2008 UCF GPGPU Class @ UCF

HD™ 2900
HD™ 3850 & 3870
HD™ 3870X2

Mike Mantor
Fellow
AMD Graphics Products Group
michael.mantor@amd.com

Agenda

- Introduction
- Overview of available GPUs from AMD

- Functional parts of a GPU device
- Data Flow
- Unified Shading, the core of current GPUs

- GPU -> GPGPU, Why!!
- What's next??

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)

Cutting-edge image quality features
- Advanced anti-aliasing and texture filtering capabilities
- Fast High Dynamic Range rendering
- Programmable Tessellation Unit

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

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

DirectX® 10
- Massive shader and geometry processing performance
- Shader Model 4.0 with Integer support
- Enabling the next generation of visual effects

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

AMD Radeon HD™ 3850 & HD™ 3870

- RV670 (Available Now)
- 320 Stream Processors
- ~512 GigaFLOPS of compute
- ~102 GFlops Double precision FP support
- Under $0.50 per GigaFLOP (including memory)
- >3 GFlops-per-watt
- 64 GB/sec memory bandwidth
- 192 Sqmm

– AMD FireStream™ 9170:

– Industry's First GPU with Double-Precision Floating Point

– AMD FireStream 9170 Specifications

– Features

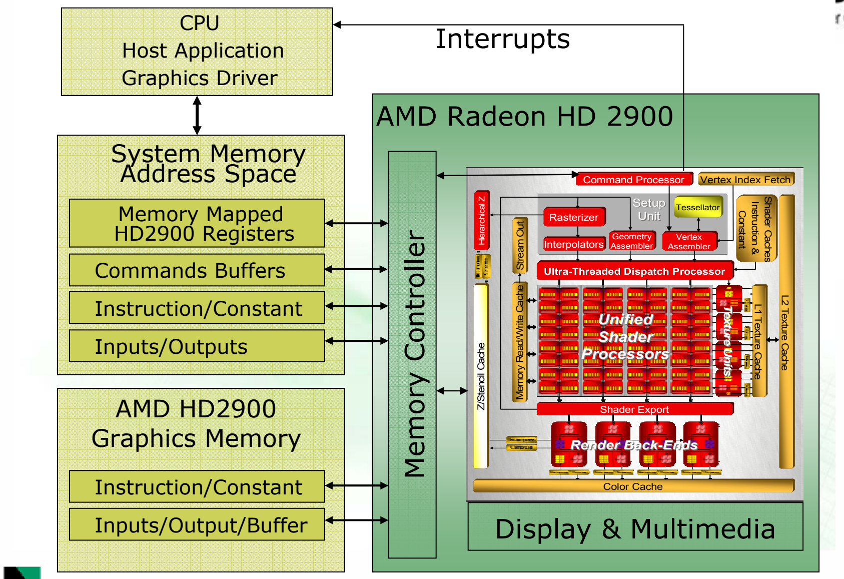
–Powered by next-generation ATI GPU from AMD
–Parallel processing architecture with 320 stream cores
–Up to 500 GFLOPs single precision performance
–2GB GDDR3 on-board memory
–Double Precision Floating Point
–PCIe 2.0 x16 interface
–< 150W power consumption
–Memory export
–BIOS settings optimized for stream processing
–API and OS Support
–Windows XP, XP64
–Linux 32 and Linux 64

AMD’s twin –GPU Radeon HD™ 3870X2

Radeon HD™ 3870X2

– Core:	R680 (2x RV670)
– Manufacture Process:	55nm
– Transistor Count:	~1333 million
– Shaders:	640
– Core Clcok	825 MHz
– Memory Clock	900Mhz
– Memory Interface	256 bit(x2)
– Memory Type	GDDR3
– Memory Size	1024MB
– Math Rate	>1 teraflop SPF
– Interface	PCI Express 2.0
– Support	DirectX 10.1
	Shader Model 4.1

