Heterogeneous Computing

Featured Speaker

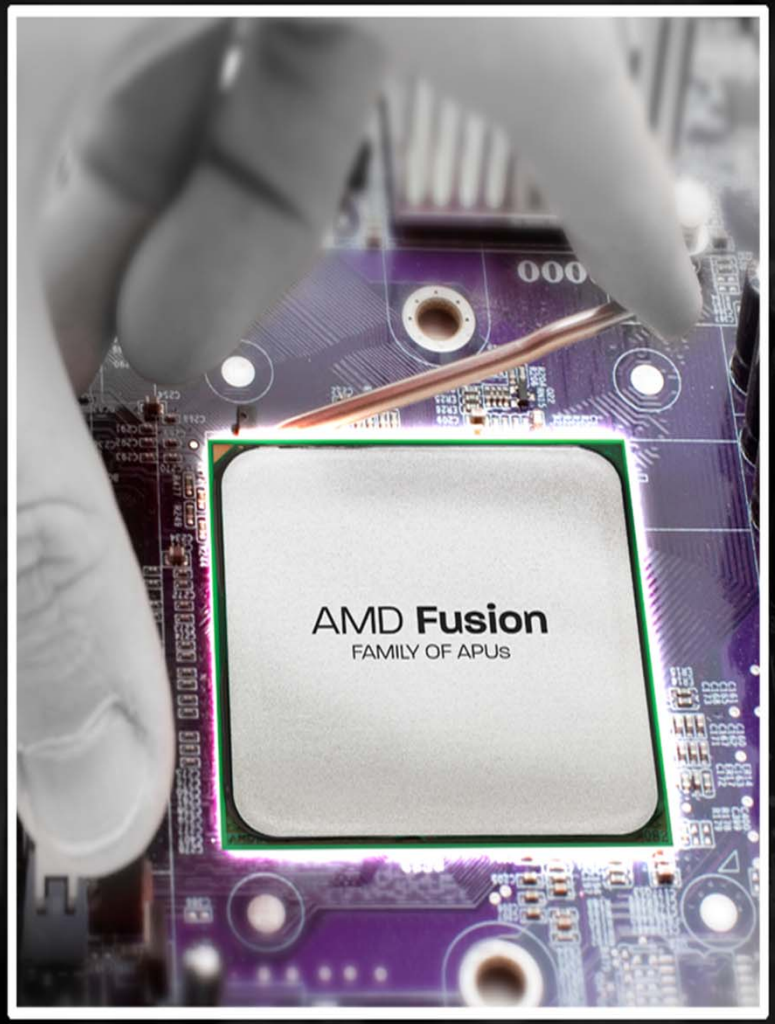
Ben Sander
Senior Fellow
Advanced Micro Devices (AMD)

