Software architecture for next generation hyperparallel cyber-physical hardware platforms: challenges and opportunities

Moris Behnam, Federico Ciccozzi, Mikael Sjödin
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
[name.surname]@mdh.se

Fredrik Bruhn
Bruhnspace AB
Uppsala, Sweden
f@bruhnspace.com

ABSTRACT

We present what is destined to become the de-facto standard for hardware platforms for next generation cyber-physical systems. Heterogeneous System Architecture (HSA) is an initiative to harmonize the industry around a common architecture which is easier to program and is an open standard defining the key interfaces for parallel computation. Since HSA is supported by virtually all major players in the silicon market we can conjecture that HSA, with its capabilities and quirks, will highly influence both the hardware and software for next generation cyber-physical systems.

In this paper we describe HSA and discuss how its nature will influence architectures of system software and application software. Specifically, we believe that the system software needs to both leverage the hyperparallel nature of HSA while providing predictable and efficient resource allocation to different parallel activities. The application software, on the other hand, should be isolated from the complexity of the hardware architecture but yet be able to efficiently use the full potential of the hyperparallel nature of HSA.

Categories and Subject Descriptors

C.1.3 [**Other Architecture Styles**]: Heterogeneous (hybrid) systems; D.2.11 [**Software Architectures**]: Languages

Keywords

HSA, software architecture, cyber-physical systems, model-driven engineering

1. INTRODUCTION

Currently the silicon market for embedded and Cyber-Physical Systems (CPS) is standing in front of a revolution with the newly launched standard for next generation hyper-parallel systems (HSA) being put into manufacturing. However, the software communities has, yet, paid remarkably little attention to the impact this new generation hardware will have on future software architecture and structure.

HSA stands for Heterogeneous System Architecture and the standard is managed by the HSA Foundation[1], which is supported by virtually all major players on the silicon market. HSA defines the key constituents and interfaces needed to harmonize use and programming of tightly integrated CPUs, GPUs, DSPs and other programmable accelerators (e.g., FPGAs). Thus programmers need only to learn one set of APIs for programming parallel applications and do not need to re-learn vendor-specific APIs for silicon from different manufacturers [1].

HSA is royalty free and hence attractive for many vendors of both hardware and software platforms. Some of the founding companies are AMD, ARM, Qualcomm and Texas Instruments. The HSA Foundation seeks to create applications that seamlessly blend scalar processing on the CPU, parallel processing on the GPU, and optimized processing on a DSP or FPGA via high bandwidth shared memory access enabling greater application performance at low power consumption. An important point for our research interest is that the HSA Foundation members are building a heterogeneous compute software ecosystem built on open, royalty-free industry standards and open-source software. For instance, the HSA runtimes and compilation tools are based on open-source technologies such as LLVM[2] and GCC[3].

Heterogeneous computing seems to be an emerging technology for power-efficient system design of modern platforms that no longer rely on a single general-purpose processor, but instead benefit from dedicated processors tailored for specific tasks. These specialized processors have traditionally been difficult to program due to separate memory spaces, kernel-driver-level interfaces, and specialized programming models. HSA aims to bridge this gap by providing a common system-architecture and a basis for designing higher-level programming models for all devices and hence is an interesting option for development of next generation CPS to be used in computationally demanding, safety critical, real-time applications for automation, aerospace, smart grid control etc.

In this paper we describe some of the unique features of HSA and outline some of the resulting research challenges and opportunities with respect to defining suitable archi-

[1]http://www.hsafoundation.com/
[2]http://llvm.org/
[3]https://gcc.gnu.org/

tectures for application- and system-software for using HSA in next generation CPS. In these challenges we also include design-time tools for platform-independent design and automated code generation from such tools.

The remainder of the paper is organized as follows. In Section 2 we provide an introduction on the exclusive characteristics of HSA, while in Section 3 and Section 4 we present challenges and opportunities related to the definition of suitable architectures for application- and system-software for HSA, respectively. The paper is concluded with a summary in Section 5.

