HETEROGENEOUS SYSTEM ARCHITECTURE (HSA) AND THE SOFTWARE ECOSYSTEM

MANJU HEGDE, CORPORATE VP, PRODUCTS GROUP, AMD
OUTLINE

Motivation

HSA architecture v1

Software stack

Workload analysis

Software Ecosystem
PARADIGM SHIFTS....

Single-Core Era

Enabled by:
✓ Moore’s Law
✓ Voltage Scaling

Constrained by:
✗ Power
✗ Complexity

Assembly ➔ C/C++ ➔ Java

Time ➔ Single-thread Performance

Single-thread Performance ➔ ?

we are here

Multi-Core Era

Enabled by:
✓ Moore’s Law
✓ SMP architecture

Constrained by:
✗ Power
✗ Parallel SW
✗ Scalability

pthreads ➔ OpenMP / TBB ...

Time ➔ Throughput Performance

we are here

Heterogeneous Systems Era

Enabled by:
✓ Abundant data parallelism
✓ Power efficient GPUs

Temporarily Constrained by:
✗ Programming models
✗ Comm. overhead

Shader ➔ CUDA ➔ OpenCL ➔ !!!

Time (Data-parallel exploitation) ➔ Modern Application Performance

we are here
WITNESS DISCRETE CPU AND DISCRETE GPU COMPUTE

- Compute acceleration works well for large offload
- Slow data transfer between CPU and GPU
- Expert programming necessary to take advantage of the GPU compute
First integration of CPU and GPU on-chip
Common physical memory but *not to programmer*
Faster transfer of data between CPU and GPU to enable more code to run on the GPU
COMMON PHYSICAL MEMORY BUT NOT TO PROGRAMMER

- CPU explicitly copies data to GPU memory
- GPU completes computation
- CPU explicitly copies result back to CPU memory
WHAT ARE THE PROBLEMS WE ARE TRYING TO SOLVE

- SOCs are quickly following into the same many CPU core bottlenecks of the PC
  - To move beyond this we need to look at right processor(s) and/or execution device for given workload at reasonable power

- While addressing the core issues of
  - Easier to program
  - Easier to optimize
  - Easier to load balance
  - High performance
  - Lower power
COMBINE INTO UNIFIED PROGRAMMING MODEL

- CPU
- GPU
- Audio Processor
- Video Hardware
- Encode Decode Engines
- Fixed Function Accelerator
- DSP
- Image Signal Processing
- Shared Memory, Coherency, User Mode Queues
WHO IS DOING THIS?
HSA FOUNDATION MEMBERSHIP – JUNE 2013

Founders
AMD | ARM | Imagination | MediaTek
Qualcomm | Samsung | Texas Instruments

Promoters
LG Electronics

Supporters
Arteris | codeplay | FABRICENGINE | MULTICOREWARE | SandiaNationalLaboratories

Contributors
Analog Devices | apical | CEVA | CANONICAL | OMP | ITRI | Marvell
Sony | Sonics | ST | Ericsson | Swarm64 | symbio

Academic
NTHU Programming Language Lab | NTHU System Software Lab | University of Bristol | University of Missouri | The University of Edinburgh informatics | University of Illinois Computer Science

Associates
National Taiwan University | Northeastern University | OleMiss | SGT
HSA FOUNDATION’S FOCUS

Identify design features to make accelerators first class processors

Attract mainstream programmers

Create a platform architecture for ALL accelerators
HSA ARCHITECTURE v1

- 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 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
WITH HSA

- CPU simply passes a pointer to GPU
- GPU completes computation
- CPU can read the result directly – no copying needed!
HSA ARCHITECTUREv1

- 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
- HSA JIT
- Task Queuing Libraries
- HSA Runtime
- HSA Kernel Mode Driver

Shared Memory

- CPU
- Audio Processor
- Video Hardware
- Encode/Decode
- Fixed Function Acctr
- DSP
- Image Signal Processing

