUL_IEEE R123.w, R1.z, PS
   4 t: INT_TO_FLT R122.x, R2.y
   5 y: MUL_IEEE R127.y, PV(3).w,
   6 z: MUL_IEEE R123.z, R1.w, PS
   7 x: MUL_IEEE R123.x, PV(4).z,
   8 w: ADD R123.w, R127.y, PV(5)
   9 t: LOG_IEEE/2 R1.w, PV(6).w
03 TEX: ADDR(50) CNT(1) VALID_PIX
   8 SAMPLE_L R1.xyz_, R1.xy0w, t0,
04 ALU: ADDR(42) CNT(3)
   9 x: MUL_IEEE R0.x, R0.x, R1.x
   0 y: MUL_IEEE R0.y, R0.y, R1.y
   1 z: MUL_IEEE R0.z, R0.z, R1.z
    
```

Compiler Statistics (Using Catalyst 7.12)

Name	GPR	Min	Max	Avg	Est Cycles(Bi)	ALU:TEX(Bi)	Est Cycles(Tri)	ALU:TEX(Tri)	Est Cycles(Aniso)	ALU:TEX(Aniso)	BottleNeck(Bi)	BottleNeck(Tri)	Bot
Radeon HD 2900	4	2.00	2.80	2.27	2.00	1.00	2.40	0.83	2.80	0.71	ALU	TEX	
Radeon HD 2400	4	4.00	4.00	4.00	4.00	2.00	4.00	1.67	4.00	1.43	ALU	ALU	
Radeon HD 2600	4	2.67	2.80	2.67	2.67	1.33	2.67	1.11	2.80	0.95	ALU	ALU	
Radeon HD 3870	4	2.00	2.80	2.27	2.00	1.00	2.40	0.83	2.80	0.71	ALU	TEX	