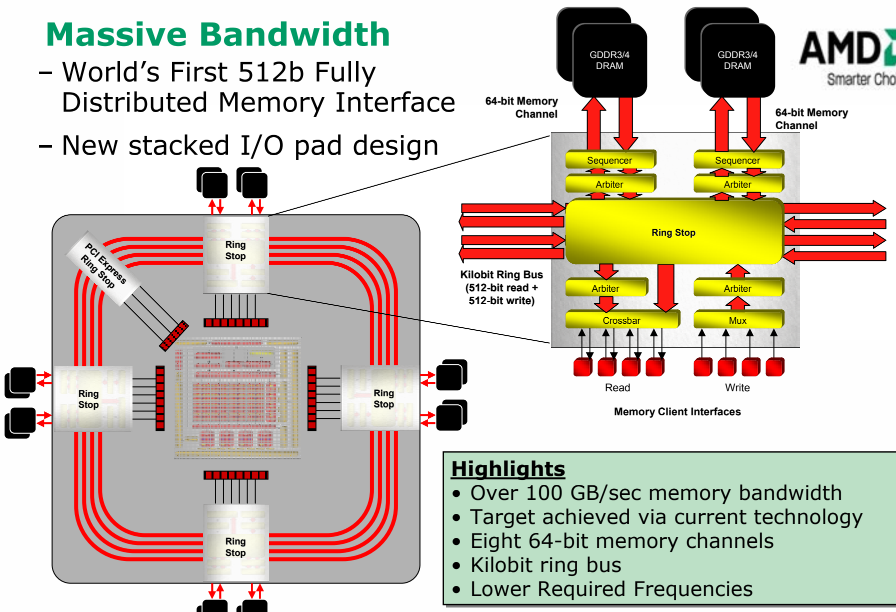
AMD Radeon HD2900 Graphics System



Massive Bandwidth

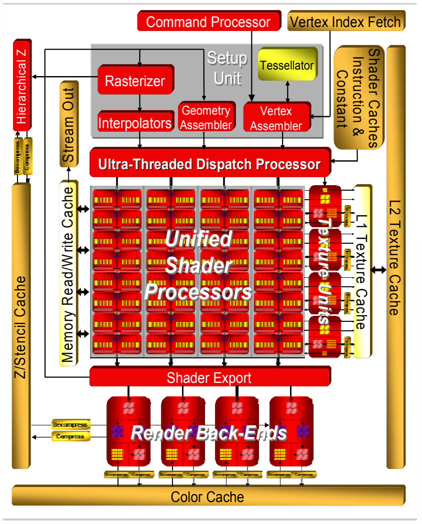
– World's First 512b Fully Distributed Memory Interface

– New stacked I/O pad design



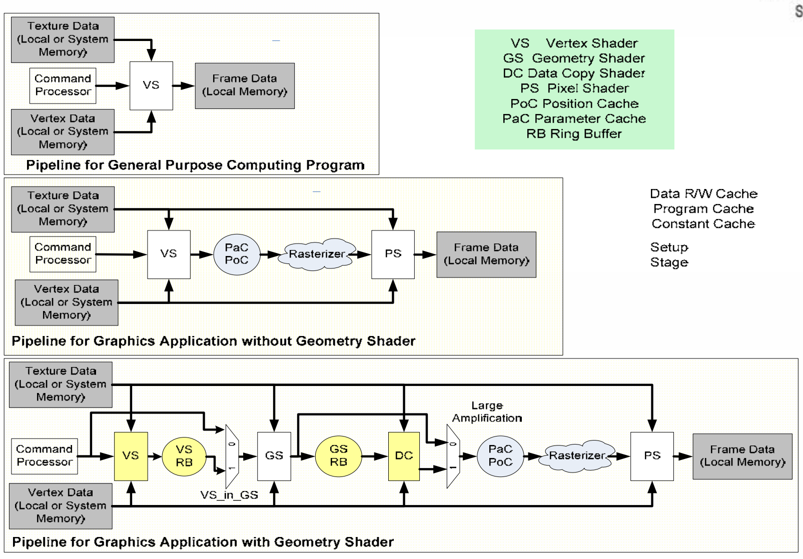
Highlights
- Over 100 GB/sec memory bandwidth
- Target achieved via current technology
- Eight 64-bit memory channels
- Kilobit ring bus
- Lower Required Frequencies

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
- Scalar 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

Programmer's View of Shader Dataflow



Texture Data (Local or System Memory)
Command Processor
Vertex Data (Local or System Memory)
VS
Frame Data (Local Memory)
Pipeline for General Purpose Computing Program
Texture Data (Local or System Memory)
Command Processor
Vertex Data (Local or System Memory)
VS
PaC PoC
Rasterizer
PS
Frame Data (Local Memory)
Pipeline for Graphics Application without Geometry Shader
Texture Data (Local or System Memory)
Command Processor
Vertex Data (Local or System Memory)
VS
VS RB
0
1
VS_in_GS
GS
GS RB
DC
Large Amplification
0
1
PaC PoC
Rasterizer
PS
Frame Data (Local Memory)
Pipeline for Graphics Application with Geometry Shader

VS Vertex Shader
GS Geometry Shader
DC Data Copy Shader
PS Pixel Shader
PoC Position Cache
PaC Parameter Cache
RB Ring Buffer