Coherency, User Mode Queues
HETEROGENEOUS COMPUTE DISPATCH

How compute dispatch operates today in the **driver model**

How compute dispatch improves under HSA
TODAY’S COMMAND AND DISPATCH FLOW
TODAY’S COMMAND AND DISPATCH FLOW
TODAY’S COMMAND AND DISPATCH FLOW
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!
COMMAND AND DISPATCH CPU <-> GPU
MAKING GPUs AND APUS EASIER TO PROGRAM: TASK QUEUING RUNTIMES

- Popular pattern for task and data parallel programming on SMP systems today

- Characterized by:
  - A work queue per core
  - Runtime library that divides large loops into tasks and distributes to queues
  - A work stealing runtime that keeps the system balanced

- HSA is designed to extend this pattern to run on heterogeneous systems
TASK QUEUING RUNTIME ON CPUs
TASK QUEUING RUNTIME ON THE HSA PLATFORM
Driver Stack

HSA Software Stack

Hardware - APUs, CPUs, GPUs

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

HSA Domain Libraries, OpenCL™ 2.x Runtime

- HSA JIT
- Task Queuing Libraries
- HSA Runtime
- HSA Kernel Mode Driver
- Apps

User mode component
Kernel mode component
Components contributed by third parties
HSA INTERMEDIATE LANGUAGE - HSAIL

- HSAIL is the intermediate language for parallel compute in HSA
  - 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 a low level instruction set designed for parallel compute in a shared virtual memory environment. HSAIL is SIMT in form and does not dictate hardware microarchitecture

- HSAIL is designed for fast compile time, moving most optimizations to HL compiler

- 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
HSA BRINGS A MODERN OPEN COMPILATION FOUNDATION

- This brings about fully competitive rich complete compilation stack architecture for the creation of a broader set of GPU Computing tools, languages and libraries.
  - HSAIL supports LLVM and other compilers – GCC, Java VM
OPENCL™ AND HSA

- HSA is an optimized platform architecture for OpenCL™
  - Not an alternative to OpenCL™
  - Focused on the hardware platform more than API
  - Ready to support many more languages than C/C++

- 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
  - Optimized libraries may choose the lower level interface
HSA DELIVERED VIA ROYALTY FREE STANDARDS

- Royalty Free IP, Specifications and API’s

- Three primary specifications are
  - HSA Platform System Architecture Specification
    - Focus on hardware requirements and low level system software
  - HSA Programmer Reference Manual
    - Definition of HSAIL Virtual ISA
    - Binary format (BRIG)
    - Compiler writers guide and Libraries developer guide
  - HSA System Runtime Specification
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

<table>
<thead>
<tr>
<th>Component Name</th>
<th>AMD Specific</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSA Bolt Library</td>
<td>No</td>
<td>Enable understanding and debug</td>
</tr>
<tr>
<td>HSAIL Code Generator</td>
<td>No</td>
<td>Enable research</td>
</tr>
<tr>
<td>LLVM Contributions</td>
<td>No</td>
<td>Industry and academic collaboration</td>
</tr>
<tr>
<td>HSA Assembler</td>
<td>No</td>
<td>Enable understanding and debug</td>
</tr>
<tr>
<td>HSA Runtime</td>
<td>No</td>
<td>Standardize on a single runtime</td>
</tr>
<tr>
<td>HSA Finalizer</td>
<td>Yes</td>
<td>Enable research and debug</td>
</tr>
<tr>
<td>HSA Kernel Driver</td>
<td>Yes</td>
<td>For inclusion in linux distros</td>
</tr>
</tbody>
</table>
WORKLOAD ANALYSIS
HAAR Face Detection

CORNERSTONE TECHNOLOGY
FOR COMPUTERVISION
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

Feature k
Feature l
Feature m
Stage N
Feature p
Feature r
Feature q
Stage N+1
Face still possible?
Yes
No
REJECT FRAME
22 CASCADE STAGES, EARLY OUT BETWEEN EACH

