Heterogeneous Systems Architecture - memory sharing and task dispatching

Want to get an overview of what Heterogeneous Systems Architecture (HSA) does, or want to know what terminology has changed since version 1.0? Read further.

Back in 2012 the goals for HSA were high. The group tried to design a system where CPU and GPU would work together in an efficient way. In the 2013/2014 time-frame you'll find lots of articles around the web, including on our blog, describing the capabilities of HSA. Unfortunately with the 1.0 specifications most terminologies have been changed.

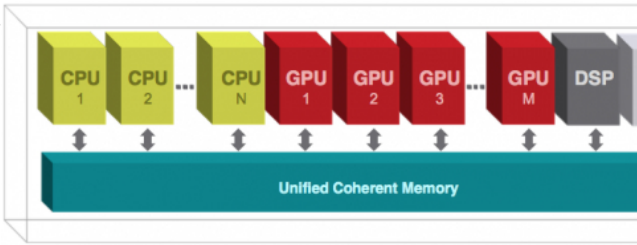
In March 2015 the HSA Foundation released the final 1.0 specifications. It does not discuss hUMA (Heterogeneous Uniform Memory Access) nor hQ (Heterogeneous Queuing). These two techniques had undergone so many updates, that new terminologies were used.

In this blog post, we'll present you an updated description of the two most important problems tackled by HSA: **memory sharing** and **task dispatching**.

We'll be tuning the below description, so feedback is always welcome - focus is on clarity, not on completeness.

What is an HSA System?

Where the original HSA goals focused more on SoCs with CPU and GPU cores, now any compute core can be used. The reason was that modern SoCs are much more complex than just a CPU and GPU - integrated DSPs and video-decoder are found on many processors. HSA thus now (officially) supports truly heterogeneous architectures.



The idea is that any heterogeneous processor can be designed by the principles of HSA. This will bring down design costs and enable more exotic configurations from different vendors.

And interesting fact about the HSA-specifications is that it only specifies goals, not how it must be implemented. This makes it possible to implement the specifications in software instead of hardware, making it possible to upgrade older hardware to HSA.

Why is HSA important?

A simple question: "will there be more CPUs with embedded GPU or discrete GPUs?". A simple answer: "there are already more

integrated GPUs than discrete ones". HSA defines those chips with mixed processors.

CPUs with embedded GPUs used to be not much more than the discrete GPUs with shared memory we know from cheap laptops in the 00's. When the GPU got integrated, each vendor started to create solutions for inter-processor dispatching (threading extended to heterogeneous computing), course-grained sharing (transferring ownership between processor units) and fine grained sharing (atomics working with all processor units).

The HSA Foundation

Sometimes an industry makes bigger steps by competing and sometimes by collaborating
AMD recognised the need for a standard. As AMD wanted to avoid the problems with introducing 64 bit into X86 and therefore initiated the HSA foundation. The founding members are AMD, ARM, Imagination Technologies, MediaTek, Qualcomm, Samsung and Texas Instruments. NVidia and Intel are awkwardly absent.

Memory Sharing

HSA uses a relaxed memory model, which has full memory coherence (data guaranteed to be the same for all processes on all cores) and is pageable (subsets can be reserved by programs).

The below write-up is heavily simplified to give an overview how memory sharing is designed under HSA. If you want to know more, read chapter 5 from the HSA book.

Inter-processor memory-pointer sharing - Unified Addressing

The most important part is the unified memory model (previously referred to as "hUMA"), which makes programming the memory-interactions in a heterogeneous processor with CPU-cores, GPU-cores and DPS-cores comparable to a multi-core CPU.

Like other modern memory models, HSA defines various segments, including global, shared and private. A difference is that flat addressing is used. This means that each address pointer is unique: you don't have an address 0 for private and an address 0 for global. Flat addressing simplifies optimisation operations for higher level languages. Ofcourse you still need to be aware that each segment size is limited and there will be consequences when defining larger memory chunks than is available in the segment.

When you have created a memory object and want the DSP or GPU continue to work on it, then you can use the same pointers without any translations.