DR. DOBB'S: GPU AND CPU PROGRAMMING WITH HETEROGENEOUS SYSTEM ARCHITECTURE

Ben Sander
AMD
Senior Fellow

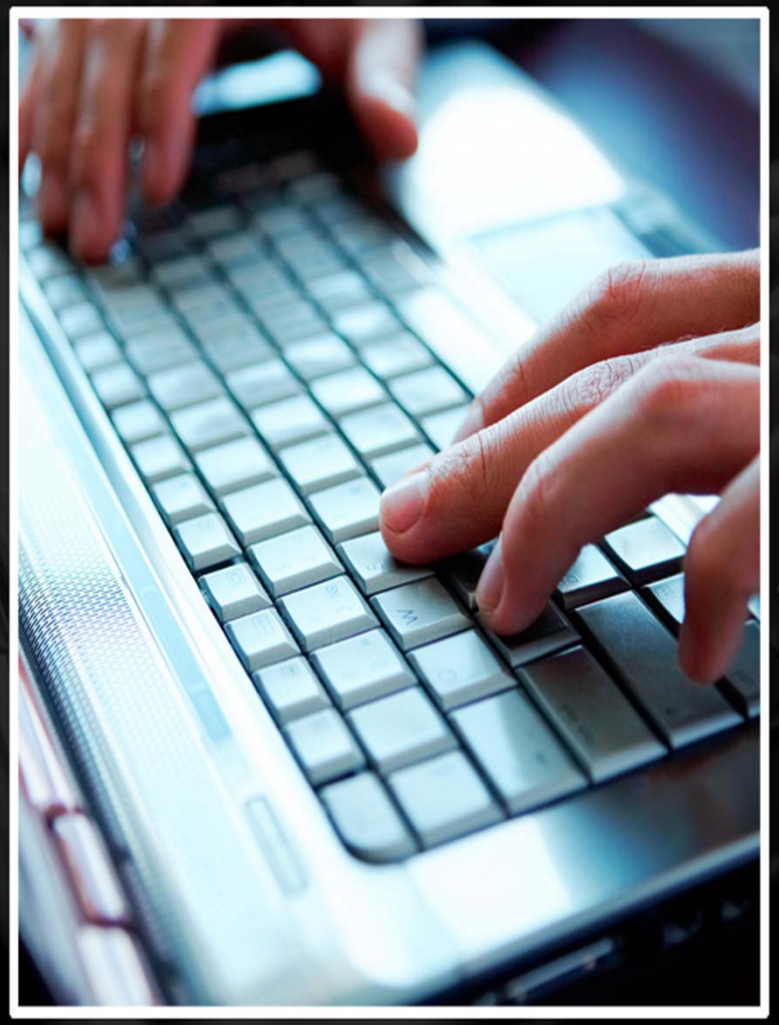
APU: ACCELERATED PROCESSING UNIT

- The APU has arrived and it is a great advance over previous platforms
- Combines scalar processing on CPU with parallel processing on the GPU and high bandwidth access to memory
- How do we make it even better going forward?
  - Easier to program
  - Easier to optimize
  - Easier to load balance
  - Higher performance
  - Lower power

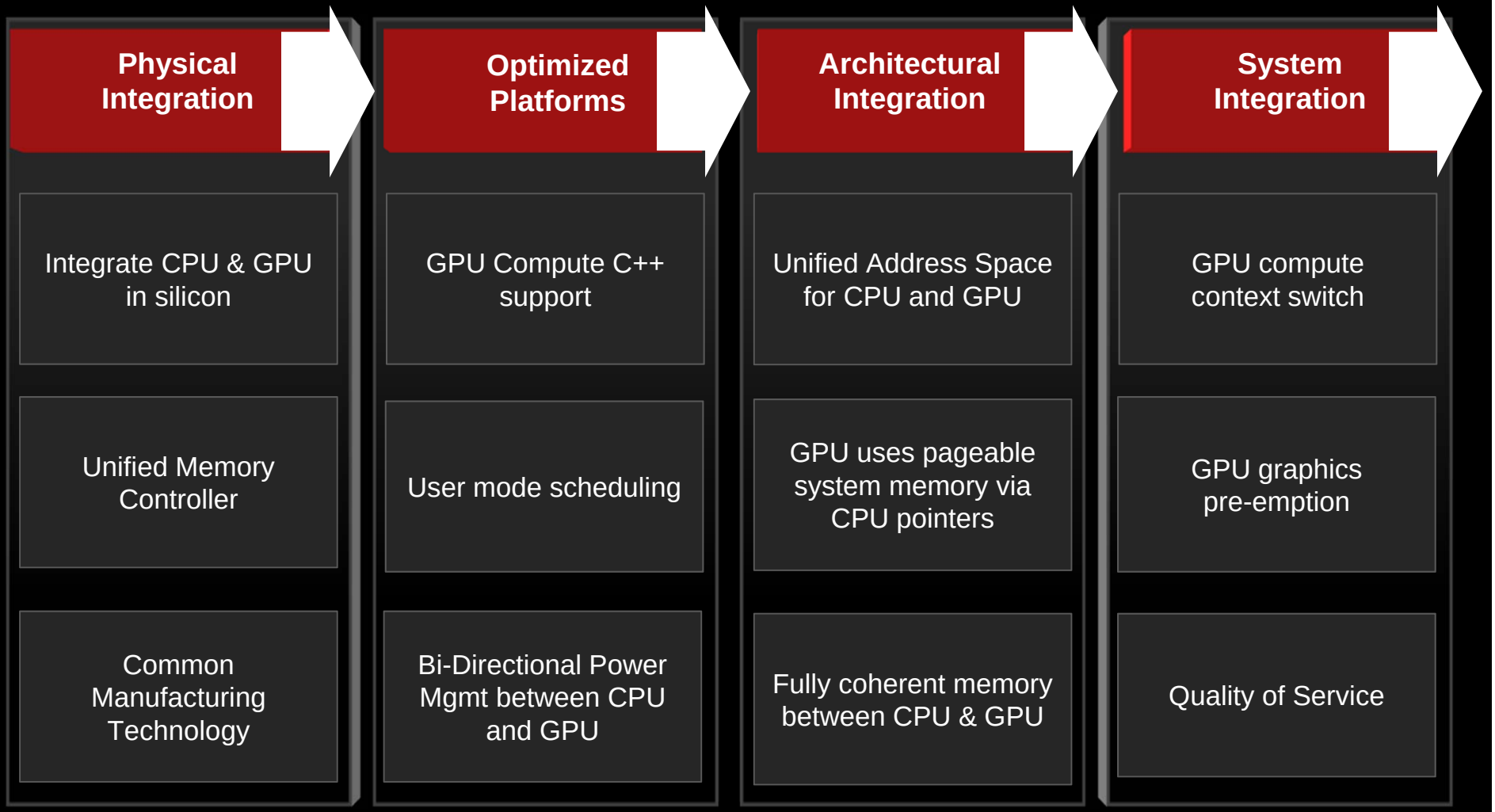


OUTLINE

- Heterogeneous System Architecture
- The future of the heterogeneous platform
- Bolt: C++ Template Library for HSA
- HSAIL and HSA Runtime