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
CASCADE DEPTH ANALYSIS

Cascade Depth

- 20-25
- 15-20
- 10-15
- 5-10
- 0-5

Image of 'Mona Lisa' with a focus area highlighted.
PROCESSING TIME/STAGE

“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)
“Trinity” A10-4600M (6CU@497Mhz, 4 cores@2700Mhz)

PERFORMANCE CPU-VS-GPU

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)
HAAR SOLUTION – RUN DIFFERENT CASCADES ON GPU AND CPU

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

40
GAMEPLAY RIGID BODY PHYSICS
RIGID BODY PHYSICS SIMULATION

- Rigid-Body Physics Simulation is:
  - a way to animate and interact with objects, widely used in games and movie production
  - used to drive game play and for visual effects (eye candy)

- Physics Simulation is used in many of today’s software:
  - Middleware Physics engines such as Bullet, Havok, PhysX
  - Games ranging from Angry Birds and Cut the Rope to Tomb Raider and Crysis 3
  - 3D authoring tools such as Autodesk Maya, Unity 3D, Houdini, Cinema 4D, Lightwave
  - Industrial applications such as Siemens NX8 Mechatronics Concept Design
  - Medical applications such as surgery trainers
  - Robotics simulation

- But GPU-accelerated rigid-body physics is not used in game play - only in effects
RIGID BODY PHYSICS - ALGORITHM

- Find potential interacting object “pairs” using bounding shape approximations.
- Perform full overlapping between potentially interacting pairs
- Compute exact contact information for various shape types
- Compute constraint forces for natural motion and stable stacking

<table>
<thead>
<tr>
<th>A</th>
<th>B0</th>
<th>B1</th>
<th>C0</th>
<th>C1</th>
<th>D1</th>
<th>D1</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
## Implementation Challenges

- Game engine and Physics engine need to interact synchronously during simulation
  - Ray-casting queries, as well as synchronous narrow-phase, constraint and collision callbacks require fast CPU round-trips and CPU modification of simulation state mid-pipeline
  - Traditional GPU solutions cannot guarantee frame-time response

- The set of pairs can be huge and changes from frame to frame
  - E.g. Thousands to Millions for any given frame

## Benefits of HSA

- Fast CPU round-trips
  - USD

- Immediate access to geometry and modification of simulation state mid-pipeline
  - SMA, COH

- Supports as large pair list as CPU
  - EMS

- GPU can resize pair list without CPU interaction overhead
  - DYN

---

**EMS**: Entire Memory Space; **PM**: Pageable Memory; **COH**: Bidirectional Coherency

**SMA**: System Memory Access; **DYN**: Dynamic Memory Allocation;

**ENQ**: GPU ENQueue; **USD**: User Mode Dispatch
## RIGID BODY PHYSICS - CHALLENGES & SOLUTIONS

<table>
<thead>
<tr>
<th>Implementation Challenges</th>
<th>Benefits of HSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation is a pipeline of many different algorithms, some of which are more suitable for CPU while others are more suitable for GPU</td>
<td>Avoidance of the data copy to/from GPU and of the overhead of maintaining two copies of simulation state</td>
</tr>
<tr>
<td>Many CPU optimizations (eg. “early outs”) aren’t efficient on GPUs, requiring the use of more brute-force but GPU-friendly algorithms</td>
<td>- SMA, COH</td>
</tr>
<tr>
<td>Diversity of intersection algorithms cause load balancing challenges</td>
<td>Usage of “early out” optimizations and more efficient load balancing</td>
</tr>
<tr>
<td>Varying object sizes require more complex and difficult to parallelize broad-phase algorithms</td>
<td>- ENQ</td>
</tr>
<tr>
<td>“sweep-and-prune” uses incremental sorting and traversal of lists</td>
<td>More efficient serial aspects of broad-phase can run on the CPU</td>
</tr>
<tr>
<td>Narrow-phase algorithms (such as SAT or GJK) cause thread divergence</td>
<td>- SMA, COH</td>
</tr>
<tr>
<td></td>
<td>Improved handling of thread divergence</td>
</tr>
<tr>
<td></td>
<td>- ENQ</td>
</tr>
</tbody>
</table>