Data R/W Cache
Program Cache
Constant Cache

Setup
Stage

Types of Parallelism Exploited

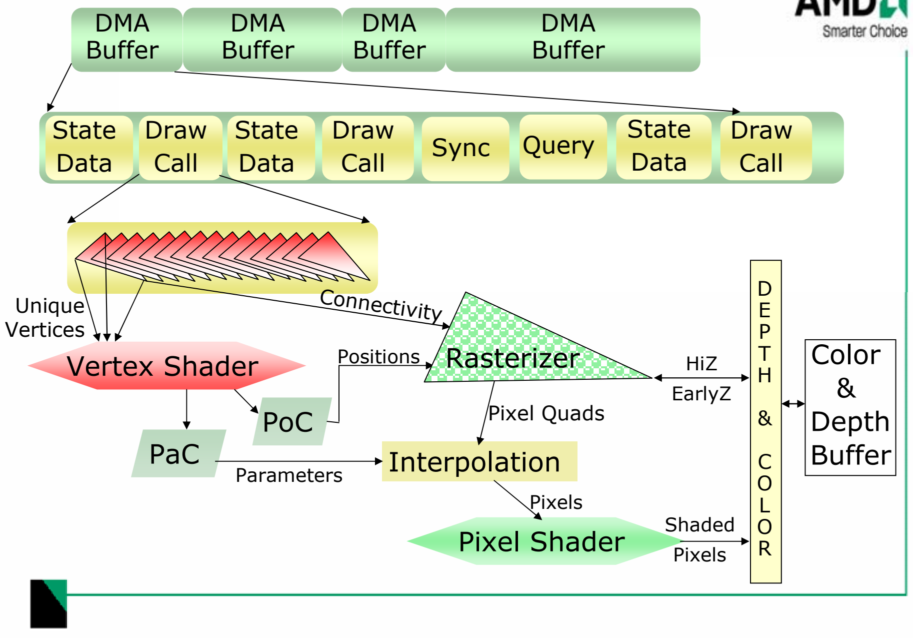
- **Data Level Parallelism**
- Multiple Data Sets: Array of Vertices, Pixels, Primitives, Data Records
- Multiple invocations/instance of a Shader
- Multiple Shader types (i.e. Unified Shaders)

- **Instruction Level Parallelism**
- Compiler based thread/trace scheduling
- Hardware: 1Macc per element + 1Macc Transcendental
- Within Instruction – 5 scalars co-issue

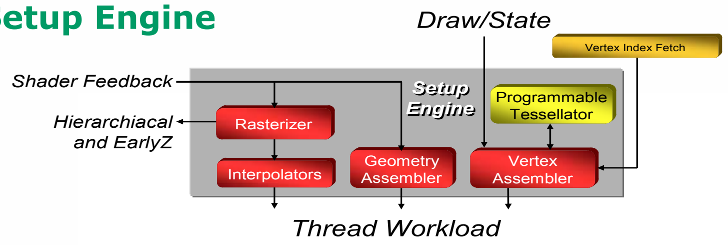
Command Processor

- GPU interface with host
- A custom RISC based Micro-Coded engine
- Memory & Register Read and Write access
- Multiple buffers with dependant fetch latency hiding
- Surface coherency synchronization
- Host interrupt notification system
- Hardware based validation of state data at draw call

Graphics Pipeline Data Flow without GS



Setup Engine

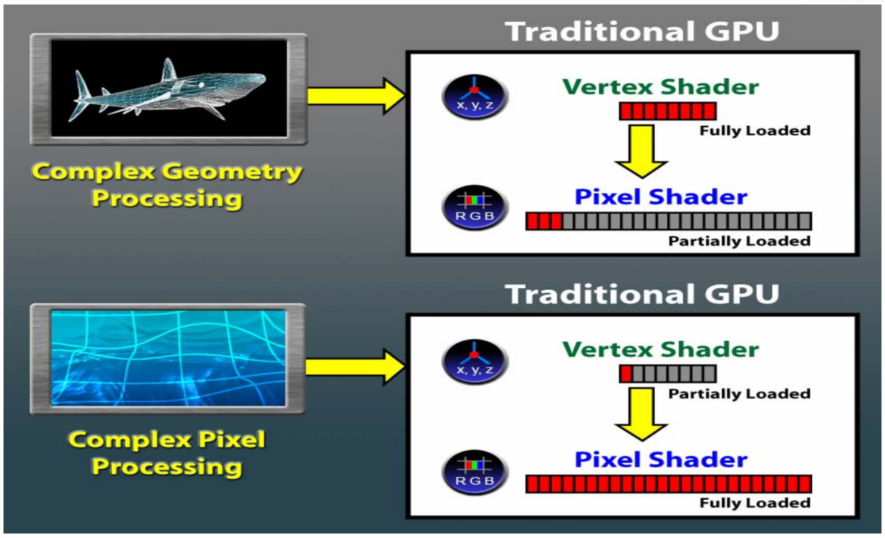


Workload preparation for Shader

- Staging to collect data for submission
  - Different arrival/drain rates
  - Different storage requirements
  - Different processing needs
- Submittal Arbitration policies
  - Output Need feedback/Availability/Balance
  - Prevent over-subscription
  - when in doubt favor pixels

