HSA: A New Architecture for Heterogeneous Computing

Whitepaper Sponsored by AMD
January 28, 2013

The Heterogeneous Computing Dilemma

Moore's Law[1] has enabled the semiconductor industry to a point where an entire system can be placed on a chip, including memory, I/O, high-speed networks, networking interfaces, and various dedicated and programmable logic components. As a result, the performance and complexity of the processors (or System-on-chip (SoCs) as a more accurate definition) and each of the functional blocks continues to increase, translating into increased complexity in programming models. Although programmers typically program to the Central Processing Unit (CPU), increasingly other logic blocks, such as Graphics Processing Units (GPUs), Digital Signal Processors (DSPs), Field Programmable Gate Arrays (FPGAs), and other programmable logic units are being utilized as accelerators. For example, the parallel computation engines of GPU cores can be used to execute highly parallel tasks in a fraction of the power and time required to execute the same functions on CPUs making them well suited as coprocessors or accelerators. However, programming these heterogeneous solutions has become a challenge for all but expert, or "ninja", programmers.

Programming heterogeneous solutions is a challenge because each type of compute unit (CPUs, GPUs, DSPs, etc.) has a different execution model due to differing instruction set architectures (ISAs) and separate memory spaces. The current programming model calls for an Application Programming Interface (API) approach to programming and execution that is designed around the CPU and operating system. This requires all functions, even if intended for an accelerator, to be delegated by the CPU and executed through the operating system and existing software layers. In addition, most SoCs have separate physical and/or virtual memory spaces for each type of compute unit that often vary in size from the system main memory requiring complex memory mapping and data transfers. Programmers must account for and maintain the status of memory contents when switching between different compute units such as CPUs and GPUs. While some heterogeneous programing toolsets like CUDA and OpenCL have been developed, they require detailed knowledge of the hardware architecture, learning new programming languages and tools, and continuous management of memory contents. As a result, only a small percentage of applications and programmers have adopted these programming models for heterogeneous computing.

To address these challenges, AMD, ARM, Imagination, MediaTek, Qualcomm, Samsung, and Texas Instruments (TI) formed the Heterogeneous System Architecture (HSA) Foundation.

[1] Moore's Law is the economic principle that has driven the semiconductor industry for over 40 years. Each doubling of transistors per mm^2 has allowed for continued integration and performance scaling.

The Goal of the HSA Foundation is four-fold:

- To enable power-efficient performance
- To improve programmability of heterogeneous processors
- To increase the portability of code across processors and platforms
- To increase the pervasiveness of heterogeneous solutions throughout the industry

Power-Efficient Performance

The first of HSA's goals is to increase performance by reducing redundancy in the system design and execution, and improving the utilization of system resources. In many cases, the accelerators can improve the performance of many tasks at a fraction of the power. The first area of improvement is in the allocation and execution of tasks on accelerators that can be accomplished more efficiently than on CPUs. The second area is to reduce unnecessary memory traffic that results from having separate and varying sizes of memory structures for each type of compute unit. Current protocols require the flushing and copying of memory contents from one memory structure to another, in particular, when delegating tasks to accelerators, such as GPUs and DSPs.

Programmability

In terms of programmability, HSA's goal is to eliminate the difficulty of programming with low-level languages and the fragmentation associated with different programming environments for each hardware solution. Despite having some power-efficiency and performance advantages, current SoC solutions offer end users only narrow and limited utilization of these benefits since they require expert programming, which does not scale well. The programmability goal of HSA addresses this bottleneck by targeting common hardware and software specifications that enable the development of a common intermediate language, and programming tools. This commonality enables the use of common higher-level languages and existing heterogeneous programming languages like C++, C++ AMP, C#, Java, OpenCL, OpenMP, and Python.

Portability

Through common specs, tools, and an intermediate compiler target language called the Heterogeneous System Architecture Intermediate Language (HSAIL), the HSA's goal is to improve the portability of code across all HSA-compliant hardware solutions. This allows high level compilers like C++ and Java to generate a single output form that will execute across a range of HSA devices and vendors. In turn, this allows programmers to develop code for one platform that would run on other platforms without requiring recompilation of the application. As part of this goal, HSA seeks to make the hardware differences between platforms as

transparent to the programmer as possible, thus providing a large install base to motivate developer adoption.

