THE HETEROGENEOUS SYSTEM ARCHITECTURE
ITS (NOT) ALL ABOUT THE GPU

PAUL BLINZER, FELLOW, HSA SYSTEM SOFTWARE, AMD

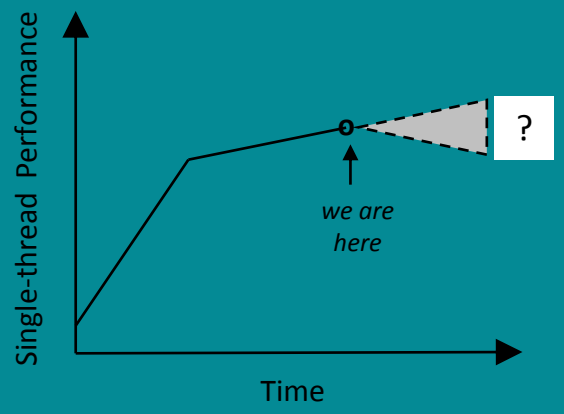
SYSTEM ARCHITECTURE WORKGROUP CHAIR, HSA FOUNDATION

THREE ERAS OF PROCESSOR PERFORMANCE

Single-Core Era

Enabled by:
- ✓ Moore's Observation
- ✓ Voltage Scaling
- ✓ Micro-Architecture

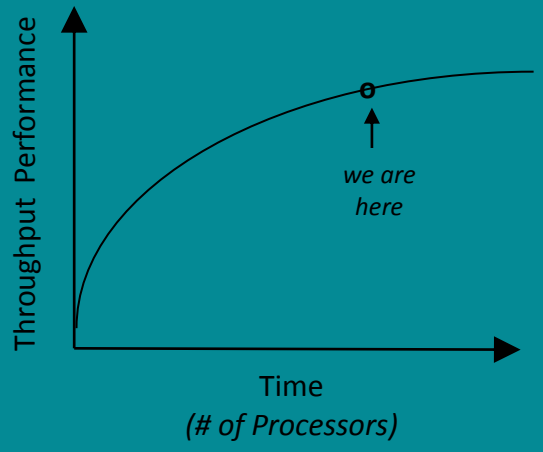
Constrained by:
- ✘ Power
- ✘ Complexity



Multi-Core Era

Enabled by:
- ✓ Moore's Observation
- ✓ Desire for Throughput
- ✓ 20 years of SMP arch

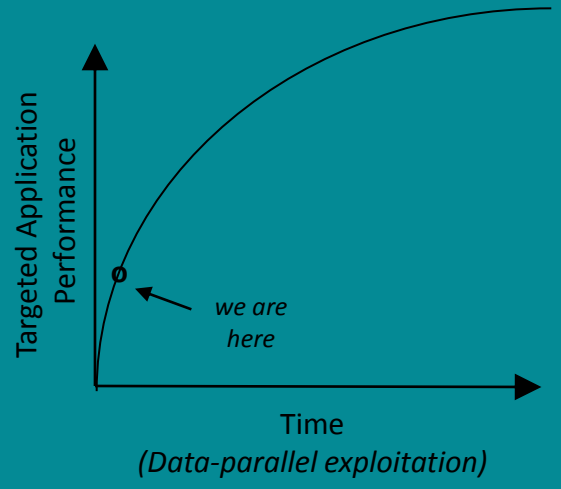
Constrained by:
- ✘ Power
- ✘ Parallel SW availability
- ✘ Scalability