2. HSA

The main aspects characterizing HSA are the following:

- Unified memory-space across all processing elements;
- Operation in pageable system memory;
- Full memory coherency;
- Hardware queuing model;
- Scheduling and context switching;
- HSA Intermediate language (HSAIL);
- High level language support for GPUs;

HSAIL is an explicitly parallel virtual instruction-set architecture for parallel programs with support for high level language features such as exceptions and virtual functions. It can support widely used high level languages and programming models like OpenMP, C++, and Python. The HSA memory model defines visibility ordering between all threads in the HSA enabled system and is also designed to be compatible with C++, OpenCL and .NET memory models with a relaxed consistency memory model for parallel computation performance. HSAIL and its place in the architecture of the compilation-chain is shown in Figure 1. The figure shows that the LLVM IR (Intermediate Representation) is used as target for the off-line part of the compilation-chain and that an on-line part converts to HSAIL which is then refined to target-specific code when a computational task is dispatched.

The HSA queuing model is an important feature that consists of user mode queuing for low latency hardware-dispatching where applications dispatch directly without OS or driver required in the dispatch path. With HSA enabled hardware, applications can create data structures in a single unified address space and can initiate work items on the most appropriate computational element for a given task. Sharing data between computational elements is as simple as sending a pointer. Multiple computational tasks can work on the same coherent memory, utilizing barriers and atomic memory operations as needed to maintain data synchronization.

As seen in the past, it is not sufficient to ask application vendors to change their software to fit a new kind of hardware. To reach the mainstream developers, it must be easy for everyone to participate. The HSA approach tries to bring such simplicity by bringing the hardware to the application programmer. HSA includes hardware, interfaces, common intermediate language, and standard runtime components to do all the necessary work. HSA maintains memory coherency and manages work queues under the hood, without exposing the underlying system complexity to the application developer and hence is attractive to the users. However, this also causes concerns for real-time applications due to, e.g., issues with on-line compilation, delays caused by cache-coherence protocols, hardware based dispatching and scheduling. Thus, to use HSA in safety critical and/or dependable applications, more research is needed on how to leverage these powerful hardware mechanisms, while maintaining control of execution, separation of different criticality levels, allocation and policing of resource usage in different co-existing applications, and so forth.

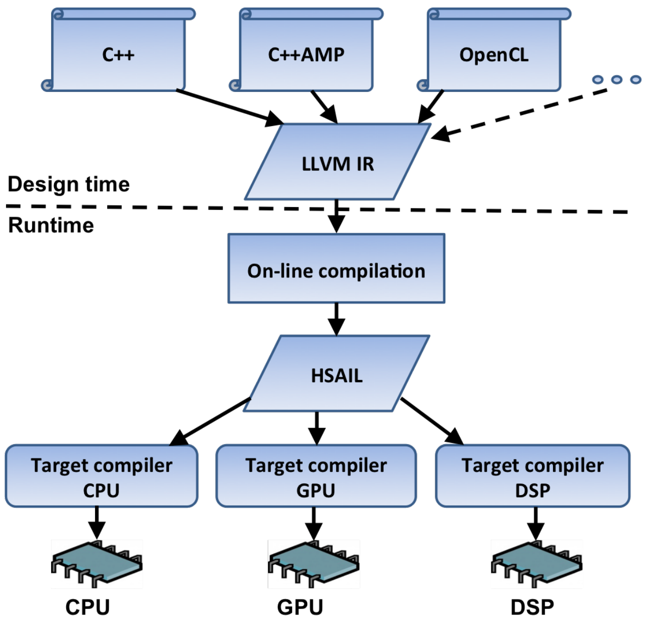

Figure 1: HSA compilation process

HSA example implementation.