Pervasiveness

The HSA Foundation is encouraging support by inviting all relevant organizations to join the HSA Foundation and by making the HSA software stack an open source solution that can be used by anyone. The HSA Foundation also encourages the inclusion of intellectual property (IP) by the participating entities to foster an open environment.

The HSA Solution

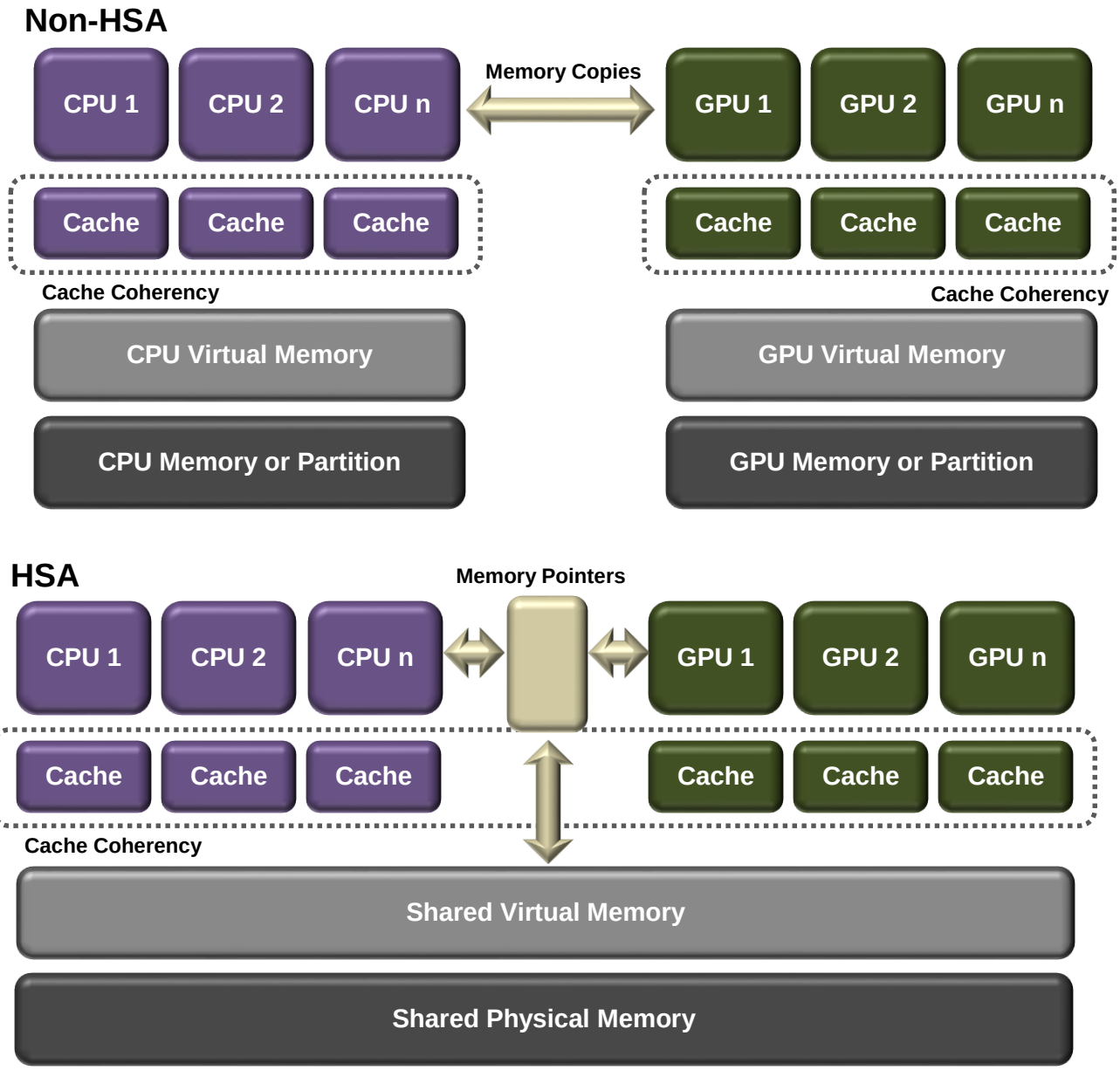
Unlike other heterogeneous programming standards, HSA is a platform standard that includes both hardware and software specifications. Hardware vendors must not only meet the hardware requirements, but must submit and support runtime libraries for their solutions. To achieve the goals outlined above, there are four key components to HSA:

- Memory architecture
- User mode scheduling
- Preemption and context switching
- Support for programming languages and the HSA Intermediate Language

Memory Architecture

The most critical hardware element of the HSA solution is a Shared Coherent Virtual Memory (SCVM) architecture The SCVM includes a single virtual memory accessed by all the compute units, allowing them to use the same data through the same addresses. The SCVM also includes shared page tables that allow pointers to the memory to be passed between compute units. Together, the shared memory and page tables eliminate the copying of memory content from one memory subsystem to another, flushing of the memory content to ensure coherency of the data, and the complex memory mapping required by the programmer on legacy heterogeneous systems. Because non-HSA accelerators often have smaller memory sizes than the system main memory, using a single memory architecture also eliminates the challenges of managing frequent memory transfers between memory subsystems of varying sizes and the overhead associated with the additional memory transfers. Note that compute units may still have banks of dedicated physical memory, but all shared data must use the shared virtual memory.

Figure 1. Shift to a Shared Coherent Virtual Memory Architecture (SCVM)



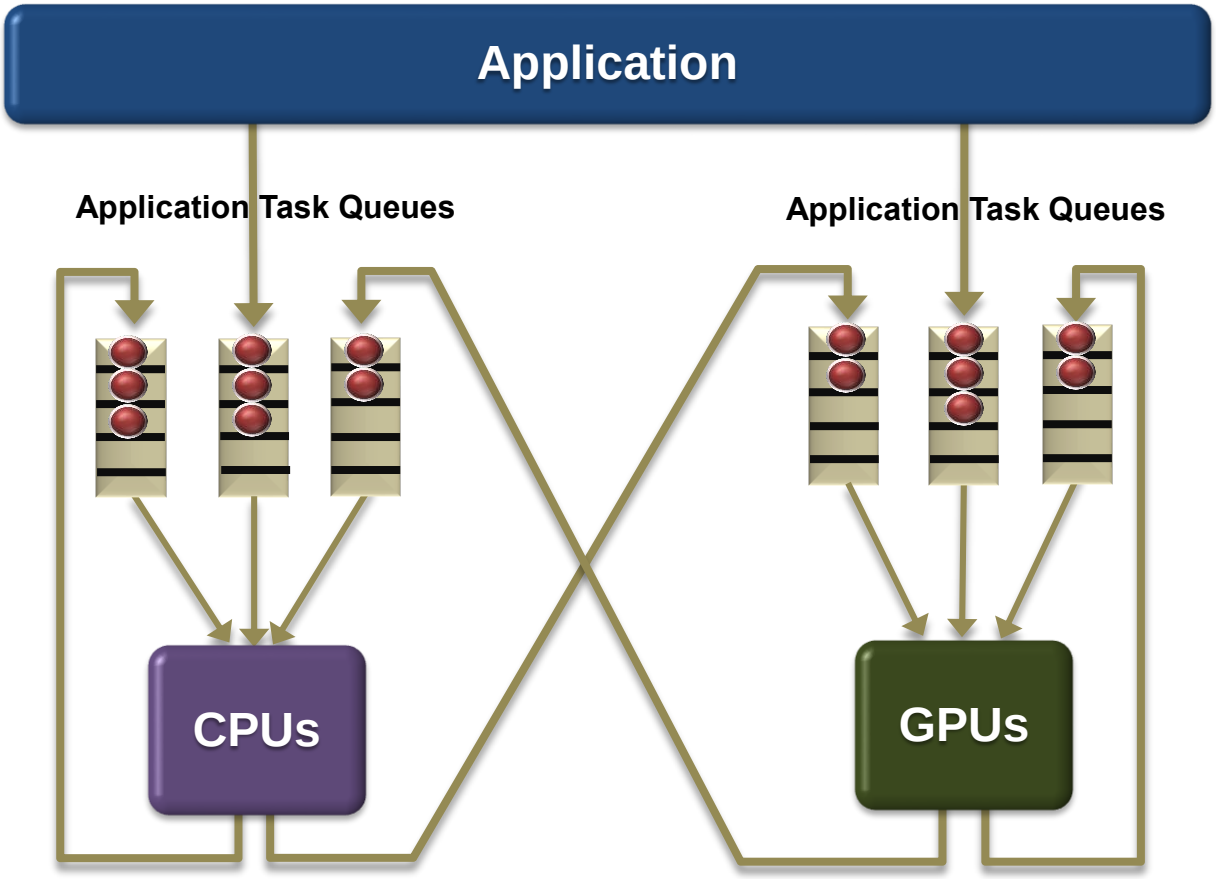
Source: HSA Foundation, 7/13

