HETEROGENEOUS SYSTEM ARCHITECTURE (HSA) AND THE SOFTWARE ECOSYSTEM
CUDA gives developers access to unprecedented performance.

Not easy to use … but enough performance-hungry developers willing to endure pain.

Low Consumer space adoption … esp. due to lack of cross-platform support.
THE RUNAWAY SUCCESS OF JAVA

Easy to program

Truly cross platform – **Write Once Run Anywhere**

Lack of performance efficiency offset by platform capability

Adoption

<table>
<thead>
<tr>
<th>Year</th>
<th>JDK1.0</th>
<th>J2SE 5.0</th>
<th>Java SE 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td></td>
<td>4.5M developers</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td></td>
<td></td>
<td>6M developers</td>
</tr>
<tr>
<td>2002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Java 7
10M+ developers
Millions of Apps
CURRENT HETEROGENEOUS SYSTEM: CPU+dGPU

NUMA based CPU + GPU can reach high performance but
- Memory Copy always Power Consumption
- System Cost always high and not flexible
OpenCL’s Cross-Platform Appeal on APU/DGPU

- **OpenCL 1.0 Announced**
- **OpenCL 1.1 SDK 2.2**
- **35k+ downloads**
- **11 Llano launch Apps**
- **300K+ downloads**
- **100+ Apps**

Abundant performance + same complexity as CUDA programming

Cross platform resonates with developers (*needs per-platform optimization*)
Innovation: HSA – CPU and GPU under UMA

HSA: Heterogeneous System Architecture

Unified memory Model
Intractability between CPU and GPU
Co-processor not Accelerator

CPU and GPU in FUSION SoC

Serial and Task Parallel Workloads
WHAT IS HSA?

An *intelligent computing architecture* that enables CPU, GPU and other processors to work in *harmony* on a single piece of silicon by *seamlessly* moving the right tasks to the best suited processing element.
AND DESIGNED BEYOND GPU

Dedicated Processor/Engine becoming co-processor

- Full Programming language support
- User Mode Queueing
- Heterogeneous Unified Memory Access (hUMA)
- Pageable Memory
- Bidirectional coherency
- Compute context switch and preemption
components:
- Compliant heterogeneous computing hardware
- A software compilation stack
- A user - space runtime system
- Kernel - space system components

Overall Vision:
- Make GPU easily accessible
  - Support mainstream languages
  - Expandable to domain specific languages
- Make compute offload efficient
  - Direct path to GPU (avoid Graphics overhead)
  - Eliminate memory copy
  - Low-latency dispatch
- Make it ubiquitous
  - Drive HSA as a standard through HSA Foundation
  - Open Source key components

HSA SW STACK CHANGING PROGRAMMING MODEL
HSA OBJECTIVE

HSA creates an improved processor design that exposes the benefits and capabilities of mainstream programmable compute elements, working together seamlessly.
**AMD HSA ROADMAP**

**Physical Integration**
- Integrate CPU & GPU in silicon
- Unified Memory Controller
- Common Manufacturing Technology

**Optimized Platforms**
- GPU Compute C++ support
- User mode scheduling
- Bi-Directional Power Mgmt between CPU and GPU

**Architectural Integration**
- Unified Address Space for CPU and GPU
- GPU uses pageable system memory via CPU pointers
- Fully coherent memory between CPU & GPU

**System Integration**
- GPU compute context switch
- GPU graphics pre-emption
- Quality of Service
WHAT IS hUMA?

heterogeneous UNIFORM MEMORY ACCESS
UNDERSTANDING UMA

Original meaning of UMA is **Uniform Memory Access**

- Refers to how processing cores in a system view and access memory
- All processing cores in a true UMA system share a single memory address space

Introduction of GPU compute created systems with Non-Uniform Memory Access (NUMA)

- Require data to be managed across multiple heaps with different address spaces
- Add programming complexity due to frequent copies, synchronization, and address translation