**EMS**: Entire Memory Space; **PM**: Pageable Memory; **COH**: Bidirectional Platform Coherency
**SMA**: Shared Virtual Memory; **DYN**: Dynamic Memory Allocation; **ENQ**: GPU ENQueue;
**USD**: User Mode Dispatch
GESTURE RECOGNITION
GESTURE RECOGNITION

- An emerging natural way of interacting with a computer
- Compute intensive where the computational complexity depends on the number and complexity of recognized gestures.
- Strongly benefits from availability of depth information
- Browsing (previous/next, scroll), media players (next/previous song/video/image, pause/start), collaboration tools, such as slideshows, gaming (finger/hand as the controller), immersive environments, virtual reality

- Today’s systems are tuned to today’s HW, lacking in robustness and usability, which can only be achieved by use of special-purpose HW. They do not do well for
  - A wide variety of useful gestures (one or two hand, multiple finger, arm or full body)
  - Motion dependent gestures (e.g. finger pinch), which requires correlating information from multiple frames
  - Adaptability to variable lighting conditions
  - Larger region/distance of input, enabled by processing higher resolution video
ALGORITHM PIPELINE

- Image processing:
  - adaptive light normalization
  - Edge and corner detection
  - Erode/dilate/threshold filter, to produce a feature image.
- Depth analysis (for fg/bg segmentation, if using stereo cameras)
  - Sparse approach, correlate salient points in the feature image, and validate via local histogram matching in the original image.
- Connected components analysis, for hand identification (based on level sets)
  - GPU can recognize local connectivity with a parallel scan. CPU can apply transitivity of labels (the neighbor of your neighbor is your neighbor).
- Feature vector (local histogram) extraction
  - Global: HOG on tiles; or
  - Contextual: SURF/SIFT keypoints
- Find best match of histogram, with the training set (support vector machine), optionally update the training set.
- Update temporal model state machine
GESTURE RECOGNITION – CHALLENGES AND SOLUTIONS

**Implementation Challenges**

- Transfer of raw image data from CPU to GPU adds latency
- Feature matching and depth reconstruction is a divergent workload, as images are sparsely populated by keypoints, which require extensive processing.
  Connected component analysis on GPU uses parallel scan, of which the last stages of reduction are more efficiently performed on the CPU.
- High overhead of the per-frame updates to the GPU copy of the feature database, for unsupervised learning algorithms (e.g. Oja’s rule).

**Benefits of HSA**

- Avoidance the latency of duplicating data in GPU memory – SMA
- Higher GPU utilization is achieved via wavefront reshaping - ENQ
- Reduction is most optimally implemented by using both CPU and GPU - COH, SMA
- CPU can update the database, while the GPU is accessing it –SMA, COH

**Abbreviations**

- EMS: Entire Memory Space; PM: Pageable Memory; COH: Bidirectional Platform Coherency
- SMA: Shared Virtual Memory; DYN: Dynamic Memory Allocation; ENQ: GPU ENQueue;
- USD: User Mode Dispatch
RAY TRACING
RAY TRACING

- Photo-realistic visualization method that is widely used in movie production and high-fidelity visual effects
- Used in many of today's photorealistic rendering packages
  - Maxwell Render (photorealistic high-end renderer)
  - Nvidia’s Optix (Nvidia GPU ray tracing renderer)
  - POV-Ray (popular CPU-only ray tracer)
  - Luxmark (popular ray tracing benchmark)
- Rendering method that is friendly to parallelism, however not trivially ported to parallel architectures, due to the complexity of an efficient implementation.
- However it is not used in interactive applications due to performance limitations
RAY TRACING - ALGORITHM

- Rays are being traced from the eye to the scene and intersections are tracked.

- Many subsequent child (reflected or refracted) rays are traced, until a limit is reached.
  - The scene are usually complex, so we have to build an acceleration data structure to speed-up ray-object intersections.
  - This is usually the most compute intensive part of the algorithm.