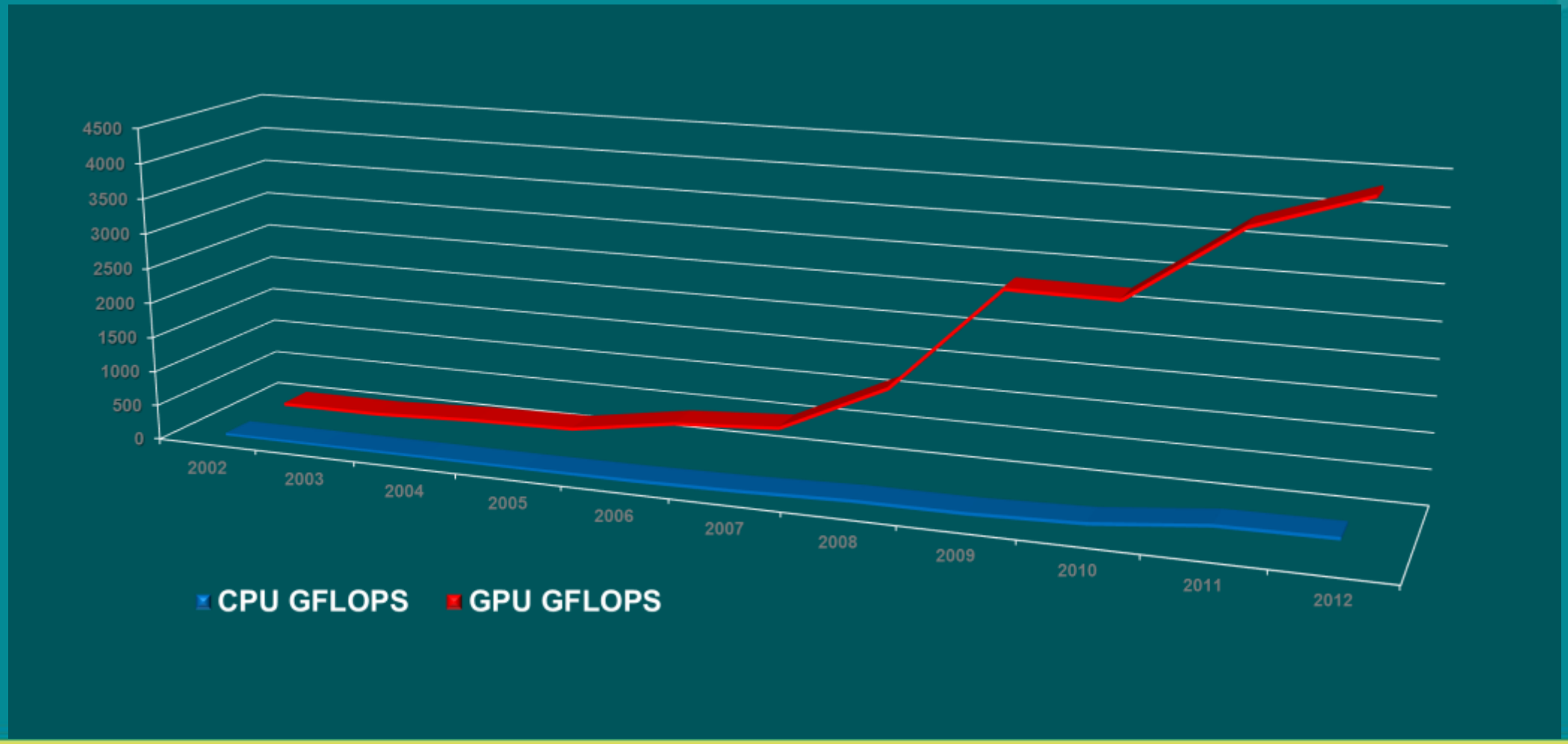
Heterogeneous Systems Era

Enabled by:
- ✓ Moore's Observation
- ✓ Abundant data parallelism
- ✓ Power efficient data parallel processing (GPUs)

Constrained by:
- ✘ Programming models
- ✘ Communication overheads



THE PROGRESS OF FLOATING POINT PERFORMANCE



WHY THE DIFFERENCE? ITS ABOUT THE PURPOSE

- CPUs: built for general purpose, serial instruction threads, often high data locality, lot's of conditional execution and dealing with data interdependency
  - CPU cache hierarchy is focused on general purpose data access from/to execution units, feeding back previously computed data to the execution units with very low latency
  - Comparatively few registers (vs GPUs), but large caches keep often used "arbitrary" data close to the execution units
  - Execution model is synchronous
- GPUs: purpose-built for highly data parallel, latency tolerant workloads
  - Apply the same sequence of instructions over and over on data with little variation but high throughput ("streaming data"), passing the data from one processing stage to another (latency tolerance)
  - Compute units have a relatively large register file store
  - Using a lot of "specialty caches" (constant cache, Texture Cache, etc), data caches optimized for SW data prefetch
  - separate memory spaces (e.g. GPU local memory)
  - Execution model is asynchronous