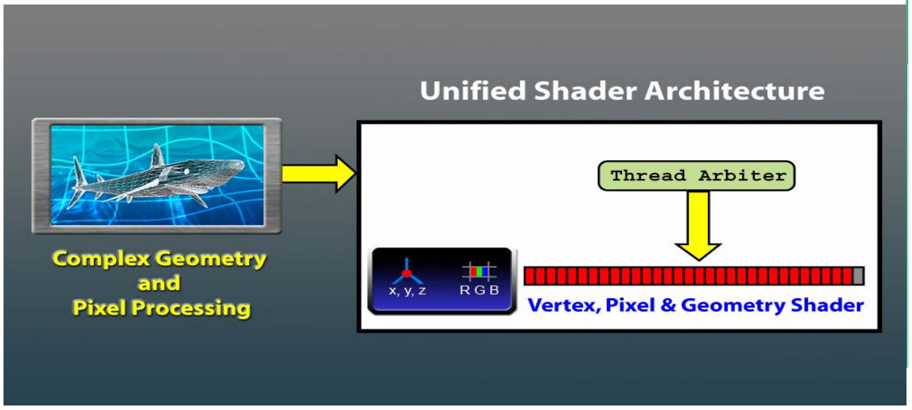
- **Vertex workload**
  - Primitive Assembly & Vertex Reuse
  - Primitive Tessellation (742Mtri/sec)
  - Inputs – Index & Instancing Data
- **Geometry Shader Staging**
  - On\off chip staging
  - Amplification and parallelism
    - Dependence on SIMD size
- **Pixel Shader Staging**
  - Rasterization and Interpolation
  - Vertex/Pixel I/O mappings
  - Inputs- System variables, z, center, centroid, sample, linear

Motivations for Unified Shader



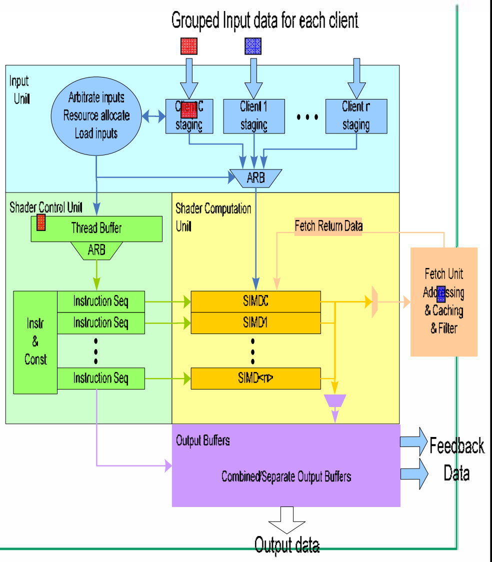
Unified Shader

- Scene varies in shader type and resources are dynamically shared
- Resources allocated and distributed to balance workloads
- Parts of a frame requiring one shader type will have access to more resources
- Addresses internal frame changing workloads as well as application or market
- Resource scaling decoupled from workload clients


Unified Shader Architecture
Complex Geometry and Pixel Processing
Thread Arbiter
x, y, z
RGB
Vertex, Pixel & Geometry Shader

Basic Unified Shader Model

- Basic Unified Shader System
  - Input Unit
    - Staging storage and assembly
    - Input arbitration & resource allocation
  - Shader Computation Unit
    - SIMDs (Groups of Pipes)
    - Pipes (Groups of GPR/ALU)
    - General Purpose Register (GPR)
    - Arithmetic Logic Unit (ALU)
  - Shader Control Unit
    - Thread Buffer - Scheduling
    - Instruction/Constant Store
    - Arbitration/Instruction SEQ
  - Fetch Units
    - Process fetch request
    - Provides Address calculations
    - Data Caching, fetch, return
  - Output Buffers
    - Shared or exclusive buffers
    - Output results for each clients
- Ideal execution – minimize latency & storage
  - Oldest thread of most in demand type whenever ready
  - Always fill gaps with next oldest in that type when ready
  - Minimize a threads lifetime in shader



Design Goals for Unified Shader

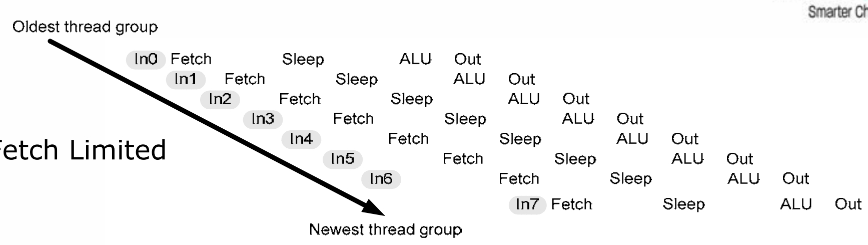
- Maximize Performance via ALU utilization
- Provide shared resources for all shader types
- Sustain peak Fetch and I/O Rates
- Provide a common programming language
- Simplify design and verification process
- Enable common tool chain
- Flexibility and Scalability

Requirements to meet goals

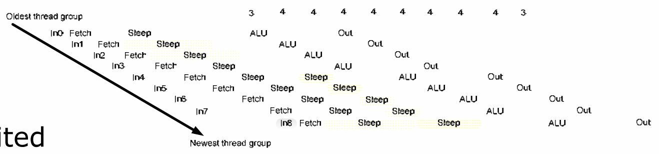
- Hide latency of memory fetches
- Create cache locality to prevent over-fetch
- Prevent resource over subscription
- Arbitrate on age/need to protect bandwidth
- Enable performance scaling for all workloads
- Provide ALU & I/O Ratios for typical workloads
- Interleaved diverse workloads to balance