HSA FEATURE ROADMAP

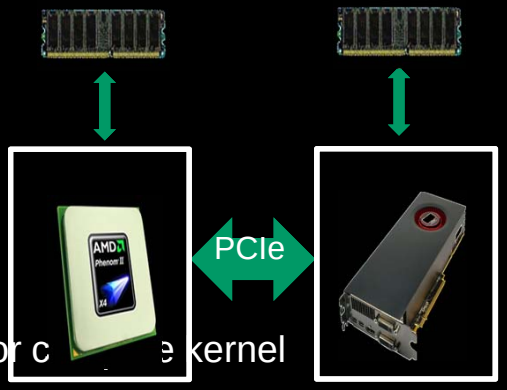


HETEROGENEOUS SYSTEM ARCHITECTURE – AN OPEN PLATFORM

- Open Architecture, published specifications
  - HSAIL virtual ISA
  - HSA memory model
  - HSA system architecture
- ISA agnostic for both CPU and GPU
- Inviting partners to join us, in all areas
  - Hardware companies
  - Operating Systems
  - Tools and Middleware
  - Applications
- HSA Foundation formed in June 2012

Today's Challenge

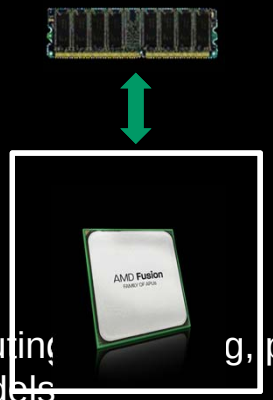
- Separate address spaces
  - Copies
  - Can't share pointers



- New language required for c kernel
  - OpenCL™ looks like C, but sometimes different
  - Compute kernel compiled separately than host code

Emerging Solution

- APUs and HSA!



- Bring GPU computin g, popular, programming models
  - Single-source, fully supported by compiler

BRINGING GPU ACCELERATION TO THE PROGRAMMERS

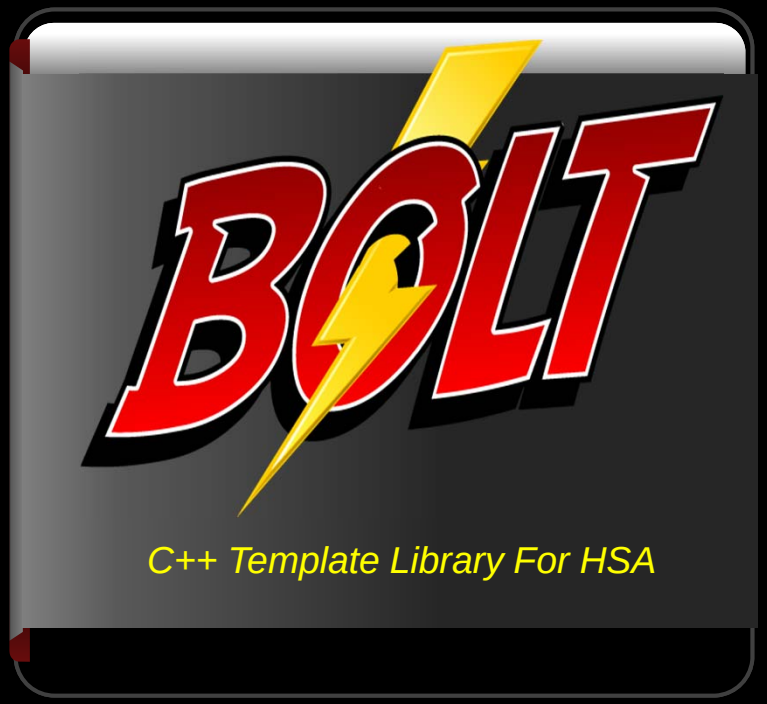
- C++ Accelerated Massive Parallelism (C++ AMP)
  - Adds one language extension
    - "restrict" marks kernel regions that can run on GPU
    - Restricts language features not appropriate for GPUs
  - Included in Microsoft Visual Studio 2012 (August 2012)
    - Includes debugger and profiler support
  - Open spec for C++ AMP available
- Java
  - "**AMD, Oracle Team for OpenJDK 'Sumatra' Java GPU Project" – eWeek, October-2012**
- Bolt
  - C++ Template Library for HSA (announced June-2012)
  - Common library functions: sort, scan, reduce, transform, etc
- HSA Software Stack
  - Runtime and Compiler "building blocks" for other programming