D3D Assembly Statistics

Shader Version = 4.0
 ConstantBuffers = 0, BoundResources = 2, InputParameters = 4, OutputParameters = 1
 InstructionCount = 4, TempRegisterCount = 1, TempArrayCount = 0, DefCount = 0, DdCount = 4



08/09/1999 08:34:45 Auto Trigger
Resolution: 22.876 Hz; Span: 11.610 ms
(Fixed) Ch1 Amp1: -600.00 to 600.0 m
(Fixed) Ch2 Amp1: -600.00 to 600.0 m

AGENT RUBY

PIPELINE
START TIME: 10:00:00

CAMERA LIVE FEED

ARMED LIVE FEED

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

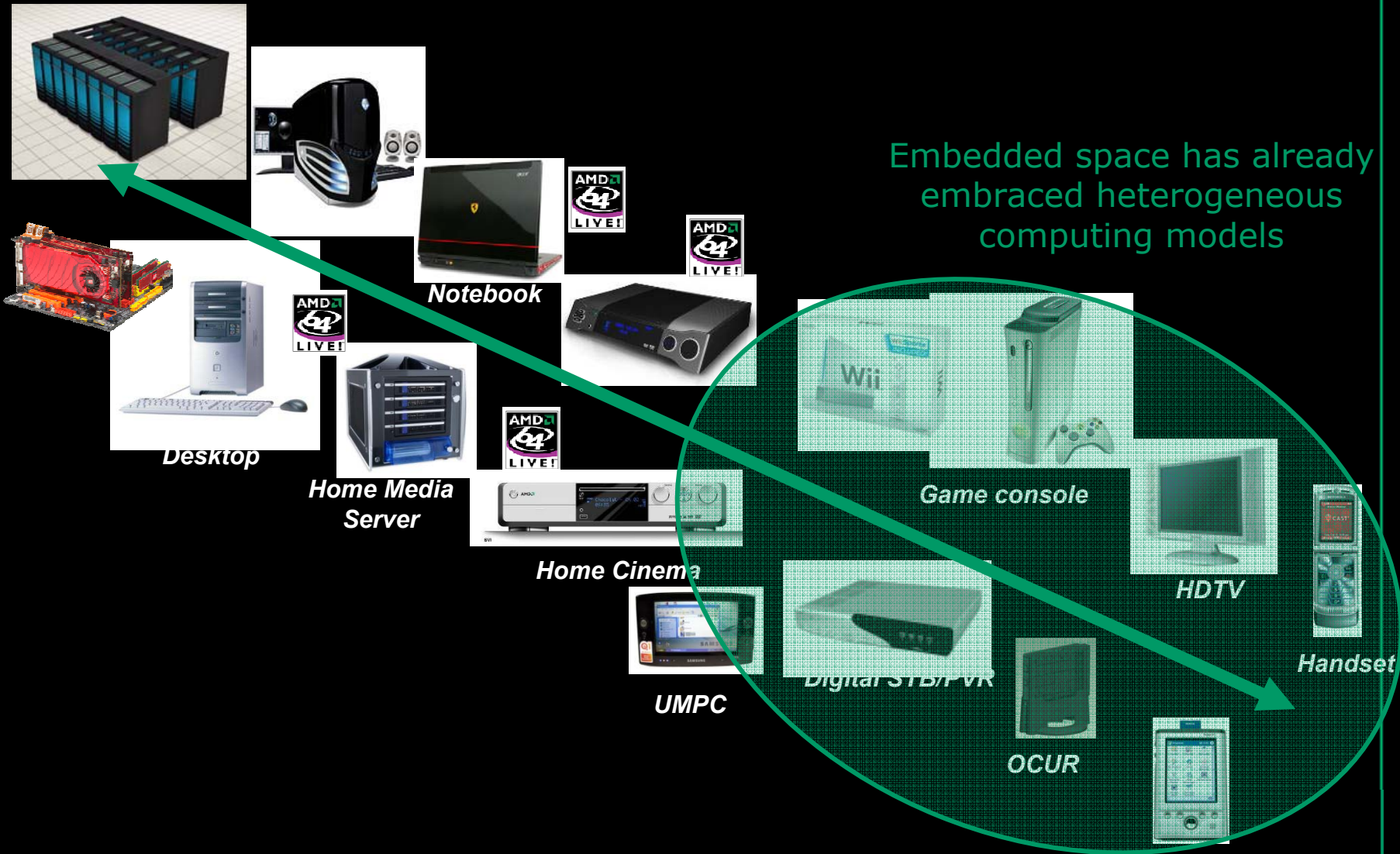
AMD's Accelerated Computing Initiative

- Discrete CPUs + GPUs
- Full Integration - Fusion - Multi-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

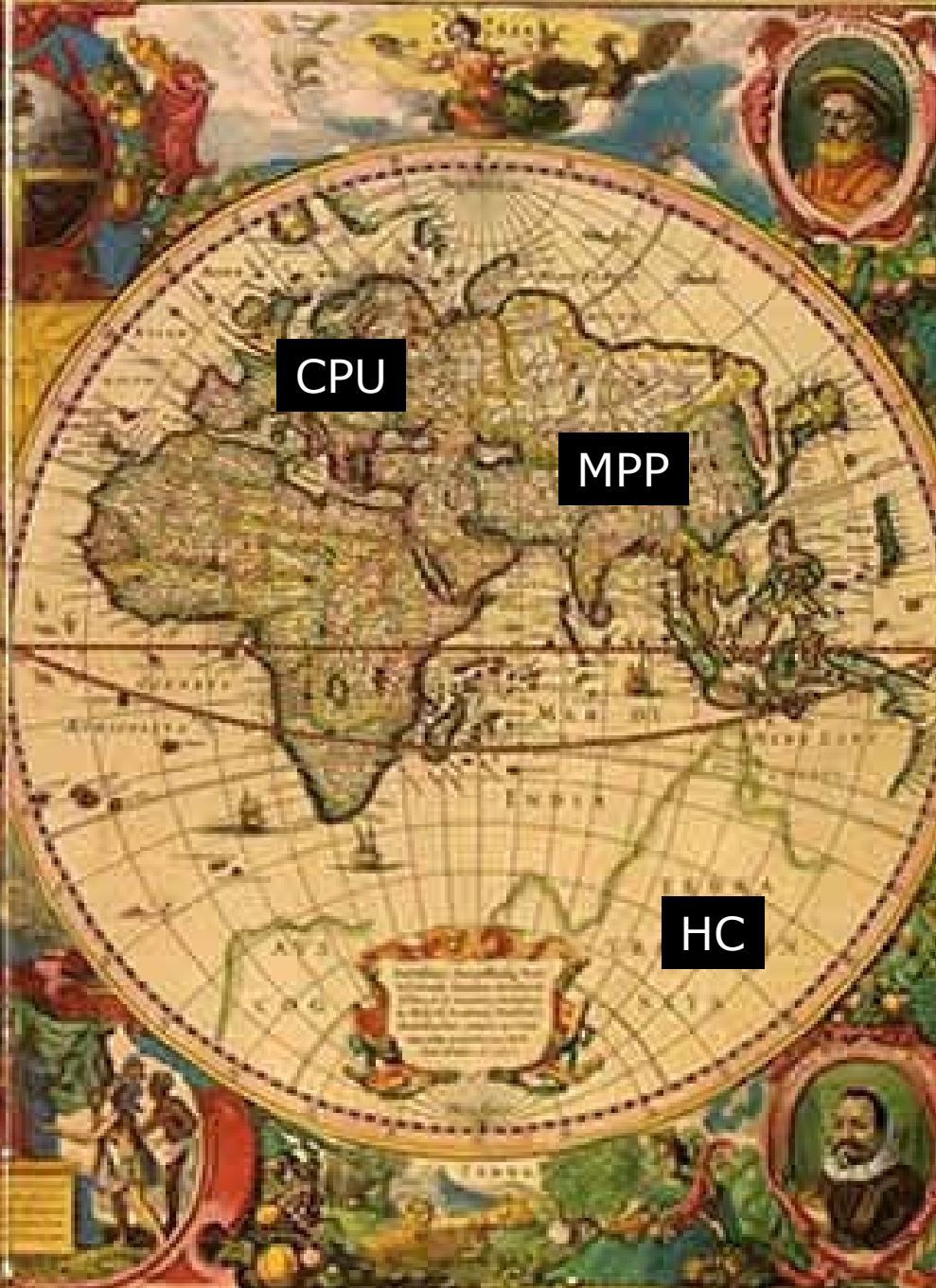
Lots of Challenges ...

- 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



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