Inter-processor cache coherency

In HSA-systems global memory is coherent without the need for explicit cache maintenance. This means that local caches are synchronised and/or that caches are shared. For more information, read this blog from ARM.

Fine grained memory - Atomic Operations

HSA allows protecting memory segments to be atomicly accessed. This makes it possible to have multiple threads running on different cores of different processor units, all accessing the same memory in a safe manner.

Small and large consequtive memory segments can be reserved for sharing, from very fine to coarse grained. All threads that have access to that segement are notified when atomic operations are done.
Fine Grained Shared Virtual Memory (HSA compatibility for discrete GPUs)
AMD has done some efforts to extend HSA to discrete GPUs. We'll see the real advantages with dispatching, but it also works to create a cleaner memory management.

The so called "Fine Grained Shared Virtual memory" makes it possible use HSA with discrete GPUs that have HSA-support. Because it's virtual and data is continuously transferred between GPU and the HSA-processor, the performance is ofcourse lower than when using real shared memory. You can compare it to NVidia's Unified Virtual Memory, and it also has been planned to be in

OpenCL 2.0 for a long time.

Dispatching

HSA defines in detail how a task gets into the queue of a worker thread. Below is an overview of how queues, threads and tasks are defined and are named under HSA.

Queueing

Before HSA 1.0 we only spoke of "Heterogeneous Queue" (hQ). This is now further developed to "User Mode Queues". A User Mode Queue holds the list of tasks for that specific (group of) processor cores, resides in the shared memory and is allocated at runtime.

Such task is described in a language called "Architected Queueing Language" (AQL), and is called an "AQL package".

Agents and Kernel Agents

HSA threads run on one or a group of processor cores. These threads are called "Agents" and come in two variations: normal Agents and Kernel Agents. A Kernel Agents is an Agent that has a User Mode Queue and can execute kernels that work on a segment of memory. A normal Agent doesn't have a queue and can only execute simple tasks.

If a normal agent cannot run kernels, but can run tasks, then what can it actually do? Here are a few examples:

- Allocate memory, or other tasks only the host can do.
- Send back (intermediate) data to the host - for example progress indication.

If you compare to OpenCL, an agent is the host (which creates the work) and kernel agents are the kernels (which can issue new threads under OpenCL 2.0).

AQL packages: communicating dispatch tasks

There are different types of the AQL (Architected Queueing Language) packets, of which these are the most important:

- Agent dispatch packet: contains jobs for normal agents.
- Kernel dispatch packet: contains jobs for kernel agents.
- Vendor-specific packet: between processors of the same vendor there can be more freedoms.

In most cases, we'll be talking about kernel dispatch packages.

The Doorbell signal: low latency dispatching

HSA dispatching is extremely fast and power-efficient due to the implementation of a "doorbell". The doorbell of an agent is signalled when a new tasks is available, making it possible to take immediate action. A problem in OpenCL is the high dispatch times for GPUs without a doorbell - up to the millisecond range, as we have measured. For HSA-enabled GPUs the response-time before a kernel starts running is in the microseconds range.

Context switching

Threads can move from one core to another core - the task will be removed from the current queue and added to another queue. This can even happen when the thread is in running state.

StreamHPC's position

The solution simply works and makes faster code - we have done a large project with it last year.

It seems that almost the whole embedded processor industry believes in it. AMD (CPU+GPU), ARM (CPU+GPU), Imagination (GPU), Mediatek, Qualcomm (GPU), Samsung and Texas Instruments (DSP) are founders. Companies like Analog Devices, CEVA, Sony, VIA, S3, Marvell and Cadence have later joined the club. Important Linux clubs like Linaro and Canonical are also seen.

The system-on-a-chip only will get more traction, and we see HSA as an enabler. Languages like OpenCL and OpenMP can be compiled down to HSA, so it just takes switching the compiler. HSA-capable software can be written in a more efficient manner, as now can be assumed that memory can efficiently be shared and dispatching new threads is really fast.