BOLT:
HSA C++ TEMPLATE LIBRARY

AMD Fusion 12 DEVELOPER SUMMIT

MOTIVATION

- Improve developer productivity
  - Optimized library routines for common GPU operations
  - Works with open standards (OpenCL™ and C++ AMP)
  - Distributed as open source
- Make GPU programming as easy as CPU programming
  - Resemble familiar C++ Standard Template Library
  - Customizable via C++ template parameters
  - Leverage high-performance shared virtual memory
- Optimize for HSA
  - Single source base for GPU and CPU
  - Platform Load Balancing



SIMPLE BOLT EXAMPLE

```
#include <bolt/amp/sort.h>
#include <vector>
#include <algorithm>

void main()
{
    // generate random data (on host)
    std::vector<int> a(1000000);
    std::generate(a.begin(), a.end(), rand);

    // sort, run on best device
    bolt::amp::sort(a.begin(), a.end());
}
```

- Interface similar to familiar C++ Standard Template Library
- No explicit mention of C++ AMP or OpenCL™ (or GPU!)
  - More advanced use case allow programmer to supply a kernel in C++ AMP or OpenCL™
- Direct use of host data structures (ie std::vector)
- bolt::sort implicitly runs on the *platform*
  - Runtime automatically selects CPU or GPU (or both)

BOLT FOR C++ AMP : USER-SPECIFIED FUNCTOR

```
#include <bolt/amp/transform.h>
#include <vector>

struct SaxpyFunctor
{
   float _a;
   SaxpyFunctor(float a) : _a(a) {};

   float operator() (const float &xx, const float &yy) restrict(cpu,amp)
   {
         return _a * xx + yy;
   };
};

void main() {
   SaxpyFunctor s(100);
   std::vector<float> x(1000000); // initialization not shown
   std::vector<float> y(1000000); // initialization not shown
   std::vector<float> z(1000000);

   bolt::amp::transform(x.begin(), x.end(), y.begin(), z.begin(), s);
};
```

BOLT FOR C++ AMP : LEVERAGING C++11 LAMBDA

```
#include <bolt/transform.h>
#include <vector>

void main(void)
{
   const float a=100;
   std::vector<float> x(1000000); // initialization not shown
   std::vector<float> y(1000000); // initialization not shown
   std::vector<float> z(1000000);

   // saxpy with C++ Lambda
   bolt::amp::transform(x.begin(), x.end(), y.begin(), z.begin(),
       [=] (float xx, float yy) restrict(cpu, amp) {
              return a * xx + yy;
       });
};
```

- Functor (“a * xx + yy”) now specified inline
- Can capture variables from surrounding scope (“a”) – eliminate boilerplate class

BOLT FOR OPENCL™

```
#include <bolt/cl/sort.h>
#include <vector>
#include <algorithm>

void main()
{
    // generate random data (on host)
    std::vector<int> a(1000000);
    std::generate(a.begin(), a.end(), rand);

    // sort, run on best device
    bolt::cl::sort(a.begin(), a.end());
}
```

- Interface similar to familiar C++ Standard Template Library
- clbolt uses OpenCL™ below the API level
  - Host data copied or mapped to the GPU
  - First call to clbolt::sort will generate and compile a kernel
- More advanced use case allow programmer to supply a kernel in OpenCL™

BOLT FOR OPENCL™ : USER-SPECIFIED FUNCTOR

```
#include <bolt/cl/transform.h>
#include <vector>


BOLT_FUNCTOR(SaxpyFunctor,
struct SaxpyFunctor
{
   float _a;
   SaxpyFunctor(float a) : _a(a) {};

   float operator() (const float &xx, const float &yy)
   {
         return _a * xx + yy;
   };
};
);

void main() {
   SaxpyFunctor s(100);
   std::vector<float> x(1000000); // initialization not shown
   std::vector<float> y(1000000); // initialization not shown
   std::vector<float> z(1000000);

   bolt::cl::transform(x.begin(), x.end(), y.begin(), z.begin(), s);
};
```

- Challenge: OpenCL™ split-source model
  - Host code in C or C++
  - OpenCL™ code specified in strings
- Solution:
  - BOLT_FUNCTOR macro creates both host-side and string versions of “SaxpyFunctor” class definition
    - Class name (“SaxpyFunctor”) stored in TypeName trait
    - OpenCL™ kernel code (SaxpyFunctor class def) stored in ClCode trait.
  - Clbolt function implementation
    - Can retrieve traits from class name
    - Uses TypeName and ClCode to construct a customized transform kernel
    - First call to clbolt::transform compiles the kernel
  - Advanced users can directly create ClCode trait