- Each generated ray is subsequently colored based on a shading computation, final color is accumulated for each pixel.

- Problem scales to the full frame with 100Ks of primary rays and millions of total rays.
## Implementation Challenges

- Scene database and acceleration data structure can be huge
  - Eg. A “power plant” scene (shown left) contains 12.7M polygons, has a size of 500MBytes, and an acceleration data structure of 250MB-1.5GB (depending on renderer)
  - Today’s GPUs have problems fitting them into video memory
- Acceleration data structure has to be built and updated using the CPU and transferred to video memory
  - 8ms time to transfer above data structure (250MB) to the GPU

## Benefits of HSA

- GPU Compute Units can access scene and acceleration data structure from main memory
  - SMA, PM
- Avoidance of acceleration data structure copy to GPU memory
  - SMA

---

**EMS**: Entire Memory Space; **PM**: Pageable Memory; **COH**: Bidirectional Platform Coherency
**SMA**: Shared Virtual Memory; **DYN**: Dynamic Memory Allocation; **ENQ**: GPU ENQueue;
**USD**: User Mode Dispatch
# RAY TRACING - CHALLENGES & SOLUTIONS

<table>
<thead>
<tr>
<th>Implementation Challenges</th>
<th>Benefits of HSA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dynamic Scenes</strong> are impractical with current GPU compute implementations</td>
<td>CPU updates to scene are transparently and immediately available (without any transfer penalty) to the GPU</td>
</tr>
<tr>
<td>Data structure build time too long for interactive frame rates</td>
<td>– SMA, PM</td>
</tr>
<tr>
<td>Simple data structures can be built fast, but are difficult to traverse</td>
<td>Casting of child rays with no CPU-GPU round trip</td>
</tr>
<tr>
<td>Faster traversal requires complex structures that require a long time to compute and are difficult to transfer to the GPU</td>
<td>– ENQ</td>
</tr>
<tr>
<td>Ray divergence caused by child rays hitting different object types with different shading models (both GPUs &amp; APUs like regular operations) results in lower utilization of CUs</td>
<td>Wavefront reshaping can improve CU utilization</td>
</tr>
<tr>
<td>The amount of rays can be immense (in the billions), and the ray intersection process is compute intensive</td>
<td>– ENQ</td>
</tr>
<tr>
<td>“power plant” scene at 1080p conservative est. 2 billion rays.</td>
<td></td>
</tr>
</tbody>
</table>

**EMS**: Entire Memory Space; **PGM**: Pageable Memory; **COH**: Bidirectional Coherency; **SMA**: System Memory Access; **DYN**: Dynamic Memory Allocation; **ENQ**: GPU ENQueue; **USD**: User Mode Dispatch
ACCELERATING MEMCACHED

CLOUD SERVER WORKLOAD
MEMCACHED

- 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

GPU PROGRAMMING OPTIONS FOR JAVA™ PROGRAMMERS

- Existing Java™ GPU (OpenCL™/CUDA™) bindings require coding a ‘Kernel’ in a domain-specific language.

```java
// JOCL/OpenCL kernel code
__kernel void squares(__global const float *in, __global float *out){
    int gid = get_global_id(0);
    out[gid] = in[gid] * in[gid];
}
```

- Along with the Java ‘host’ code to:
  - Initialize the data
  - Select/Initialize execution device
  - Allocate or define memory buffers for args/parameters
  - Compile 'Kernel' for a selected device
  - Enqueue/Send arg buffers to device
  - Execute the kernel
  - Read results buffers back from the device
  - Cleanup (remove buffers/queues/device handles)
  - Use the results
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
WHAT IS APARAPI?