Simultaneous Multi-Threaded Engine

Fetch Limited



ALU Limited



Unified Shader\Stream Processors

AMD Smarter Choice

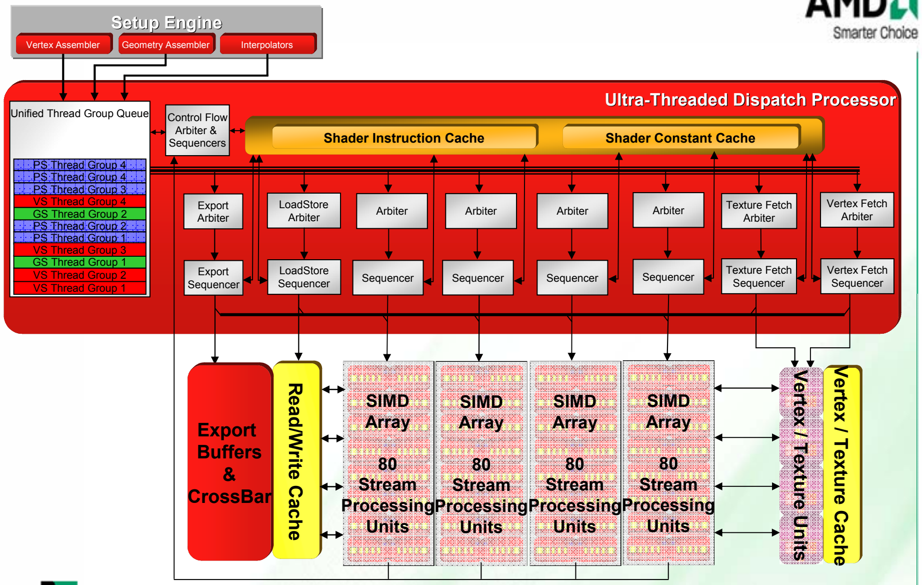
- **S**ingle **I**nstruction **M**ultiple **D**ata
  - Each SIMD receives independent ALU instruction stream
  - Each SIMD applies instruction stream to multiple data elements
- **M**ultiple **I**nstruction **M**ultiple **D**ata
  - Multiple SIMD units operating in parallel (Multi-Processor System)
  - Distributed or shared memory
- **V**ery **L**ong **I**nstruction **W**ord (VLIW) design
  - Co-issued up to 6 operations (5 ALU + 1 FC)
  - 1.25 Machine Scalar operation per clock for each of 64 data elements
  - Independent scalar source and destination addressing
- Simultaneous Instruction Issue
  - Input, Output, Fetch, ALU, and Control Flow per SIMD

Shader Instructions

AMD Smarter Choice

- **VLIW** (Very Long Instruction Word), variable length
- Control Flow Instructions
  - Control branch, loop, stack operations
  - Clause launch
  - Barriers, Allocation, and Exports
- Clause – Set of instructions that executes w/o pre-emption
  - ALU Instructions
  - Texture & Vertex Fetch Instructions
  - Memory Read/Write Instructions
- ALU Instruction (1 to 7 64-bit words)
  - 5 scalar ops – 64 bits each
  - 2 additional words for literal constants

Ultra-Threaded Dispatch Processor



Shader Processing Units (SPU)

Arranged as 5-way scalar stream processors

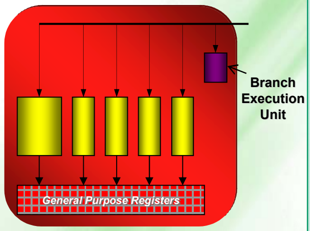
- Co-issue up to 5 scalar FP MAD (Multiply-Add)
- Up to 5 integer operations supported (cmp, logical, add)
- One of the 5 stream processing units additionally handles
  - * transcendental instructions (SIN, COS, LOG, EXP, RCP, RSQ)
  - * integer multiply and shift operations
- 32-bit floating point precision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

Branch execution units handle flow control and conditional operations

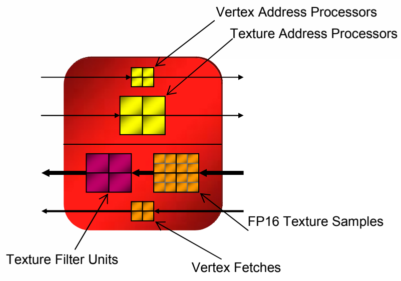
- Condition code generation for full branching
- Predication supported directly in ALU