BOLT: C++ AMP VS. OPENCL™

BOLT for C++ AMP

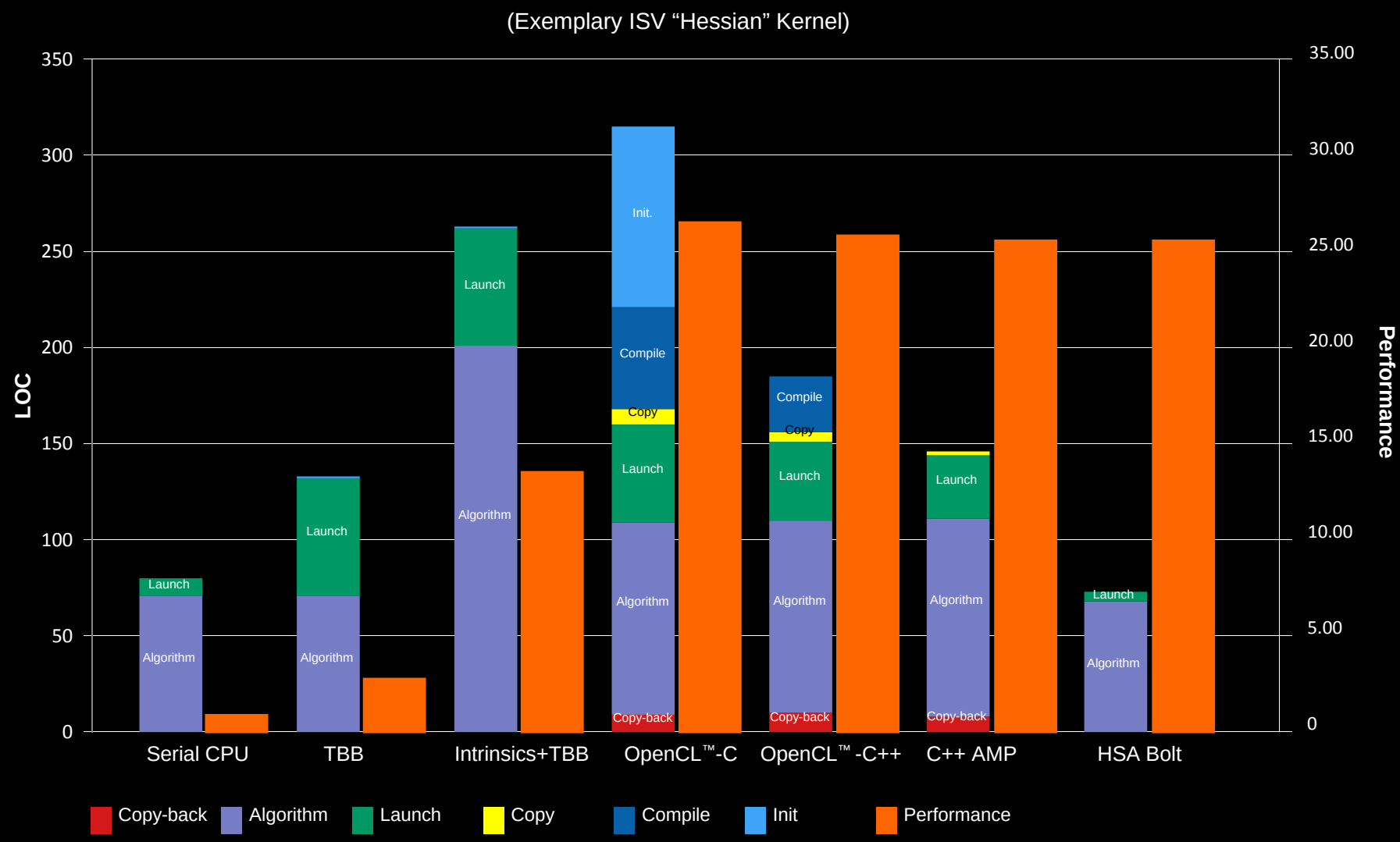
- C++ template library for HSA
  - Developer can customize data types and operations
  - Provide library of optimized routines for AMD GPUs.
- C++ Host Language
- Kernels marked with “restrict(cpu, amp)”
- Kernels written in C++ AMP kernel language
  - Restricted set of C++
- Kernels compiled at compile-time

- C++ Lambda Syntax Supported
- Functors may contain array_view
- Parameters can use host data structures (ie std::vector)
- Parameters can use device memory
- Use “bolt::amp” namespace