THE HSA (= HETEROGENEOUS SYSTEM ARCHITECTURE) GOALS

- To bring accelerators forward as a first class processor within the system
  - Unified process address space access across all processors
  - Operates in pageable system memory
  - Full memory coherency between the CPU and HSA components
  - Well-defined relaxed consistency memory model suited for many high level languages
  - User mode dispatch/scheduling (eliminates “drivers” from the dispatch path)
  - QoS through pre-emption and context switching*
- It is not dictating a specific platform or component microarchitecture!
  - It defines the “what needs to be supported”, not the “how it needs to be supported”
  - Focus is on providing a robust data parallel application execution infrastructure for “regular languages”

THE GOALS OF HSA, PART 2

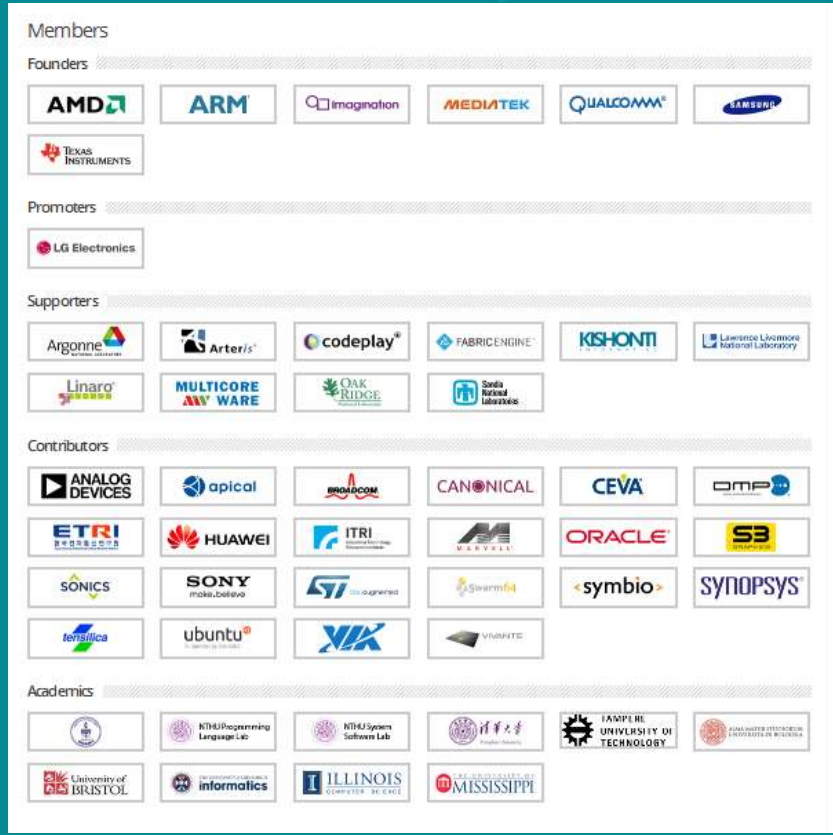
- To attract mainstream programmers
  - By making it easier writing data parallel code
  - Support broader set of languages beyond traditional GPGPU Languages
  - Support for Task Parallel & Nested Data Parallel Runtimes
  - Rich debugging and performance analysis support
- Create a platform architecture accessible for all accelerator types, not only GPU
  - Focused on the APU/SOC and compute, but not preventing other accelerator types to participate
- But one can’t do it alone…

WHAT IS THE HSA FOUNDATION?