Although having shared main memory eliminates many of the challenges in memory management, each compute unit and/or group of compute units may still have independent cache structures. To address this, the HSA specification calls for cache coherency between all the compute units. The requirement of cache coherency will eliminate the need for data copying and pointer re-mapping. This also allows for differentiated hardware solutions to evolve and compete while remaining conformant to programming standards that enable developer and end-user compatibility.

User-Mode Scheduling

Another key efficiency in HSA is support for User-Mode Scheduling (UMS) to allow applications to schedule directly to the desired compute unit, including accelerators. This avoids the use of a kernel scheduler, such as the operating system, that intermixes threads from different applications. In addition, HSA calls for the scheduling of tasks by any compute unit to other compute units, including the accelerators. This is a change from the CPU-centric model that requires all scheduling and tasks to be scheduled through the CPU and operating system. To accomplish UMS, HSA allows for the addition of a hardware scheduler that operates in conjunction with the kernel scheduler.

Figure 2. User-Mode Scheduling between all Compute Units



Source: AMD, 10/13

Preemption and Context Switching

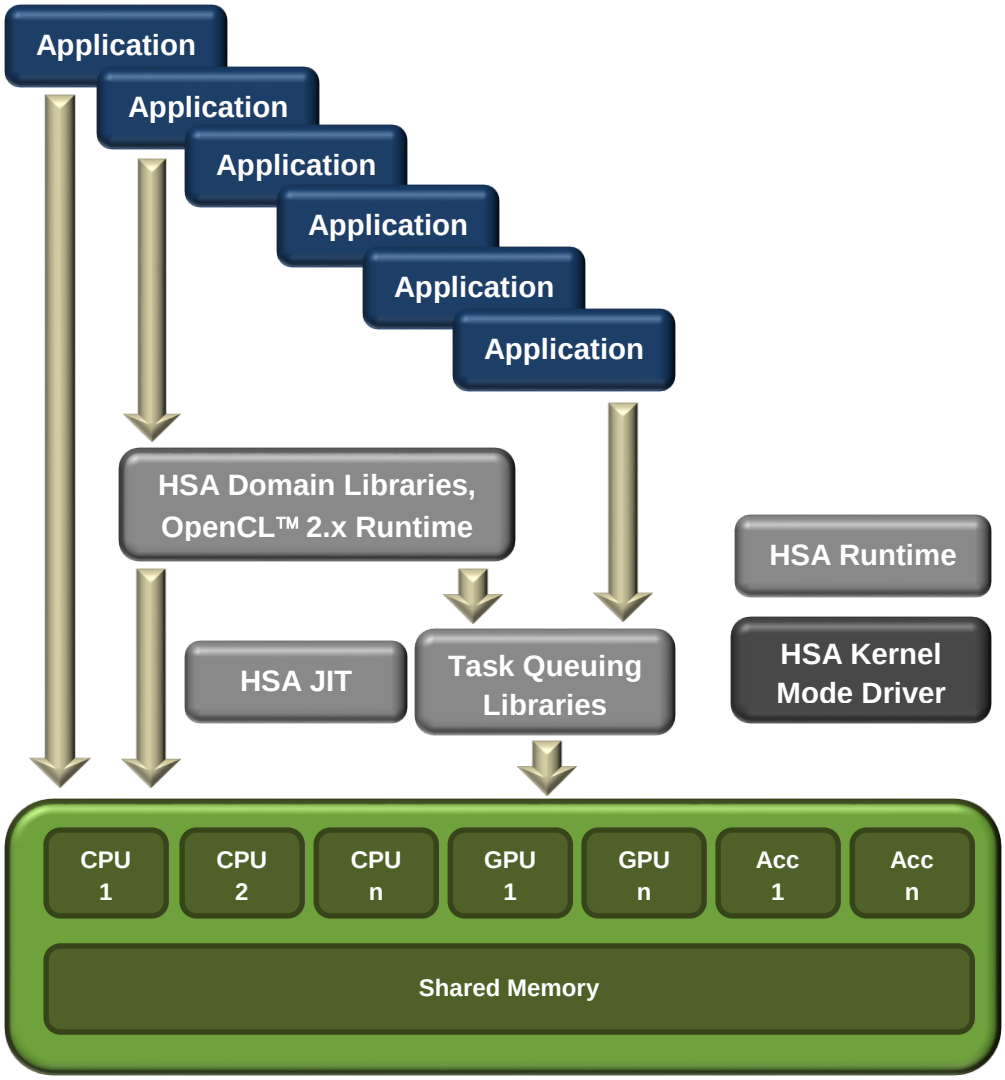
In addition to the UMS model, HSA also requires support for task preemption and context switching by the operating system for time slicing, which ensures quality of service on the machine and that all processes can make progress. The preemption and context switching capability ensures the state of each process is saved prior to switching to a new task and restored prior to the continued execution of the original task.