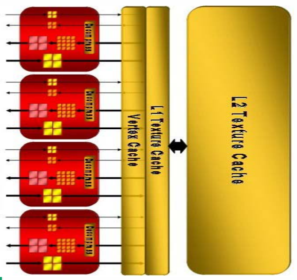
General Purpose Registers

- 1 MByte of GPR space for fast register access



Fetch Unit Design





Fetch units

- Fetch Address Processors each
  - 4 filtered (fetch neighboring data for filtering)
  - 4 un-filtered raw data fetch
- 20 Samples accessed from cache per clock
- 4 bilinear filter results per clock (with BW)
  - Filter rate for each pixel:

    one 64-bit FP texture result per clock,

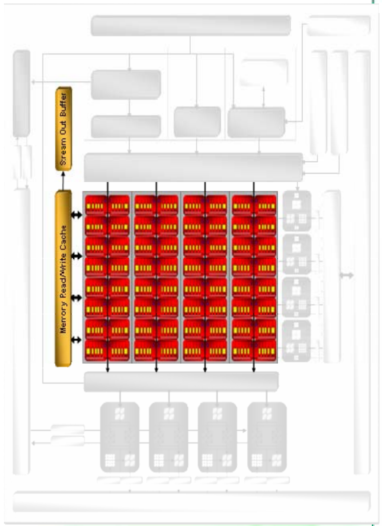
    one 128-bit FP result per 2 clocks

Multi-level fetch cache design

- L2/L1 cache structures
  - Unified 4kb L1 structured cache (unfiltered)
  - Unified 32kb L2 structure cache (unfiltered)
  - Unified 32k L1 texture cache
  - Unified 256KB L2 texture cache

Memory Read/Write Cache

- Virtualizes register space
  - Allows overflow to graphics memory
  - Can be read from or written to by any SIMD (fetch caches are read-only)
  - Can export data to stream out buffer
  - 8KB Fully associative cache, write combining
- Stream Out
  - Allows shader output to bypass render back-ends and color buffer
  - Outputs sequential stream of data instead of bitmaps
- Uses include:
  - Inter-thread communication
  - Render to vertex buffer
  - Overflow storage/output for Geometry Shader data (allowing parallel processing for large amplification)



Render Back-Ends

Alpha testing, Alpha and fog blending

Double rate depth/stencil test

- 32 pixels per clock for ATI Radeon HD 2900

Multi-Sample Anti-Aliasing (MSAA) resolve functionality is programmable

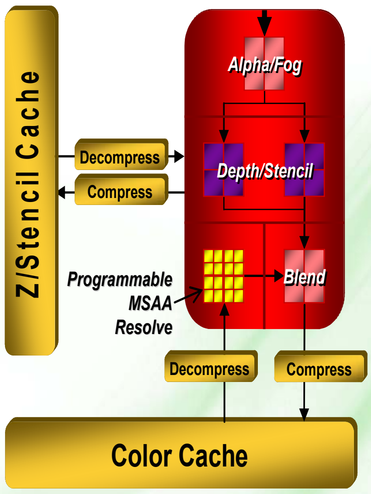
- Makes Custom Anti-Aliasing Filter possible

New blend-able surface formats

- Allows new DirectX10 formats to be displayable
  - 128-bit floating point format
  - 11:11:10 floating point format

MRT (Multiple Render Target) support

- Up to 8 MRTs with MSAA support



A Scalable Family

ATI RadeonHD 2900
320 Stream Processors
4 SIMDs
4 Texture Units
4 Render Back-End

ATI Radeon™HD 2600
120 Stream Processing
3 SIMDs
2 Texture Units
1 Render Back-End

ATI Radeon™HD 2400
40 Stream Processing
2 SIMDs
1 Texture Unit
1 Render Back-End
Shared vertex/texture cache

- Designed with a "numbers of" for most elements
- Shader, Texture, Interpolate, Raster Backend Units
- Core functionality exists in all parts
- Target specific cost/performance levels for each part

GPU Processing Implications

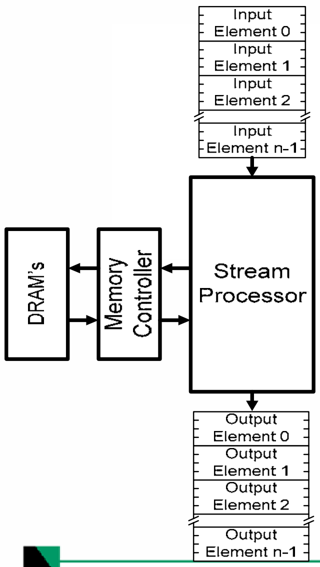
- Compute is cheap but you need lots of parallelism to keep all those GPU alu's busy. (graphics shading is highly parallel)
- Compute goes up by 70% a year but bandwidth goes up by 25% a year, latency goes down by 5% a year (arithmetic intensity – lots of alu ops per read)
- GPU wins when arithmetic intensity is high
- GPU wins when streaming (little reuse – lots of data)

Elements of good streams processor

Programming Model

- Familiar programming tools and interfaces
- Ease of use and full control
- Compilers, assemblers, libraries and middleware
  - No one size fits all - need all of these
- Documented and open interfaces

The stream computing model



- Apply the same function to "n" data elements.
  - For GPU's, a data element is typically a vertex or pixel.
- No communication or synchronization between elements*.
- Optimal performance requiring "n" to be very large (many hundreds or thousands).