- This is the short version…

> "The HSA Foundation is a not-for-profit consortium of SOC and SOC IP vendors, OEMs, academia, OSVs and ISVs defining a consistent heterogeneous platform architecture to make it dramatically easier to program heterogeneous parallel devices"

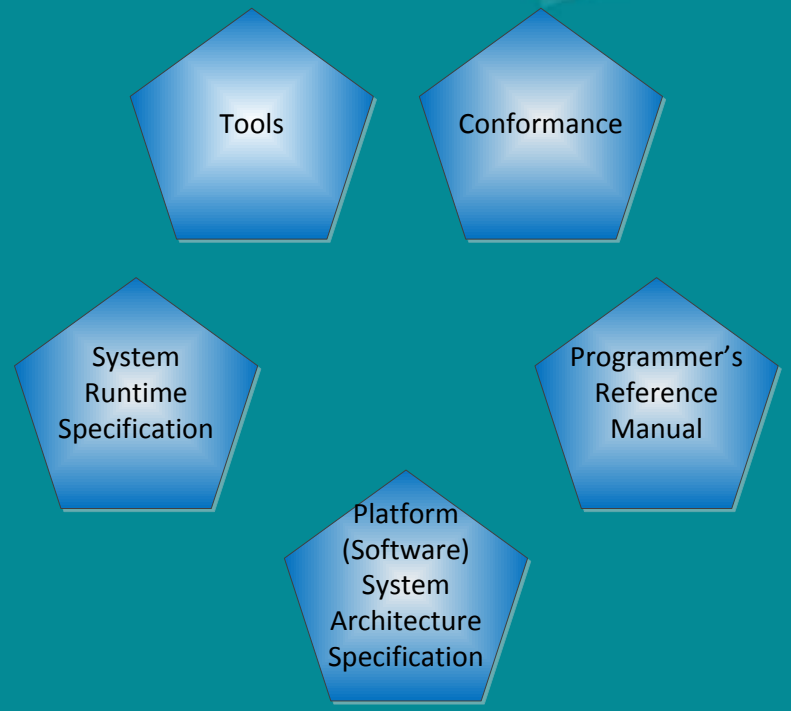
- It spans multiple host platform architectures and programmable data parallel components collaborating within the same HSA system architecture
  - CPU: x86, ARM, MIPS, …
  - Accelerator types: GPUs, DSPs, …
- Specifications define HW & SW platform requirements for applications to target the feature set from high level languages and APIs
- The System Architecture specification defines the required component and platform features for HSA compliant components
- Other HSA specifications leverage these properties for the SW definition



Feb 2014

THE KEY DELIVERABLES OF THE HSA FOUNDATION

- IP, Specifications and API's
- The primary specifications are
  - The HSA Platform System Architecture Specification
    - Focus on hardware requirements and low level system software
    - Support "Small Model" (32bit) and "Large Model" (64bit)
  - The HSA Programmer Reference Manual
    - Definition of an HSAIL Virtual ISA
    - Binary format (BRIG)
    - Compiler writers guide and Libraries developer guide
  - The HSA System Runtime Specification
  - HSA Tools
    - LLVM to HSAIL Compiler
    - HSAIL Assembler
- HSA Conformance Suite
- Open source community and "reference" implementation of SW stack (Linux) by AMD
  - To "jump start" the ecosystem
  - Allow a single shared implementation where appropriate
  - Enable university research in all areas



OPENCL™ WITH HSA, NOT OPENCL™ VS HSA!

- HSA is an optimized platform architecture, which will run OpenCL™ very well
  - It is a complementary standard, not a competitor to OpenCL™
  - It is focused on the hardware and system platform runtime definition more than an API itself
  - It is ready to support many more languages than C/C++, including managed code languages
- OpenCL™ on HSA benefits from a rich and consistent platform infrastructure
  - Pointers shared between CPU and GPU (Shared Virtual Memory), Avoidance of wasteful copies
  - Low latency dispatch
  - Improved and consistent memory model
  - Virtual function calls
  - Flexible control flow
  - Exception generation and handling
  - Device and platform atomics