BOLT for OpenCL™

- C++ template library for HSA
  - Developer can customize data types and operations
  - Provide library of optimized routines for AMD GPUs.
- C++ Host Language
- Kernels marked with “BOLT_FUNCTOR” macro
- Kernels written in OpenCL™ kernel language
  - Subset of C99, with extensions (ie vectors, builtins)
- Kernels compiled at runtime, on first call
  - Some compile errors shown on first call
- C++11 Lambda Syntax NOT supported
- Functors may not contain pointers
- Parameters can use host data structures (ie std::vector)
- Parameters can use device memory
- Use “bolt::cl” namespace

LINES-OF-CODE AND PERFORMANCE FOR DIFFERENT PROGRAMMING MODELS



AMD A10-5800K APU with Radeon™ HD Graphics – CPU: 4 cores, 3800MHz (4200MHz Turbo); GPU: AMD Radeon HD 7660D, 6 compute units, 800MHz; 4GB RAM.
Software – Windows 7 Professional SP1 (64-bit OS); AMD OpenCL™ 1.2 AMD-APP (937.2); Microsoft Visual Studio 11 Beta

HSA LOAD BALANCING : KEY FEATURES AND OBSERVATIONS

- High-performance shared virtual memory
  - *Developers no longer have to worry about data location (ie device vs host)*

- HSA platforms have tightly integrated CPU and GPU
  - GPU better at wide vector parallelism, extracting memory bandwidth, latency hiding
  - CPU better at fine-grained vector parallelism, cache-sensitive code, control-flow

- Bolt Abstractions
  - Provides insight into the characteristics of the algorithm
    - Reduce vs Transform
  - Abstraction above the details of a “kernel launch”
    - Don’t need to specify device, workgroup shape, work-items, number of kernels, etc
    - *Runtime may optimize these for the platform*

- Bolt has access to both optimized CPU and GPU implementations, at the same time
  - *Let’s use both!*

- *Let’s use both!*

EXAMPLES OF HSA LOAD-BALANCING

Example	Description	Exemplary Use Cases
Data Size	Run large data sizes on GPU, small on CPU	Same call-site used for varying data sizes.
Reduction	Run initial reduction phases on GPU, run final stages on CPU	Any reduction operation.
Border/Edge Optimization	Run wide center regions on GPU, run border regions on CPU.	Image processing.
Platform Super-Device	Distribute workgroups to available processing units on the entire platform.	Kernel has similar performance /energy on CPU and GPU.
Heterogeneous Pipeline	Run a pipelined series of user-defined stages. Stages can be CPU-only, GPU-only, or CPU or GPU.	Video processing pipeline.