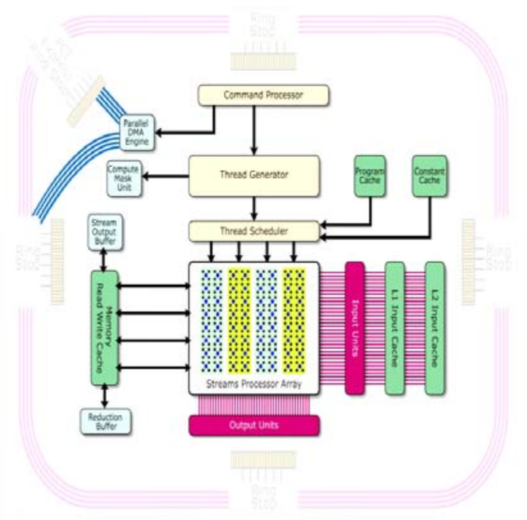
*Recent work has shown some value for certain applications (such as FFT and convolution) in breaking the stream computing model by allowing limited communication between elements

Workload Differences

General Processing	Stream Processing
• Small batches	• Large batches
• Frequent branches	• Few branches
• Many data inter-dependencies	• Few data inter-dependencies
• Scalar ops	• Scalar ops
• Vector ops	• Vector ops

ATI Radeon HD 2000 Stream Compute Architecture

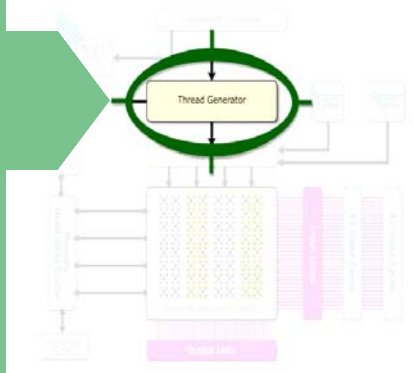
AMD Smarter Choice



ATI Radeon HD 2000 Stream Compute Architecture

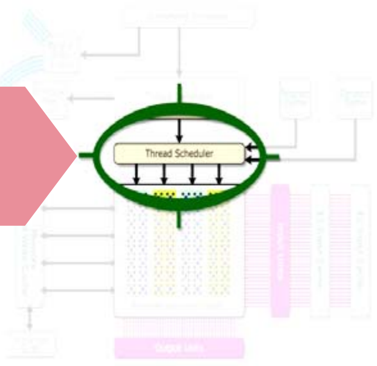
AMD Smarter Choice

- Both low-latency and high-throughput thread generation
- Low latency threads for interactive compute applications
- (Physics, AI etc)
- High-throughput threads for large compute tasks
- (HPC, Imaging etc)



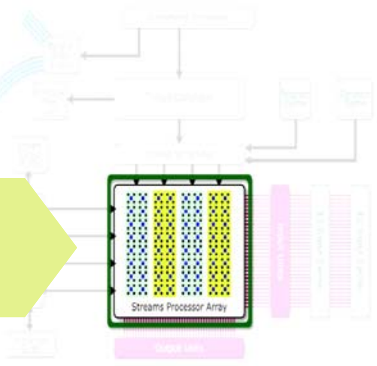
ATI Radeon HD 2000 Stream Compute Architecture

- Optimal utilization of compute resources
- Program cache allows for unlimited program size
- Constant cache allows for unlimited constants



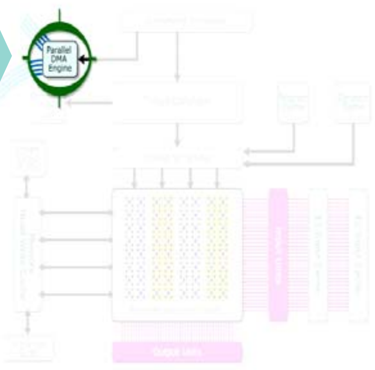
ATI Radeon HD 2000 Stream Compute Architecture

- 320 scalar stream processors optimized for utilization
- Both floating point and integer operation support
- IEEE754 compliance enhancements



ATI Radeon HD 2000 Stream Compute Architecture

- DMA engine that maximizes PCIE bandwidth
- Operates in parallel to the stream processing array
- CPU and stream compute parallelism



AMD Accelerated Computing Software

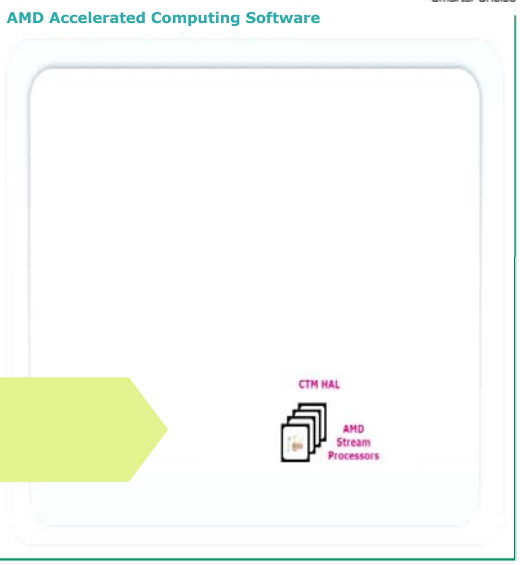
Hardware is only half the story

Open software eco-system necessary for creating rich development environment and tools