THE SYSTEM ARCHITECTURE WORKGROUP OF THE HSA FOUNDATION

- The membership spans a wide variety of IP and platform architecture owners
  - Several host platform architectures (x86, ARM, MIPS, …)
- The specifications define a common set of platform properties
  - provide a dependable hardware and system foundation for application software, libraries and runtimes
  - Eliminate “weak points” in the system software- and hardware architecture of traditional platforms, causing overhead processing data parallel workloads
  - Focusing on a few key concepts (queues, signals, memory) optimized and used throughout the architecture

THE SYSTEM ARCHITECTURE WORKGROUP OF THE HSA FOUNDATION

- The main deliverables are:
  - Well-defined, consistent and dependable memory model all HSA agents operate in
  - Share access to process virtual memory between HSA agents (“ptr-is-ptr”)
  - Low-latency workload dispatch, contained in user-mode queues
  - Scalability across a wide range of platforms, from smartphones to clients and servers
  - These properties are leveraged in the “HSA Programmer’s Reference”, HSAIL and HSA Runtime specifications

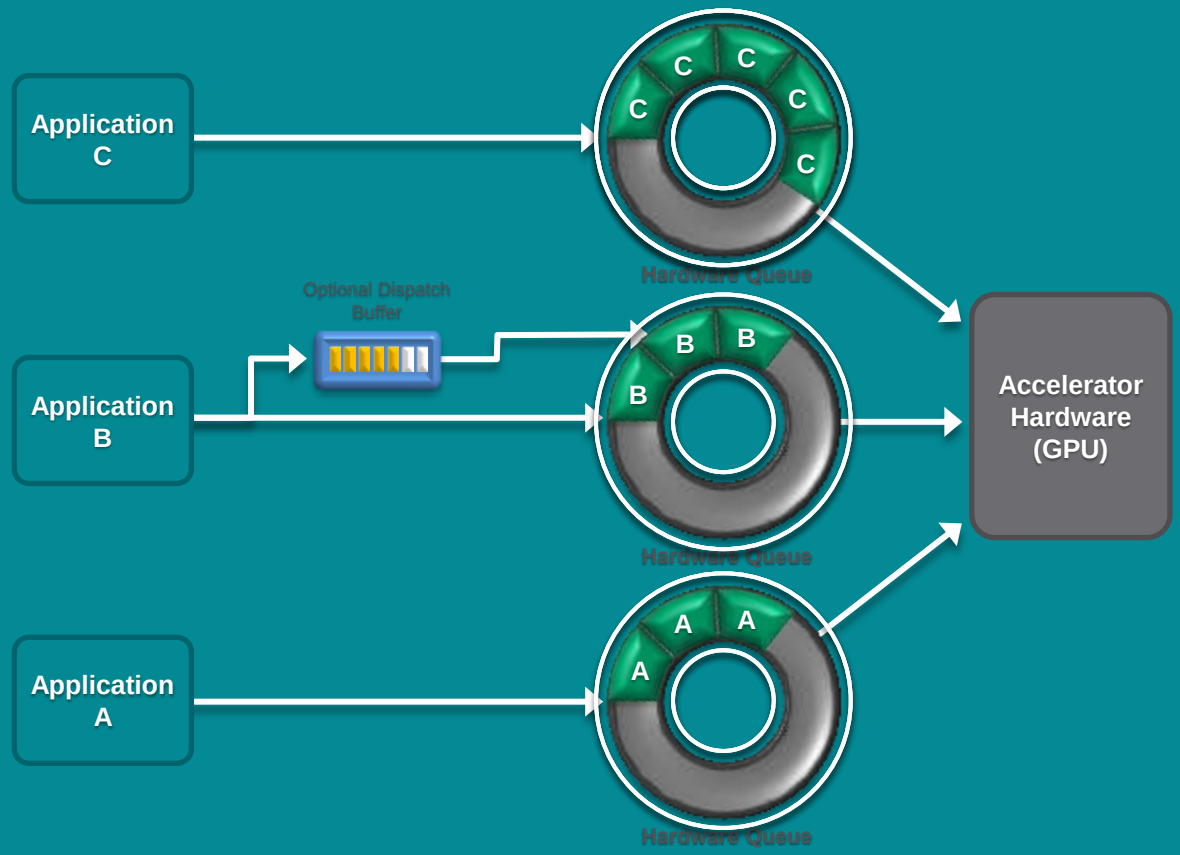
DESIGN CRITERIA FOR THE HSA INFRASTRUCTURE

- HSA defines platform & HW requirements Software can depend on
  - Comprehensive Memory model (well-defined visibility and ordering rules for transactions)
  - Shared Virtual Memory (same page table mapping, HW access enforcement)
  - Page fault capability for every HSA agent
  - Cache Coherency Domains
  - Architected, memory-based signaling and synchronization
  - User Mode Queues, Architected Queuing Language (AQL), Service Queues for runtime callbacks
  - Preemption / Quality of Service*
  - Error Reporting
  - Hardware Debug and profiling support requirements
  - Architected Topology Discovery
- Thin System SW Layer enables HW features for use by an application runtime
  - Mainly responsible for hw init, access enforcement and resource management
  - Provides a consistent, dependable feature set for application layer through SW primitives

HSA SECURITY AND EXECUTION MODEL

- HSA components operate in the same security infrastructure as the host CPU
  - User and privileged memory distinction
  - Hardware enforced process space isolation
  - Page attributes (Read, write, execute) protections enforced by HW and apply as defined by system software
- Internally, the platform partitions functionality by privilege level
  - User mode queues can only run AQL packets within the defined process context
- HSA defines Quality of Service requirements