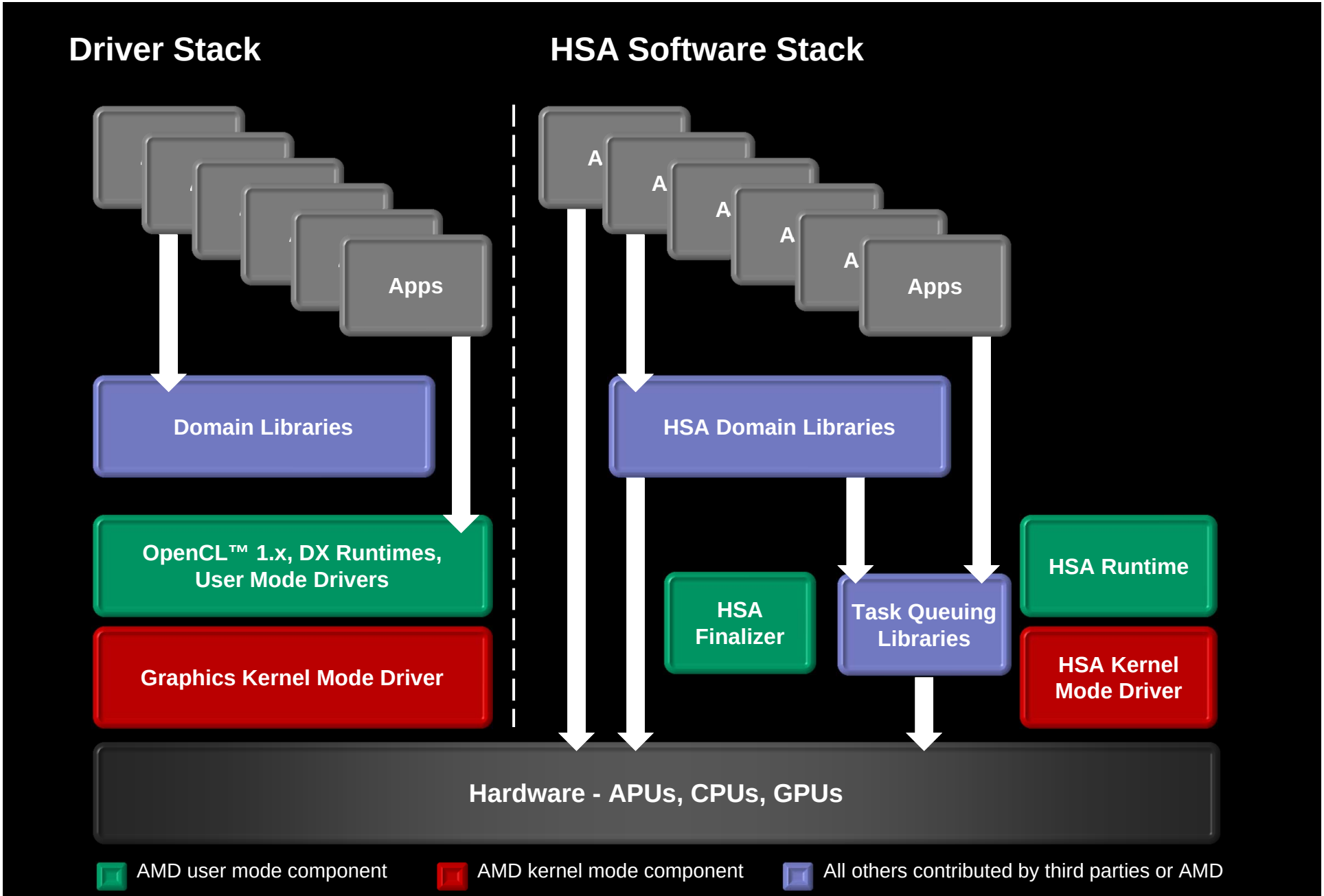
HSA SOFTWARE STACKS

APPLICATIONS AND SYSTEM

AMD Fusion 12 DEVELOPER SUMMIT

HSA INTERMEDIATE LAYER - HSAIL

- HSAIL is a virtual ISA for parallel programs
  - Finalized to ISA by a JIT compiler or "Finalizer"
  - ISA independent by design for CPU & GPU
- Explicitly parallel
  - Designed for data parallel programming
- Support for exceptions, virtual functions, and other high level language features
- Syscall methods
  - GPU code can call directly to system services, IO, printf, etc
- Debugging support


Driver Stack
HSA Software Stack
Apps
A
A
A
A
A
Apps
Domain Libraries
HSA Domain Libraries
OpenCL™ 1.x, DX Runtimes,
User Mode Drivers
HSA
Finalizer
Task Queuing
Libraries
HSA Runtime
Graphics Kernel Mode Driver
HSA Kernel
Mode Driver
Hardware - APUs, CPUs, GPUs
AMD user mode component
AMD kernel mode component
All others contributed by third parties or AMD

AMD'S OPEN SOURCE COMMITMENT TO HSA

- We will open source our linux execution and compilation stack
  - Jump start the ecosystem
  - Allow a single shared implementation where appropriate
  - Enable university research in all areas