Researchers at Mälardalen University (MDH) have, together with industry, developed a fault tolerant heterogeneous computing module for industrial applications [2]. The AMD “Steppe Eagle” G-series System-on-Chip was selected as the HSA reference architecture together with a Microsemi SmartFusion2 FPGA in an industrial standard Qseven form factor. This hardware allows MDH researchers to work on real hardware exposing the power of CPU, GPU, and FPGA/DSP. Ongoing hardware upgrades are being pursued using the new AMD Carrizo platform.

3. APPLICATION SOFTWARE

In several domains (e.g., medical, automotive, aerospace) CPS are expected to process massive amounts of data in real-time. Aiding in fulfilling these expectations, the development of hardware technologies towards heterogeneous configurations makes CPS able to handle, e.g., very high input data rates [3]. A typical scenario would be represented by video-input data coming into a CPU, which in turn may exploit one or more GPUs as coprocessors for parallel processing of large blocks of data. In this scenario other programmable accelerators (e.g., DPSs or FPGAs) can be exploited for switching among data transmission settings, performing non-vectorized signal processing, or other types of parallel processing. This ongoing technology shift from homogeneous (i.e., unicore or multicore platforms) to heterogeneous platforms raises a number of new research challenges and opportunities on how to design application-level software to be deployed on hyperparallel heterogeneous hardware.

As aforementioned, one of the main goals of HSA is to close the gap between hardware and software by making complex hardware-related mechanisms transparent to the software developer. In other words, it advocates abstraction from the intricacy of the underlying system-complexity and allows the developer to focus on the software-related issues. Advances and lessons learned in software development for homogeneous platforms should be built upon when designing the application-development support for this next generation software. More specifically, we advocate the exploitation of *Model-Driven Engineering (MDE)* [4], that has been attracting an increasing amount of industrial attention for the last 15 years due to its ability to shift the development focus from hand-written code (which is too close to the underlying computation technology), to models (which are closer to the problem domain and thereby more suitable for most application engineers) from which the implementation can be automatically generated through model manipulations. MDE aims at easing development by promoting models as primary artifacts. By dealing directly with domain-specific abstractions, models can reduce complexity and allow the developer to focus on the concepts that matter in the design of the application [5].

Moreover, through separation of concerns and platform-independence, same application-level software models, or parts of them (e.g., components), might be deployed to different hardware configurations. By means of MDE it is possible to systematically focus on different levels of abstractions at which all involved developing teams can (i) be relieved from the idiosyncratic intricacy of CPS low-level software, (ii) focus on improving the quality of CPS in terms of, e.g., safety, timeliness, and reusability, and (iii) promote the reuse of software components across different hardware configurations as well as different CPS. However, a research challenge here is that contemporary modelling techniques and tools are very seldom designed to allow modelling in a way that is suitable for later implementation on hyper-parallel heterogeneous platforms. For instance modelling of parallelizable algorithms (e.g., [6]) are typically done in languages quite far from those suited for large scale system-development.

In this direction, the Unified Modeling Language (UML), which represents the de-jure standard for MDE in industry, and its profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [7] have been leveraged for development and early analysis of parallel systems [8], system-on-chip [9], and safety-critical systems [10]. There exists some initial work towards the exploitation of UML and MARTE for the development of heterogeneous platforms too [11]. However, it is a big challenge to define a suitable architecture for modelling languages that allow all important concerns for next generation CPS-applications to be catered for (including, but not restricted to, concerns like parallelism, timeliness, energy-consumption, safety and security).