  - Requires support for mechanisms to schedule both HSA and non-HSA workloads for devices that support both task types with appropriate priority, latency, throughput and scheduling constraints.
  - Context Switch
  - Preempt
  - Terminate and Context Reset

HSA COMMAND AND DISPATCH FLOW



SW view:

- User-mode dispatches to HW
- No Kernel Driver overhead
- Low dispatch times
- CPU & GPU dispatch APIs

HW view:

- HW / microcode controlled
- HW scheduling
- Architected Queuing Language (AQL)
- HW-managed protection

WHAT DEFINES HSA PLATFORMS AND COMPONENTS?

- An HSA compatible platform consists of “HSA components” and “HSA agents”
  - Both adhere to the same various system architecture requirements (memory model, etc)
  - HSA agents can submit AQL packets for execution, may be but are not required to be an HSA component (e.g. host CPU cores can be an HSA agent, but “step out” of the role)
  - HSA components can be a “target” of AQL, by implication they are also HSA agents (= generate AQL packets)
- Defined in the “System Architecture” specification
  - The HSAIL and “Programmer’s Reference Manual” specifications define the software execution model
  - Architected mechanisms to enqueue and dispatch workloads from one HSA agent queue to another eliminate the need to use the host CPU for these purposes for a lot of scenarios
  - Architected infrastructure allows exchanging data with non-HSA compliant components in a platform
  - Fundamental data types are naturally aligned

WHAT DEFINES HSA PLATFORMS AND COMPONENTS?

- There are two different machine models (“small” and “large”)
  - target different functionality levels
  - It takes into account different feature requirements for different platform environments
  - In all cases, the same HSA application programming model is used to target HSA agents and provides the same power–efficient and low-latency dispatch mechanisms, synchronization primitives and SW programming model
- Applications written to target HSA small model machines will generally work on large model machines
  - If the large model platform and host Operating System provides a 32bit process environment

Properties	Small Machine Model	Large Machine Model
Platform targets	embedded or personal device space (controllers, smartphones, etc.)	PC, workstation, cloud Server, etc running more demanding workloads
Native pointer size	32bit	64bit (+ 32bit ptr if 32bit processes are supported)
Floating point size	Half (FP16*), Single (FP32) precision	Half (FP16*), Single (FP32), Double (FP64) precision
Atomic ops size	32bit	32bit, 64bit