HSA restores the GPU to Uniform memory Access

- Heterogeneous computing replaces GPU Computing
INTRODUCING HUMA

UMA

CPU
CPU
CPU
CPU

NUMA

CPU Memory
CPU
CPU
CPU
CPU

GPU Memory
GPU

hUMA

Memory
CPU
CPU
CPU
CPU

GPU

APU

APU with HSA

CPU
CPU
CPU
CPU

GPU
**HUMA KEY FEATURES**

**BI-DIRECTIONAL COHERENT MEMORY**
Any updates made by one processing element will be seen by all other processing elements - GPU or CPU

**PAGEABLE MEMORY**
GPU can take page faults, and is no longer restricted to page locked memory

**ENTIRE MEMORY SPACE**
CPU and GPU processes can dynamically allocate memory from the entire memory space
HUMA KEY FEATURES

Coherent Memory:
Ensures CPU and GPU caches both see an up-to-date view of data

Pageable memory:
The GPU can seamlessly access virtual memory addresses that are not (yet) present in physical memory

Entire memory space:
Both CPU and GPU can access and allocate any location in the system’s virtual memory space
WITHOUT POINTERS* AND DATA SHARING

Without hUMA:

- CPU explicitly copies data to GPU memory
- GPU completes computation
- CPU explicitly copies result back to CPU memory

*A Pointer is a named variable that holds a memory address. It makes it easy to reference data or code segments by a name and eliminates the need for the developer to know the actual address in memory. Pointers can be manipulated by the same expressions used to operate on any other variable.

Only the data array can be copied since GPU cannot follow embedded data-structure links.
With hUMA:

- CPU simply passes a pointer to GPU
- GPU completes computation
- CPU can read the result directly – no copying needed!

**WITH POINTERS* AND DATA SHARING**

*A Pointer is a named variable that holds a memory address. It makes it easy to reference data or code segments by a name and eliminates the need for the developer to know the actual address in memory. Pointers can be manipulated by the same expressions used to operate on any other variable.*
TOP 10 DRIVING HARDWARE COHERENT ON GPU/APU

1. Much easier for programmers
2. No need for special APIs
3. Move CPU multi-core algorithms to the GPU without recoding for absence of coherency
4. Allow finer grained data sharing than software coherency
5. Implement coherency once in hardware, rather than N times in different software stacks
6. Prevent hard to debug errors in application software
7. Operating systems prefer hardware coherency – they do not want the bug reports to the platform
8. Probe filters and directories will maintain power efficiency
9. Full coherency opens the doors to single source, native and managed code programming for heterogeneous platforms
10. Optimal architecture for heterogeneous computing on APUs and SOCs
<table>
<thead>
<tr>
<th>HUMA FEATURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to Entire Memory Space</td>
</tr>
<tr>
<td>➤ Pageable memory</td>
</tr>
<tr>
<td>➤ Bi-directional Coherency</td>
</tr>
<tr>
<td>➤ Fast GPU access to system memory</td>
</tr>
<tr>
<td>➤ Dynamic Memory Allocation</td>
</tr>
</tbody>
</table>
Heterogeneous System Architecture

- Efficiently support a wide assortment of data-parallel and task-parallel programming models
- Provides a unified view of fundamental computing elements
- Allows a programmer to write applications that seamlessly integrate
  - CPUs (called *latency compute units*)
  - GPUs (called *throughput compute units*)
Balance between CPU and GPU for performance and power efficiency

Make GPUs accessible to wider audience of programmers
  - Programming models close to today’s CPU programming models
  - Enabling more advanced language features on GPU
  - Shared virtual memory enables complex pointer-containing data structures (lists, trees, etc.) and hence more applications on GPU
  - Kernel can enqueue work to any other device in the system (e.g. GPU->GPU, GPU->CPU)
    - Enabling task-graph style algorithms, Ray-Tracing, etc

Clearly defined HSA memory model enables effective reasoning for parallel programming

HSA provides a compatible architecture across a wide range of programming models and HW implementations.
**IMPROVE PERFORMANCE AND REDUCE POWER**