In the context of HSA, UML could be used for modelling the application software architecture in terms of, e.g., software components, while the MARTE profile could be leveraged to model schedulable and executable resources as, e.g., tasks and the allocations of software components to them, as well as memory partitions when needed. Moreover, the employment of other formalisms of the UML family of languages, such as Foundational UML (fUML)[4] and the Action Language for Foundation UML (ALF)[5] would enable the definition of complex algorithmic specifications within the modelled application software. In particular, ALF provides annotation mechanisms to define portions of behaviors (e.g., parallelizable constructs such as the `for` loop) as possibly parallel. "Possibly", since it would not enforce any parallelism but rather leave that possibility open at implementation or code generation time where this information is taken into account and, depending on the actual deployment configuration, can drive the generation of either parallel or sequential code. Doing so, no hardware-specific decision needs to be taken when modelling the application software and this transparency embodies one of the idiosyncratic pillars of the HSA initiative. A research challenge in this context is how to leverage the defined compilation chain in HSA, where intermediate languages like LLWM and HSAIL are exploited. While these languages provide some degree of hardware independence, they are also likely to be sources of unpredictability (e.g., due to online-compilation) if not used in a wise manner. In this matter, we foresee the opportunity of leveraging ALF in combination with HSAIL, where synergies between the two languages possibility to express . Also the interaction with the system-software (described in Section 4) needs to be automatically generated from models in a way that the application engineer, e.g., do not need to know the details of how to allocate resources and monitor for resource violations.

4. SYSTEM SOFTWARE

Most CPS are real-time embedded systems that must satisfy certain timing requirements. Depending on the type and the design of the CPS, the timing requirements might be meeting deadlines, bounding end to end delays/response times, bounding jitters, bounding deadline misses, etc. To satisfy these requirements sufficient system resources should be provided to each software application[6]. Furthermore, to reduce system-complexity and -cost of CPSes, a current trend is to integrate more software applications into a lower number of computing nodes. This implies that multiple applications share the node-local resources and thus it makes the execution of the real-time applications unpredictable.

Virtualization has been proposed as an attractive technique to solve the problem of sharing the system resources by creating a set of Virtual Machines (VMs), each of which is dedicated to executing a single application (or a small subset of the applications) on the physical system. Each VM can allow its associated application to access the system resources within a predefined bandwidth in order to manage the sharing of system resources among different applications. During runtime a middleware is responsible to provide the required resources to the VMs according to the specifications of their associated applications. This middleware is called Virtual Machine Monitor (VMM) or hypervisor.

Many commercial and open source solutions have been proposed to support virtualization on complex embedded

[4]http://www.omg.org/spec/FUML/1.1/

[5]http://www.omg.org/spec/ALF/1.0.1/

[6]Here we use the term application to denote a course grained, semi-independent, potentially reusable, architectural element. It typically includes a set of tasks or processes. An application will often be a top-level component in a component assembly if a component-based development strategy is employed.

systems such as VxWorks 653[7], LynxOS-178[8], Integrity MultiVisor[9], Real-Time Hyper Hypervisor[10], Enea Hypervisor[11], Xen[12], KVM[13], OKL4[14], to mention a few. However, these target unicore and/or homogenous multicore processor architectures and they are not directly applicable for HSA.

In homogenous multicore processor architectures, other computation units such as GPUs do not support preemptive scheduling and are usually modelled as mutual exclusive shared resources since context switches in these units introduce huge runtime overhead. This can introduce long blocking-times for other real-time applications that need these resources and make their execution unpredictable. Due to the enhanced architecture of HSA, preemptive scheduling is enabled for all computation processor units with low runtime overhead and thus they can be managed in a way similar to the CPU cores. This implies that the hypervisor might control the scheduling in these units in addition to the CPU cores and VMs can also be executed in all processing units. Furthermore, HSA improves the scheduling of computation processor units by providing efficient hardware queue management. The hypervisor can use the hardware queue management to schedule requests to computation processor units in order to significantly decrease the runtime overhead for scheduling. In addition, by providing heterogeneous computation processors (CPU, GPU, DSP, etc.), different resource allocation strategies can be applied to optimize the resources usage and improve the timing behavior of the software applications. For instance, one VM can be assigned to each application in each of the computation processor units to execute tasks from that specific application. The main advantage of this solution is that the application can use any available computational unit among those that it is allowed to use. However, dependencies among tasks running in different units might introduce long delays. Another alternative might be allocating only one VM for each application that manages the execution of tasks in different processor units. However, the application tasks would not execute until all required processor units are available. A research challenge is to combine the above mentioned alternatives and to strike the right balance between them to get the most out of HSA platforms.