*min. Load and store on memory

THE HSA MEMORY MODEL REQUIREMENTS

- A memory model defines how writes by one work item or agent becomes visible to other work items and agents
  - Rules that need to be adhered to by compilers and application threads
  - It defines visibility and ordering rules of write and read events across work items, HSA agents and interactions with non-HSA components in the system
  - Important to define scope for performance optimizations in the compiler and allow reordering in the Finalizer
- HSA is defined to be compatible with C++11, Java and .NET Memory Models
  - Relaxed consistency memory model for parallel compute performance
  - Inherently maps to many CPU and device architectures
  - Efficient sequential consistency mechanisms supported to fit high-level language programming models
  - Scope visibility controlled by Load.Acquire / Store.Release & Barrier semantics

THE HSA MEMORY MODEL REQUIREMENTS

- A consistent, full set of atomic operations is available
  - Naturally aligned on size, small machine model supports 32bit, large machine model supports 32bit and 64bit
- Cache Coherency between HSA agents (& host CPU) is maintained by default
  - Key feature of the HSA system & platform environment
  - Defined transition points to non-HSA “data domains” (e.g. graphics, audio, …)

THE HSA SIGNALING INFRASTRUCTURE

- HSA supports hardware-assisted signaling and synchronization primitives
  - Defines consistent, memory based semantics to synchronize with work items processed by HSA agents
    - e.g. 32bit or 64bit value, content update, wait on value by HSA agents and AQL packets
  - Typically hardware-assisted, power-efficient & low-latency way to synchronize execution of work items between threads on HSA agents

Source: wikimedia commons

- Allows one-to-one and one-to-many signaling
  - The signaling semantics follow general atomicity requirements defined in the memory model
  - System Software, runtime & application SW can use this infrastructure to efficiently build higher-level synchronization primitives like mutexes, semaphores, …

WHAT IS HSAIL?

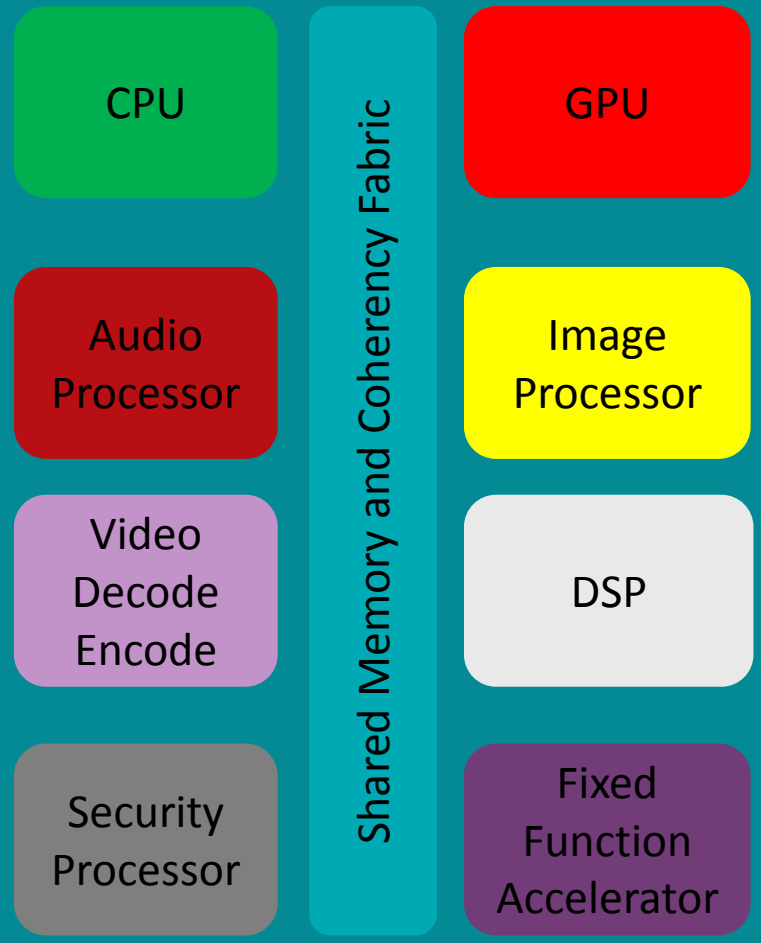
- HSAIL is the intermediate language for parallel compute in HSA
  - HSAIL is a low level instruction set designed for parallel compute in a shared virtual memory environment.
  - Generated by a high level compiler (LLVM, gcc, Java VM, etc)
  - Compiled down to GPU ISA or other parallel processor ISA by an IHV Finalizer
  - Finalizer may execute at run time, install time or build time, depending on platform type
- HSAIL is SIMT in form and does not dictate hardware microarchitecture
  - HSAIL is designed for fast compile time, moving most optimizations to HL compiler
  - Limited register set avoids full register allocation in Finalizer
  - HSAIL is at the same level as PTX: an intermediate assembly or Virtual Machine Target
  - Represented as bit-code in in a Brig file format with support late binding of libraries.