- Pass pointers rather than copying memory
  - Eliminate power and time associated with data copies

- Reduce kernel launch time
  - User-mode queuing
    - Avoid OS kernel transitions when commands are sent to the GPU
  - “Architected Queuing Language”
    - Stable, architected, heterogeneous compute engine command language

- Optimizing GPU compute compiler
  - Optimizations done in single compiler (rather than 2-3 today)
  - High-Level compiler performs optimizations and generates HSAIL
  - Finalize converts HSAIL to target ISA – focused on correctness and compilation speed
RECAP HSA SW STACK

components:
• Compliant heterogeneous computing hardware
• A software compilation stack
• A user-space runtime system
• Kernel-space system components

Overall Vision:
• Make GPU easily accessible
  • Support mainstream languages
  • Expandable to domain specific languages
• Make compute offload efficient
  • Direct path to GPU (avoid Graphics overhead)
  • Eliminate memory copy
  • Low-latency dispatch
• Make it ubiquitous
  • Drive HSA as a standard through HSA Foundation
  • Open Source key components
HSA ARCHITECTURE

- GPU compute C++ support
- User Mode Scheduling
- Fully coherent memory between CPU & GPU
- GPU uses pageable system memory via CPU pointers
- GPU graphics pre-emption
- GPU compute context switch

HSA Software Stack

- HSA Domain Libraries, OpenCL™ 2.x Runtime
- Task Queuing Libraries
- HSA JIT
- HSA Runtime
- HSA Kernel Mode Driver

Shared Memory, User Mode Queues

CPU
- Audio Processor
- Video Hardware
- Encode Decode

GPU
- Fixed Function Acctr
- DSP
- Image Signal Processing
HSA INTERMEDIATE LANGUAGE - HSAIL

- Designed for C99, C++ 2011, Java, Renderscript, OpenCL, C++ AMP
- 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
**OPENCL™ AND HSA**

- HSA is an optimized platform architecture for OpenCL™
  - Not an alternative to OpenCL™
- OpenCL™ on HSA will benefit from
  - Avoidance of wasteful copies
  - Low latency dispatch
  - Improved memory model
  - Pointers shared between CPU and GPU
- HSA also exposes a lower level programming interface, for those that want the ultimate in control and performance
  - Optimized libraries may choose the lower level interface
How compute dispatch operates today in the **driver model**

How compute dispatch improves **under HSA**
TODAY’S COMMAND AND DISPATCH FLOW

- Command Flow
- Data Flow

Application A
Direct3D
User Mode Driver
Soft Queue
Kernel Mode Driver

Command Buffer
DMA Buffer

Hardware Queue

GPU HARDWARE
TODAY’S COMMAND AND DISPATCH FLOW

Command Flow

Data Flow

Application A

Direct3D

User Mode Driver

Soft Queue

Kernel Mode Driver

Application B

Direct3D

User Mode Driver

Soft Queue

Kernel Mode Driver

Application C

Direct3D

User Mode Driver

Soft Queue

Kernel Mode Driver

DMA Buffer

Command Buffer

Hardware Queue

GPU HARDWARE
TODAY’S COMMAND AND DISPATCH FLOW

Command Flow

Data Flow

Application A
Direct3D
User Mode Driver
Soft Queue
Kernel Mode Driver
Command Buffer
DMA Buffer

Application B
Direct3D
User Mode Driver
Soft Queue
Kernel Mode Driver
Command Buffer
DMA Buffer

Application C
Direct3D
User Mode Driver
Soft Queue
Kernel Mode Driver
Command Buffer
DMA Buffer

GPU HARDWARE

Hardware Queue

Today’s Command and Dispatch Flow

Hardware Queue

Today’s Command and Dispatch Flow

Hardware Queue

Today’s Command and Dispatch Flow

Hardware Queue
HSA COMMAND AND DISPATCH FLOW

- Application codes to the hardware
- User mode queuing
- Hardware scheduling
- Low dispatch times