An overall challenge is represented by investigating the possibilities to define novel virtualization mechanisms specifically for heterogeneous configurations. But we foresee the need to have abstract virtual machines where the exact configuration of CPUs, GPUs, DPSs, etc. is not concretized. This would allow the VMM to insatiate a concrete allocation of hardware units at run-time. Today it is unknown what a suitable architecture and design for such a heterogeneous virtual machine might be.

Furthermore, developing new offline and/or online algorithms that allocate HSA resources efficiently will be a significant research direction. These algorithms will require proper mathematical models that can evaluate the required resources for each application and test the feasibility of the applications integrated in a HSA platform.

[7]http://www.windriver.com/
[8]http://www.lynuxworks.com/
[9]http://www.ghs.com/
[10]http://www.ni.com/
[11]http://www.enea.com/
[12]http://www.xen.org/
[13]http://www.linux-kvm.org/
[14]http://www.ok-labs.com/

5. SUMMARY

In this paper we have presented some of the opportunities provided by the novel standard HSA (Heterogeneous Systems Architecture), which is likely to be a dominating standard for next generation high-performance cyber-physical systems. These opportunities include massive parallel execution, energy efficient execution and simplified programming through unified interfaces and intermediate languages.

However, to leverage the computational power of HSA in safety critical and/or dependable real-time systems, significant research challenges need to be tackled. In this paper some of these challenges have been discussed, with focus on those related to future architectures for application development and system software for HSA. These challenges include architecture of modelling support and transformation as well as architecture for resource abstraction, allocation and policing by the system software.

6. REFERENCES

[1] Rogers, P. and Sander, B. and Chung, Y.-C. and Gaster, B.R. and Persson, H. and Hwu, W.-m. W. Heterogeneous System Architecture (HSA): Architecture and Algorithms Tutorial. http://www.hsafoundation.com/isca-2014-tutorial-2/.

[2] F. Bruhn, K. Brunberg, J. Hines, L. Asplund, and M. Norgren. Introducing Radiation Tolerant Heterogeneous Computers for Small Satellites. In *IEEE Aerospace Conference 2015*. IEEE, March 2015.

[3] D. Hallmans, M. Asberg, and T. Nolte. Towards using the Graphics Processing Unit (GPU) for embedded systems. In *Procs of ETFA*, pages 1–4, 2012.

[4] J. Bézivin. On the Unification Power of Models. *Software and System Modeling*, 4, 2005.

[5] D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, February 2006.

[6] E. Axelsson, K. Claessen, G. Devai, Z. Horvath, K. Keijzer, B. Lyckegård, A. Persson, M. Sheeran, J. Svenningsson, and A. Vajda. Feldspar: A domain specific language for digital signal processing algorithms. In *Procs of MEMOCODE*, pages 169–178, July 2010.

[7] S. Taha, A. Radermacher, S. Gérard, and J.-L. Dekeyser. MARTE: UML-based Hardware Design from Modelling to Simulation. In *Procs of FDL*, pages 274–279, 2007.

[8] A.W.O. Rodrigues, F. Guyomarc'h, and J.-L. Dekeyser. An MDE Approach for Automatic Code Generation from UML/MARTE to OpenCL. *Computing in Science Engineering*, 15:46–55, 2013.

[9] I.R. Quadri, S. Meftali, and J.-L. Dekeyser. Designing dynamically reconfigurable SoCs: From UML MARTE models to automatic code generation. In *Procs of DASIP*, pages 68–75. IEEE, 2010.

[10] S. Burmester, H. Giese, M. Hirsch, D. Schilling, and M. Tichy. The fujaba real-time tool suite: model-driven development of safety-critical, real-time systems. In *Procs of ICSE*, pages 670–671. ACM, 2005.

[11] F. Ciccozzi. Towards code generation from design models for embedded systems on heterogeneous CPU-GPU platforms. In *Procs of ETFA*, pages 1–4, 2013.