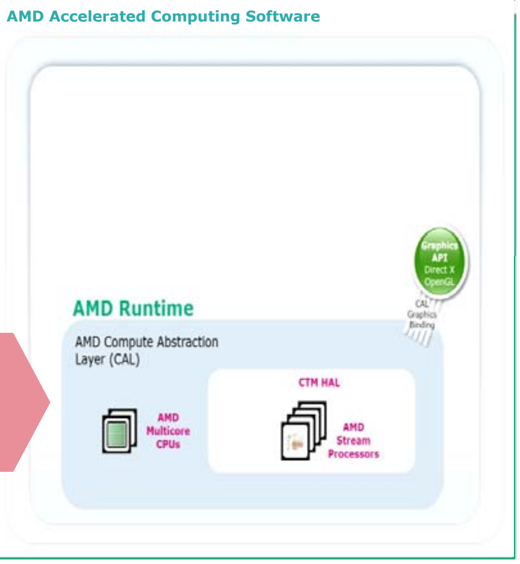
AMD accelerated computing software stack

CTM Hardware Abstraction Layer

- Announced CTM Hardware Abstraction Layer (HAL) in October 2005
- Revealing GPU ISA was a radical move
- Stream processing on GPUs was born
- Enabled partners such as PeakStream, Havok, Rapidmind



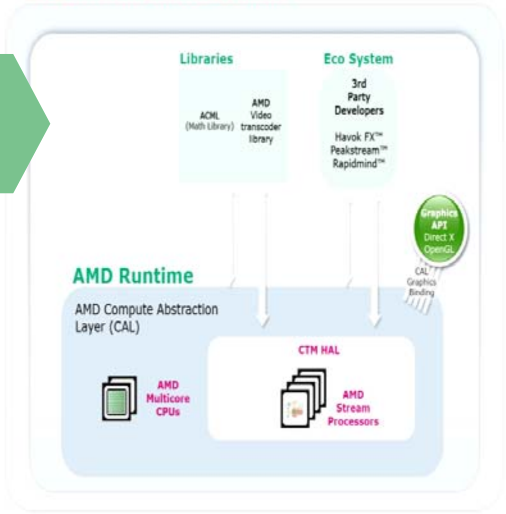
- Taking the stream computing commitment to the next level
- Introducing the AMD Runtime
- Higher level compute abstraction
- Forward compatible
- Automatic multi-core performance scaling



AMD Smarter Choice

AMD Accelerated Computing Software

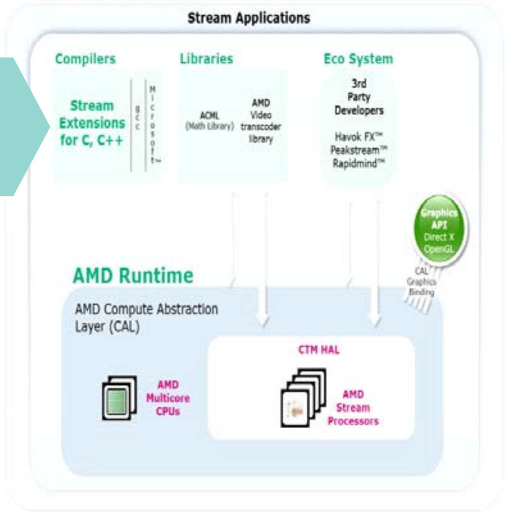
- Libraries
- ACML is a super-optimized math library for CPUs
- Adding AMD Stream Processor support to ACML



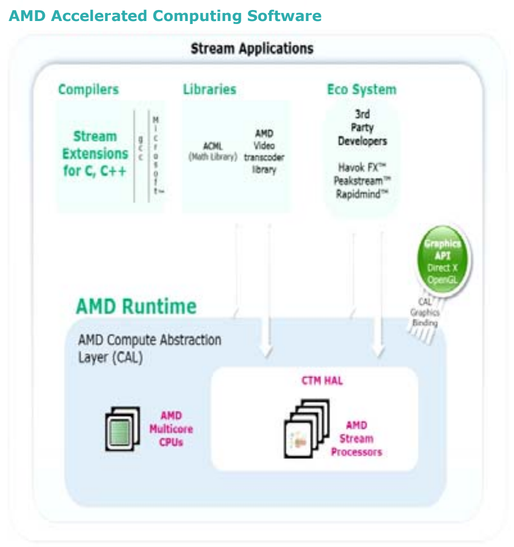
AMD Smarter Choice

AMD Accelerated Computing Software

- Compilers
- Deliver compiler extensions for C, C++
- Developers work in a familiar development environment, with existing languages
- Now they have access to new operators, and can target code at stream processors



AMD Accelerated Computing Software Stack



Elements of Good Stream Processors

Applications

- HPC
- Advanced video processing
- Image and photo processing
- Advanced human computer interfaces
- Medicine
- And many more…

Questions?