- No APIs
- No Soft Queues
- No User Mode Drivers
- No Kernel Mode Transitions
- No Overhead!
HSA COMMAND AND DISPATCH CPU <-> GPU

Application / Runtime

- B to CPU1
- CPU1 to B
- C to CPU2
- A to CPU1
- App to C
- App to A

Click 3 and 4
Driver Stack

- Domain Libraries
- OpenCL™ 1.x, DX Runtimes, User Mode Drivers
- Graphics Kernel Mode Driver

HSA Software Stack

- HSA Domain Libraries, OpenCL™ 2.x Runtime
- HSA JIT
- Task Queuing Libraries
- HSA Runtime
- HSA Kernel Mode Driver

Hardware - APUs, CPUs, GPUs

- User mode component
- Kernel mode component
- Components contributed by third parties
QUICK LOOK AT HSA COMPILER INFRASTRUCTURE

- Clang - Rendscript
  LLVM-RS-CC
  Optimized Output
  Optimized BitCode
  LibBCC
  Optimized for Fast Compiler Time
  LibRS Loader
  Processor

- Clang –C++ AMP, OpenCL, KL, etc.
  LLVM
  Optimized Output
  BRIG Binary Object (Machine Independent Optimized HSAIL)
  Finalizer/JIT
  Optimized for Fast Compiler Time
  Library
  Loader
  Processor
GPU now supports a full MMU which allow for User protected execution, protected memory and memory curtaining
  - Read, write and execute protections by page table entry
  - Brings inter-process protection
  - Improved fault handling

The ability to kill errant task on the GPU.
**INTEROP OF HSA AND GRAPHICS**

- OpenGL-ES/EGL can share data with HSA Runtime
  - Buffer (Vertex/Pixelbuffer)
  - Texture
  - Renderbuffer

- Mapping
  - HSA Image -> OpenGL-ES Texture, renderbuffer
  - HSA buffer -> OpenGL-ES buffer

- Sync
  - Acquire and Release mechanism
SECURITY IMPROVEMENTS WITH HSA

- With HSA, GPU operates in the same security infrastructure as the CPU
  - User and privileged memory
  - Read, write and execute protections by page table entry

- Internally, the GPU partitions functionality by privilege level
  - User mode compute queues can only run AQL packets
  - User mode graphics command buffers cannot write privileged registers

- HSA supports fixed time context switching, which is resistant to denial of service (DoS) attacks
  - Today’s GPUs are vulnerable to denial of service attacks
    - Long or infinite shader programs
    - Full GPU reset required to restore service
  - With HSA, fair scheduling and context switching ensures a responsive system
Support for Today GPU Programming Model
- Data Parallelism - Embarrassingly Parallel Application
- SPMD – Single Program Multiple Data Based Application

In addition supporting richer set of Parallel Solutions
- Task-Parallelism, Nested-Parallelism, Braided-Parallelism,
- Task-graph style algorithms

Support for Algorithm that need Inter-task Communication.

Application that need exceptions processing
RICHER SUPPORT OF ADVANCED ALGORITHMS

Computational Photography
- Feature Extraction
- Feature Matching
- Geometric Verification
- Image Stitching
- HDR
- DoF control
- Lightfield Imaging

Visual Search
- Feature Extraction
- Feature Matching
- Geometric Verification

Gesture Recognition
- Image Segmentation
- Motion Estimation
- Computation of Geometry (shape analysis)

Hidden Markov Model acceleration
Operate on large Markov Random Fields
APPLICATION AREAS WITH ABUNDANT PARALLEL WORKLOADS

- Natural UI & Gestures: Touch, gesture, and voice
- Biometric Recognition: Secure, fast, accurate: face, voice, fingerprints
- Augmented Reality: Superimpose graphics, audio, and other digital information as a virtual overlay
- Content Everywhere: Content from any source to any display seamlessly
- AV Content Management: Searching, indexing and tagging of video & audio, multimedia data mining
- Beyond HD Experiences: Streaming media, new codecs, 3D, transcode, audio
ACCELERATED WORKLOADS