Programming Language and HSAIL Support

The final critical element of HSA is to support existing higher-level programming languages to allow for utilization of heterogeneous computing by all programmers. This requires masking the differences in hardware solutions while optimizing the applications to run in the most efficient

manner on any hardware platform, including non-HSA supported platforms. The primary means to enable the HSA software environment will be the Heterogeneous System Architecture Intermediate Layer (HSAIL) and the Finalizer. HSAIL provides a standard target for higher level language compilers and tools allowing programs to run on any HSA platform. The programming tools under development will determine which functions or tasks should be allocated to accelerators and compile them to HSAIL. The Finalizer will then convert the HSAIL code to the accelerator's native ISA at the point of execution.

Figure 3. HSA Programming Model



Source: HSA Foundation, 7/13

The goal is to allow programmers to use existing languages without being an expert in more specialized low-level languages like OpenCL. This will allow even novice programmers the ability to effectively program heterogeneous solutions in an efficient manner. However, for those that need or choose to dig deeper into the programming and hardware optimization, support will still be available for existing heterogeneous programming languages.

Each hardware vendor will be responsible for supplying the runtime libraries to support the tools for HSAIL. Although the HSA specification can be implemented without operating system (OS) support, all OS vendors have been invited to participate in the HSA Foundation. Thus far, three OS vendors have announced support for HSA, Canonical with Ubuntu, Linaro with Linux, and Oracle with Solaris. Other OS vendors can be expected to follow suit as HSA platforms become pervasive.

HSA in Action

With a slew of hardware and software specifications in development to support HSA, implementing the hardware modifications and developing the software tools will be done incrementally. The goals and requirements of HSA will go to market as an evolution of the hardware and software models to enable more efficient and programmable platforms. (As of this writing, AMD has announced that their recently released "Kaveri" APU has HSA features that enable application developers to access the key features of HSA.)

Power-Efficient Performance

All the proposed hardware and software changes result in power and performance advantages. The most significant advantage results from the use of shared and coherent memory. Memory transfers are some of the most power intensive actions in a computing solution. The HSA memory architecture reduces power consumption and latency associated with memory transfers between memory structures.

The UMS reduces latency associated with allocating tasks to accelerators and between compute units by eliminating several software layers in the system, particularly the OS kernel scheduler. By bypassing the OS kernel scheduler, application execution time may also be reduced if threads are not intermixed with threads from other applications, which is critical for some high-performance applications. Preemption can also be used to ensure that the highest priority tasks, particularly in the case of data dependency, are executed first and on the compute units that will provide the highest efficiency. The context switching ensures that those compute units can switch back and forth between applications and tasks accordingly. All these scheduling

techniques can be used to improve the overall performance of the system while reducing power consumption through better resource allocation and usage.

Despite the many advantages of the HSA requirements, there are a few considerations that will need to be managed. The first is the impact of HSAIL and the Finalizer on application start-up latency and overall execution performance. While adding translation layers in the software stack may seem to add additional execution cycles, which must be factored into the decision to allocate tasks to the accelerators, most devices already have a hardware abstraction layer to allow compatibility with newer versions of the HW (e.g. PTX). As a result, HSAIL is really replacing that abstraction layer with a standard translation layer.