- **At development time**
  - Aparapi offers an API for expressing data parallel workloads in Java™
    - Developer uses common Java patterns and idioms
      - extend Kernel base class and implements `run()` method
    - Java source compiled to (bytecode) using standard compiler (javac)
    - Classes packaged and deployed using traditional Java tool chain

- **At runtime**
  - Aparapi offers a runtime capable of converting bytecode to OpenCL™
    - For execution on GPU/APU (or any OpenCL 1.1+ capable device)
    - OR execute via a thread pool if OpenCL is not available
JAVA AND APARAPI HSA ENABLEMENT ROADMAP
GOALS FOR HSA

DEVELOPER: Easier to program
- Expressive runtime for rich high level programming models
- Unified address space with Dynamic Memory Allocation
- Single Source for all processors on the SOC

DEVELOPER: Improved performance & power
- Reduced Kernel Launch Time
- Efficient CPU & GPU Communication
- Pass Pointers rather than move memory

OSV: Improved quality of service
- Support for Multiple Concurrent GPU processes
- Preemptive Multitasking of CPU/GPU resources
- Support for Shared Virtual Memory with paging support

ENDUSER: Rich Experiences
- Advanced Natural User Interfaces & Presence Capabilities
- Rich Cloud Computing User Experiences
- Perceptual Computing Problems
- Bring Hollywood Class Realism to Real-time Entertainment
INITIAL OPEN SOURCE TARGETS

- x264
- Handbrake
- FFMPEG
- JPEG
- VLC
- OpenCV
- GIMP
- ImageMagick
- IrfanView
- Hadoop, Memcached
- Aparapi – A parallel API (for Java)
- Bolt – a Unified Heterogeneous Library
- Crypto++
- Bullet physics library
- … + Search for “OpenCL” on Sourceforge, Github, Google Code, BitBucket finds over 2000 projects
OPENCL ON GOOGLE SCHOLAR IS GROWING RAPIDLY

http://developer.amd.com/Resources/library/Pages/default.aspx
ACADEMIC TRACTION

- Over 100 Universities teaching multi-faceted heterogeneous computing programming courses Worldwide
- Growing textbook ecosystem
- Including AMD supported books
  - OpenCL textbook (Morgan Kaufmann)
  - OpenCL Programming Guide (Addison Wesley)
- Complete University Kit available including:
  - OpenCL textbooks – US, India, & China
  - OpenCL presentation w/instructor & speaker notes, example code, & sample application
- Research projects with Top-tier Universities globally
If we build it will they come???
CUDA BROUGHT PERFORMANCE TO PRO/RESEARCH ON DISCRETE GPU

CUDA gave 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
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*)
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>4.5M developers</td>
<td>6M developers</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td>4.5M developers</td>
<td>6M developers</td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td>4.5M developers</td>
<td>6M developers</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>4.5M developers</td>
<td>6M developers</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>4.5M developers</td>
<td>6M developers</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>4.5M developers</td>
<td>6M developers</td>
<td></td>
</tr>
</tbody>
</table>

Java 7
10M+ developers
Millions of Apps
You can get developers to change!

(takes time and strategy)
THE HSA OPPORTUNITY

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

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

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

**PROBLEM**
- Historically, developers program CPUs
  - ~10+M* CPU coders
  - ~4M apps
  - Good user experiences

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

*IDC
Come to: **AMD Developer Summit – APU13**

The epicenter of heterogeneous compute

When: Nov 11 – 14, 2013
Where: San Jose, CA | McEnery Convention Center

- Over 120 Individual Presentations in 12 Different Tracks
- Keynotes from industry thought-leaders, including:
  - Lisa Su, general manager, Global Business Units - AMD
  - Mark Papermaster, senior vice president & chief technology officer - AMD
  - Phil Rogers, corporate fellow - AMD
  - Mike Muller, CTO - ARM
  - Johan Andersson, Chief Architect - DICE
  - Tony King-Smith, Executive Vice President, Marketing - Imagination Technologies
  - Chienping Lu, Senior Director - Mediatek USA
  - Nandini Ramani, Vice President of Development - Oracle Solutions
  - David Helgason, Founder & CEO - Unity Technologies

For more information and registration visit [http://developer.amd.com/apu](http://developer.amd.com/apu)
Thank you