CLIENT AND SERVER EXAMPLES
LOOKING FOR FACES IN ALL THE RIGHT PLACES

Quick HD Calculations
Search square = 21 x 21
Pixels = 1920 x 1080 = 2,073,600
Search squares = 1900 x 1060 = ~2 Million
LOOKING FOR DIFFERENT SIZE FACES – BY SCALING THE VIDEO FRAME

More HD Calculations
70% scaling in H and V
Total Pixels = 4.07 Million
Search squares = 3.8 Million
HAAR CASCADE STAGES

Stage N

Feature k
Feature l
Feature m

Stage N+1

Feature p
Feature r
Feature q

Face still possible?
Yes
No

REJECT FRAME
22 CASCADE STAGES, EARLY OUT BETWEEN EACH

STAGE 1 → STAGE 2 → STAGE 21 → STAGE 22 → FACE CONFIRMED

NO FACE

Final HD Calculations
Search squares = 3.8 million
Average features per square = 124
Calculations per feature = 100
Calculations per frame = 47 GCalcs

Calculation Rate
30 frames/sec = 1.4TCalcs/second
60 frames/sec = 2.8TCalcs/second

…and this only gets front-facing faces
“Trinity” A10-4600M (6CU@497Mhz, 4 cores@2700Mhz)

AMD A10 4600M APU with Radeon™ HD Graphics; CPU: 4 cores @ 2.3 MHz (turbo 3.2 GHz); GPU: AMD Radeon HD 7660G, 6 compute units, 685MHz; 4GB RAM; Windows 7 (64-bit); OpenCL™ 1.1 (873.1)
By seamlessly sharing data between CPU and GPU, HSA allows the right processor to handle its appropriate workload.

- **Increased Performance**: +2.5x
- **Decreased Energy Per Frame**: -2.5x
ACCELERATING MEMCACHED

CLOUD SERVER WORKLOAD
A Distributed Memory Object Caching System Used in Cloud Servers

Generally used for short-term storage and caching, handling requests that would otherwise require database or file system accesses

Used by Facebook, YouTube, Twitter, Wikipedia, Flickr, and others

Effectively a large distributed hash table

- Responds to store and get requests received over the network
- Conceptually:
  - store(key, object)
  - object = get(key)
OFFLOADING MEMCACHED KEY LOOKUP TO THE GPU

Key Look Up Performance

<table>
<thead>
<tr>
<th></th>
<th>Data Transfer</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multithreaded CPU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radeon HD 5870</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Trinity&quot; A10-5800K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zacate E-350</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Execution Breakdown

- OFFLOADING MEMCACHED KEY LOOKUP TO THE GPU

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6189209
ACCELERATING B+TREE SEARCHES

CLOUD SERVER WORKLOAD
B+Tree Searches

- B+Trees are a fundamental data structure
  - Used to reduce memory & disk access to locate a key
  - Can support index- and range-based queries
  - Can be updated efficiently

- B+Trees are used by enterprise DB applications
  - SQL: SQLite, MySQL, Oracle, and others
  - No-SQL: Apache CouchDB, Tokyo Cabinet, and others
    - Audio search, video copy detection

A simple B+Tree linking the keys 1-7. The linked list (red) allows rapid in-order traversal.
By efficiently sharing data between CPU and GPU, HSA increases performance versus Multi Threaded CPU, even for tree structures that reside in host memory.

With HSA, DB can be larger than GPU memory, and can be shared.