In addition, cache coherency adds complexity to the hardware, which must be managed well to prevent excessive probing of caches. However, this can be mitigated with software tools and innovative hardware designs. As a result, the performance boost from HSA should more than offset any additional latency and execution time.

Context switching at the fine granularity that HSA requires can also add time to executing tasks, but the benefit of being able to prioritize the tasks and allocate them to specific resources should outweigh any additional latency in a high-performance or multi-tasking situation.

Programmability & Portability

Creating a single intermediate language that supports all HSA-compliant platforms and common programming languages will enable programmers at all skill levels to benefit from heterogeneous computing while creating a standard for future hardware and software solutions. It will also allow the more skilled “ninja” programmers to continue optimizing applications as necessary.

In addition, sharing a common view to virtual memory between tasks running on multiple heterogeneous processors eliminates the requirement for the programmer to provide mappings between different address spaces. This eliminates a major challenge to programming heterogeneous SoCs.

Pervasiveness

The current state of complex and fragmented heterogeneous programming models has forced many to avoid or oppose heterogeneous programming models. Creating a collaborative environment with shared IP, open source software, and standards will foster additional industry participation and innovation. As with any ecosystem movement, support will take time, but creating hardware and software standards paves the way for better programming models and a larger ecosystem. Once the momentum is created, the solutions will become entrenched.

The goal of HSA is to support other programmable accelerators, such as DSPs and FPGA, in the future. Any programmable logic block could potentially be supported in the HSA model, as long as the SoC vendor designs the SoC to meet the HSA specifications and provides support for the tools and runtime libraries.

Applications

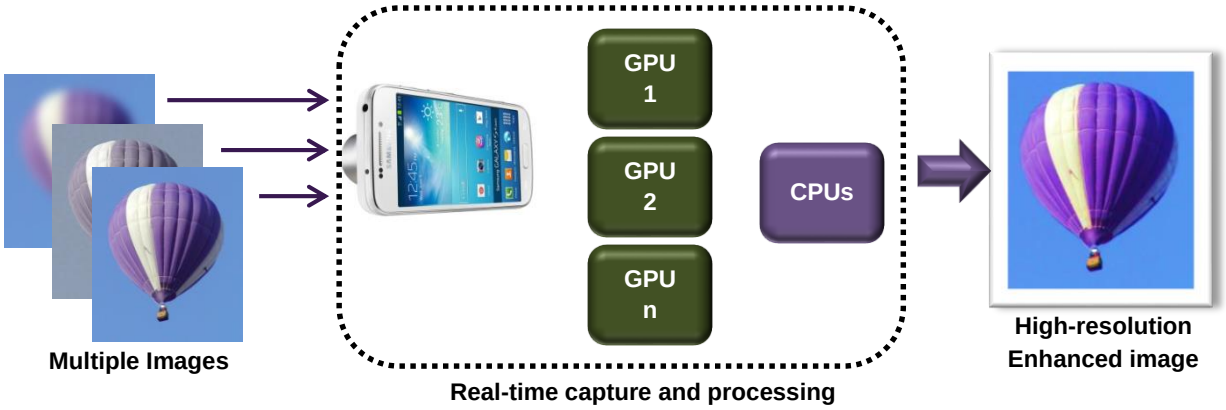
In the age of multitasking, most applications can and will benefit from various aspects of the HSA specifications. The following are descriptions of three common applications that are illustrative of the benefits of heterogeneous computing.

Image Processing

One of the first applications utilizing heterogeneous computing techniques in mobile devices is image processing. OEMs like Samsung are leveraging the ability of the processor to overcome limitations in the camera function. Even though image sensors with resolutions of 20MP and higher are available for mobile devices, the performance of the image sensor is often limited by the lens, which is kept small to reduce the weight, thickness, and the cost of the device. But, by using techniques of computational photography, such as capturing multiple successive images and processing them as one, mobile devices can produce much higher quality images while allowing for additional manipulation of the image through software.