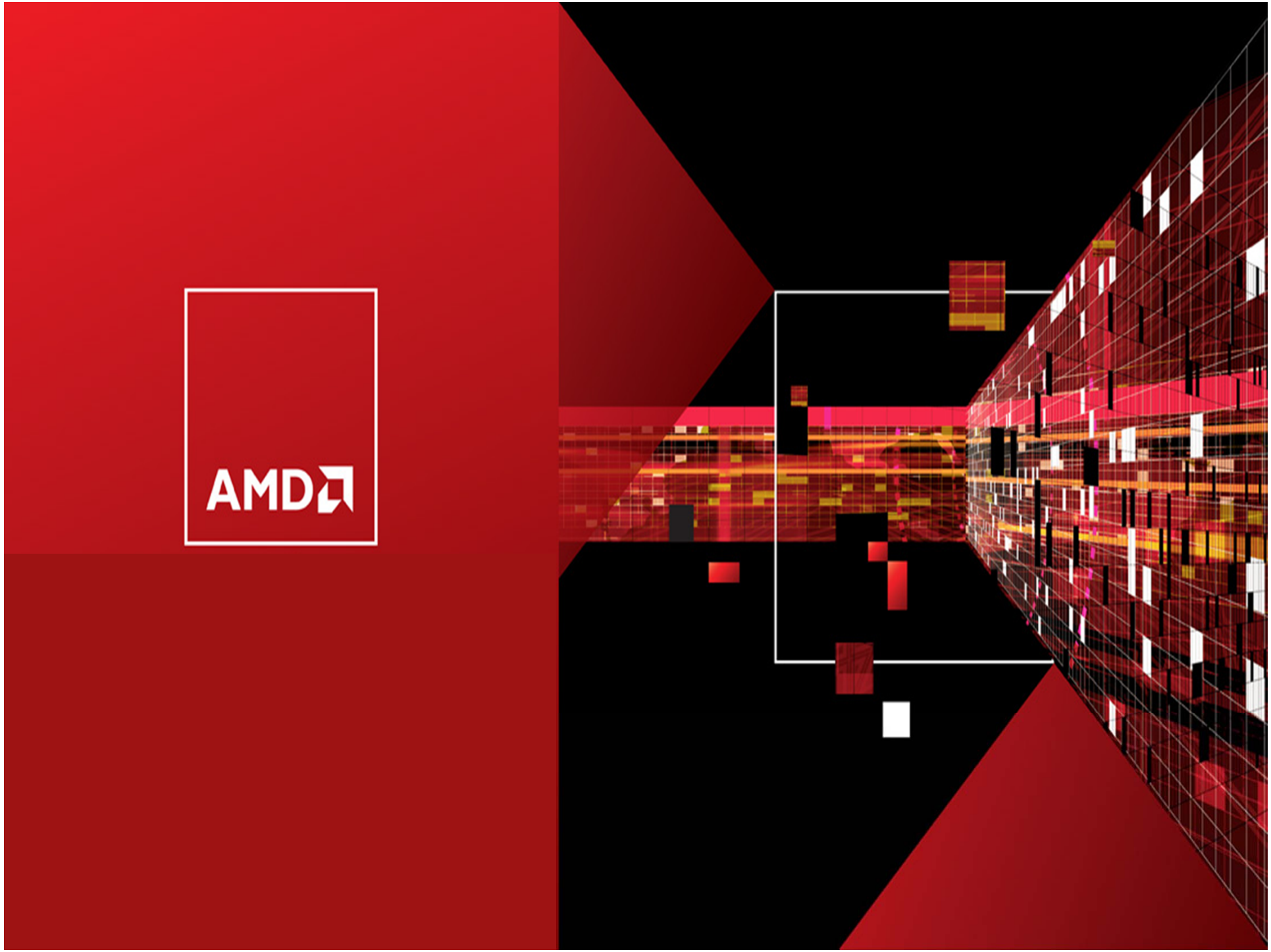
Component Name	AMD Specific	Rationale
HSA Bolt Library	No	Enable understanding and debug
LLVM HSAIL Code Generator	No	Enable research
LLVM Contributions	No	Industry and academic collaboration
HSA Assembler	No	Enable understanding and debug
HSA Runtime	No	Standardize on a single runtime
HSA Finalizer	Yes	Enable research and debug
HSA Kernel Driver	Yes	For inclusion in linux distros

CLOSING THOUGHTS

- The APU is here and is a tremendous advance over previous platforms
  - HSA will make this even better with shared memory, user-mode scheduling, and more
- This will change the way we program GPUs
  - (Same great power and performance benefits)
  - Bring GPU acceleration to existing programming models
  - Seamlessly use host-side data structures and pointers on the GPU
  - Leverage both CPU and GPU, as appropriate
- Heterogeneous System Architecture enables this vision
  - Open-source compilers and runtimes
  - Supported by multiple vendors

LINKS

- C++ “wrapper” interface for OpenCL™
  - Substantially reduce boilerplate initialization code previously required to write an OpenCL™ program
  - Works on any OpenCL™ 1.2 compliant implementation (version for OpenCL™ 1.1 also available)
  - http://www.khronos.org/registry/cl/api/1.2/cl.hpp
- OpenCL Static Kernel Language (includes templates for OpenCL kernels)
  - Supported in AMD APP SDK 2.7
  - http://blogs.amd.com/developer/2012/05/21/opencl%E2%84%A2-1-2-and-c-static-kernel-language-now-available/
- Bolt
  - Bolt will be available as an open-source project in 2H-2012
- C++ Accelerated Massive Parallelism (C++ AMP)
  - Spec available here: http://download.microsoft.com/download/4/0/E/40EA02D8-23A7-4BD2-AD3A-0BFFFB640F28/CppAMPLanguageAndProgrammingModel.pdf
  - C++ AMP supported in Microsoft Visual Studio 2012
- Aparapi (for Java)
  - Program the GPU from Java! (including ability to write kernels in Java)
  - http://code.google.com/p/aparapi/


AMD

Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

OpenCL and the OpenCL logo are trademarks of Apple, Inc. and are used by permission by Khronos.

© 2011 Advanced Micro Devices, Inc. All Rights Reserved.