- HSA lets us move compute to data
- Parallel search can move to GPU
- Sequential updates can remain on CPU

### Platform

<table>
<thead>
<tr>
<th>Platform</th>
<th>Size &lt; 1.5 GB</th>
<th>Size 1.5-2.7 GB</th>
<th>Size &gt; 2.7 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGPU (memory size = 3GB)</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>HSA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>


AMD A10 4600M APU with Radeon™ HD Graphics; CPU: 4 cores @ 2.3 MHz (turbo 3.2 GHz); GPU: AMD Radeon HD 7660G, 6 compute units, 685MHz; 4GB RAM.
ACCELERATING JAVA
GOING BEYOND NATIVE LANGUAGES
JAVA ENABLEMENT BY APARAPI

Aparapi = Runtime capable of converting Java™ bytecode to OpenCL™

Developer creates Java™ source

Source compiled to class files (bytecode) using standard compiler

For execution on any OpenCL™ 1.1+ capable device

OR execute via a thread pool if OpenCL™ is not available
JAVA AND APARAPI HSA ENABLEMENT ROADMAP
EASE OF PROGRAMMING

CODE COMPLEXITY VS. PERFORMANCE
BOLT

- Optimized template library routines for common GPU functions
  - For OpenCL™ and C++ AMP, across multiple platforms
- Programming model interface similar to multicore Task Parallel Runtimes (TBB, ConCRT)
- CPU performance as good or better than multicore Task Parallel Runtimes
- Excellent performance and power efficiency on HSA Devices
- For many applications, single source code base for both CPU and GPU!
- Leverage robust Visual Studio C++AMP debug solution
LINE-OF-CODE AND PERFORMANCE FOR DIFFERENT PROGRAMMING MODELS

(Exemplary ISV “Hessian” Kernel)

Algorithm
Launch
Copy-back    Algorithm
Launch
Copy-back
Algorithm
Launch
Copy-back
Algorithm
Launch
Copy-back
Algorithm
OpenCL™-C
OpenCL™-C++
C++ AMP
HSA Bolt
Serial CPU
TBB
Intrinsics+TBB

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

© Copyright 2012 HSA Foundation. All Rights Reserved.
## RESEARCH TOPICS IN HSA

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages/Compilers</td>
<td>Higher-level languages. GPU languages are primitive today. OpenCL is a good expert tool. Look into domain specific languages (graphics, math). Ex: HSA could have a database accelerator component</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Split compilation model – high level compilers &amp; low level compilers and how to make them work well together</td>
<td></td>
</tr>
<tr>
<td></td>
<td>How to run best on a device with multi ISA’s</td>
<td></td>
</tr>
<tr>
<td>Software Run-Time</td>
<td>Classic load balancing. Look for new ways to partition algorithms automatically in the runtime. Simultaneous running of multiple kernels or multiple applications. Quality of service &amp; virtualization. Scheduling for complex status graphs and scheduling dynamic parallelism</td>
<td></td>
</tr>
</tbody>
</table>
| System Architecture        | • Bandwidth/memory arch (balancing BW with compute)  
• Load balancing  
• Memory configurations: Stack memory devices will eventually appear and systems will change around idea of bandwidth. Shared memory stacks – what are the implications?  
• TCU/LCU ratios                                                                                           |          |
| Hardware                   | Logical split between split function hardware.  
• Applying HSA to non-GPU devices (DSPs, FPGAs, etc.)  
• Heterogeneous conformance optimization - how to run a program that runs well on all different HSA platforms and hardware  

Memory system design: low cost support for coherency and would give programmers a way to optimize their use of coherence                                                                                                                                                                                                 |          |
|                            | Security: looking into securing systems                                                                                                                                                                    |          |
|                            | Efficient synchronization primitives                                                                                                                                                                         |          |
|                            | 3D graphics pipes – integration with HSA                                                                                                                                                                   |          |
**THE HSA OPPORTUNITY FOR DEVELOPER**

**SOLUTION**
- HSA + Libraries = productivity & performance with low power
  - Few M HSA coders
  - Few 100Ks HSA apps
  - Wide range of differentiated experiences

**PROBLEM**
- Het. systems hard to program
  - Not all workloads accelerate
  - ~100K GPU coders
  - ~200 apps
  - Significant niche value

**Developer Return**
(Differentiation in performance, reduced power, features, time to market)

**Developer Investment**
(Effort, time, new skills)

*IDC