Accomplishing this requires capturing and analyzing multiple images in succession, just like capturing multiple video frames, and processing them in real-time for brightness, contrast, color, facial recognition, and a number of other characteristics. Not only are GPU cores designed for pixel manipulation, but the higher number of cores allows the images, pixels, and characteristics to be processed in parallel. Today, select OEMs are working with OpenCL to hand-code memory and compute optimizations to accomplish this task. As a result of the complexity of the programming, these applications are only available from those OEMs on the targeted platforms. With standard heterogeneous programming solutions that leverage existing programming languages, any programmer or application would be able to leverage similar capabilities in a variety of applications targeting a variety of platforms.

Figure 4. Computational Photography



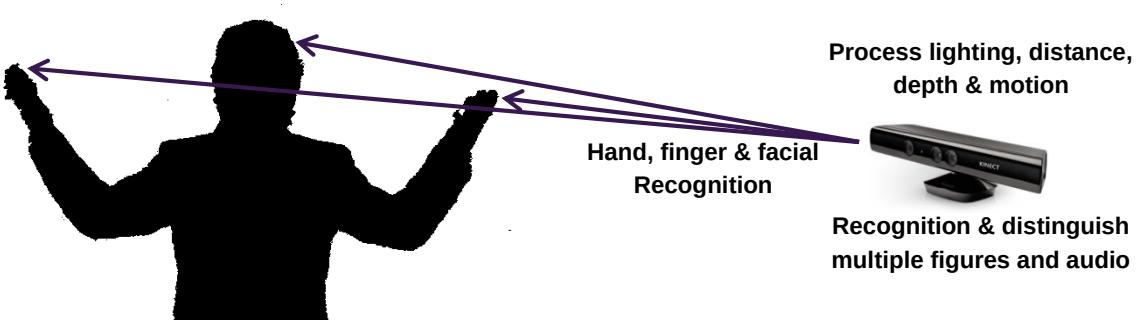
Source: TIRIAS Research, 12/13

In addition to the enhanced image quality and manipulation capabilities, using heterogeneous processing increases the performance over using CPUs while reducing the power required to perform compute intensive tasks.

Gesture Recognition

Gesture recognition is a complex form of video processing that involves a complex series of steps to analyze the characteristics of each frame, the objects within the frame, distance to each object, and changes from previous frames in real time. This includes light normalization, edge detection, depth analysis, and feature recognition to determine hands or fingers, and movement of hands and fingers. While a CPU or multiple CPUs would only be able to break this process into several threads, most current GPUs can break the process into hundreds of threads executing simultaneously. The parallel processing capability of a GPU makes it ideally suited for such as task. The ability to prioritize and allocate different steps of the algorithm to specific compute units (CPU or GPU cores) within HSA can further increase the performance while reducing the execution time and power consumed to perform the task.