LOOKING BEYOND THE TRADITIONAL GPU LANGUAGES

- Dynamic Languages are one of the most interesting areas for exploration of heterogeneous parallel runtimes and have many applications
- Dynamic Languages have different compilation foundations targeting HSA / HSAIL
  - Java/Scala, JavaScript, Dart, etc
- See Project Sumatra - http://openjdk.java.net/projects/sumatra/
  - Formal Project for GPU Acceleration for Java in OpenJDK
- HSA allows efficient support for other standards based language environments
  - like OpenMP, Fortran, GO, Haskell
  - And domain specific languages like Halide, Julia, and many others

LOOKING BEYOND THE DATA PARALLEL COMPUTE APPLICATION

- The initial release of the HSA specifications focuses on data parallel compute language and application scenarios
  - Focus is on integrating GPUs into the general software infrastructure
  - But the next generations of the specifications may apply to other domains
  - With their domain-specific HW processor language focus
- By design the HSA infrastructure is quite easy to extend
  - Initial focus is on data parallel compute tasks
  - But other areas of support are under consideration for the future
  - Architected Topology infrastructure allows to reliably identify and address domain specific accelerator capabilities
- By design the HSA infrastructure is easy to virtualize
  - Programming model does leverage few, simple hardware & platform paradigms (queues, signals, memory) for its operation
  - Future spec work may put additional requirements to cover such environments



IN SUMMARY…

- HSA is not about a specific API, feature or runtime
  - It is about a paradigm to efficiently access the various heterogeneous components in a system by software
  - It allows application programmers to use the languages of their choice to efficiently implement their code
- HSA is not about a specific hardware or vendor or Operating System
  - It defines a few fundamental requirements and concepts as building blocks software at all levels can depend on
  - HW vendors can efficiently expose their compute acceleration features to software in an architected way
  - OS, runtimes and application frameworks can build efficient data and task parallel runtimes leveraging these
  - Application software can more easily use the right tool for the job through high level language support
- HSA is an open and flexible concept
  - Collaborative participation through the HSA Foundation is encouraged for companies and academia
  - The first set of standards by the HSA Foundation is either released or imminent
  - This is a good time to engage

WHERE TO FIND FURTHER INFORMATION ON HSA?

- HSA Foundation Website: http://www.hsafoundation.com
  - The main location for specs, developer info, tools, publications and many things more
  - Also the location to apply for membership ☺
- Tools are now available at GitHUB – HSA Foundation
  - libHSA Assembler and Disassembler
    - https://github.com/HSAFoundation/HSAIL-Tools
  - HSAIL Instruction Set Simulator
    - https://github.com/HSAFoundation/HSAIL-Instruction-Set-Simulator
  - HSA ISS Loader Library for Java and C++ for creation and dspatch HSAIL Kernels
    - https://github.com/HSAFoundation/Okra-Interface-to-HSAIL-Simulator
- More to come soon …

- With thanks to Phil Rogers, Greg Stoner Sasa Marinkovic and others for some materials and feedback

ANY QUESTIONS?

- Of course there are, so go ahead ☺