Figure 5. Gesture Recognition



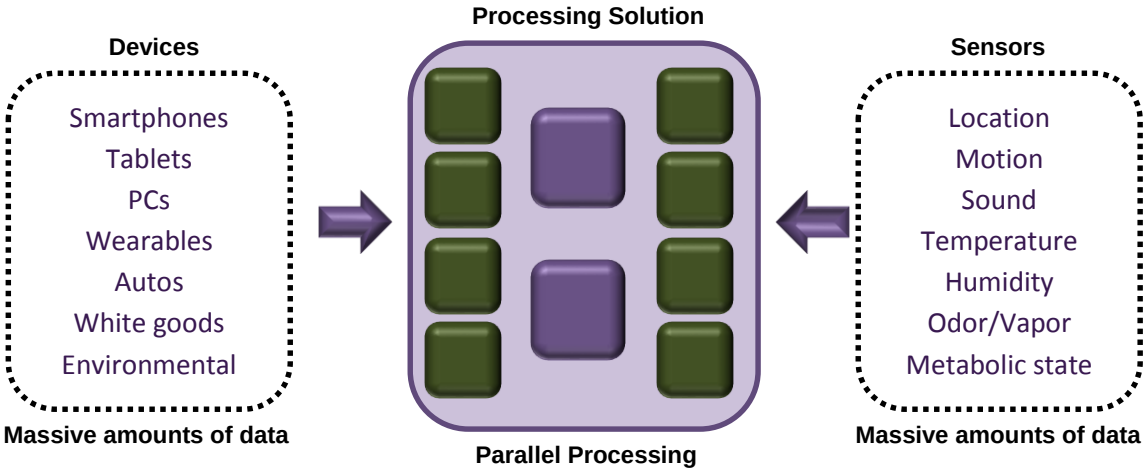
Source: TIRIAS Research, 12/13

Big Data Analytics

Big data analytics is the ability to process and analyze large amounts of data from multiple sources, essentially database mining in real time. The goal is to use algorithms to identify useful information, such as correlations and trends that can be used for other purposes. With the increasing use of information from the internet combined with more information that can be garnered from a world filled with sensors, there is considerable interest in big data analytics, but the challenge can be monumental.

As with all data mining tasks, the effort can best be accomplished by parallel processing, and the higher the number of parallel compute units the higher the performance and the less time and power required to execute tasks. The savings in die area, execution time, and power consumed increase with the complexity of the tasks and amount of data being analyzed using accelerators rather than traditional CPUs.

Figure 6. Big Data Analytics



Source: TIRIAS Research, 12/13

The Future of Heterogeneous Computing

Heterogeneous computing is already a reality. Every processor for computing, consumer, and embedded solutions is essentially an SoC with multiple programmable compute units. However, improving the efficiency and making the solutions easier to program is the next major challenge to be overcome to fully realize the end-user benefits. The formation of the HSA Foundation creates standards through an industry-wide consortium to address these issues and develop a path for innovation going forward.

Keys to the success of HSA and heterogeneous computing are common tools to support an increasing array of processing solutions and a common intermediate language to allow programming for and portability to other platforms. These are also key features not only for the computing solutions in the market today, but future solutions like wearable computing devices that will require even greater efficiency.

It is the belief of TIRIAS Research that the solutions proposed by the HSA Foundation are critical to enabling heterogeneous computing on a broader level and creating more efficient solution for future computing platforms. In addition, the broad support for the HSA Foundation by multiple IP, semiconductor, device OEMs, tools vendors, and academia creates an innovative environment for the entire industry.

Methodology

This paper was developed through interviews with HSA Foundation board members and partners, as well as a review of HSA and other heterogeneous computing presentations and documentation. Viewpoints and information from companies supporting other complementary and competing heterogeneous/parallel programming models were also evaluated and considered, as well as information from these programming models, such as OpenCL and CUDA. Links to all the publically available material are provided in the following reference section. Note that some of the presentation and technical material used in the development of the paper are not publicly available.

References

AMD, *AMD and HSA: A New Era of Vivid Digital experiences* website, 2013, http://www.amd.com/us/products/technologies/hsa/Pages/hsa.aspx#1

Benedict R. Gaster, HSA Memory Model presentation, August 2013, http://www.slideshare.net/hsafoundation/hsa-memory-model-hotchips-2013

Ben Sander, *HSAIL: Portable Compiler IR for HSA presentation*, August 2013, http://www.slideshare.net/hsafoundation/hsa-hsail-hot-chips2013-final2

HSA Foundation, *Heterogeneous System Architecture: A Technical Review* presentation, October 2012, http://www.slideshare.net/hsafoundation/hsa10-whitepaper

HSA Foundation, *HSA Programmer's Reference Manual: HSAIL Virtual ISA and Programming Model, Compiler Writer's Guide, and Object Format (BRIG)*, May 2013, https://hsafoundation.app.box.com/s/m6mrsjv8b7r50kqeyyal

Ian Bratt, *HSA Queuing* presentation, August 2013, http://www.slideshare.net/hsafoundation/hsa-queuing-hot-chips-2013

Khronos Group, OpenCL 2.0 Specifications, July 2013, http://www.khronos.org/registry/cl/

NVIDIA, *CUDA* website, 2013, http://www.nvidia.com/object/cuda_home_new.html

Phil Rogers, *Heterogeneous System Architecture Overview* presentation, August 2013, http://www.slideshare.net/hsafoundation/hsa-intro-hot-chips2013-final

Qualcomm, *Mobile Heterogeneous Computing in Action* webinar, September 2013, https://event.on24.com/eventRegistration/EventLobbyServlet?target=registration.jsp&eventid=688720&sessionid=1&key=F9F47B2FB3A1AB388B39D3AFE5D9AEDB&sourcepage=register