Project Beehive: A Hardware/Software Co-designed Stack for Runtime and Architectural Research

Christos Kotselidis, Andrey Rodchenko, Colin Barrett, Andy Nisbet, John Mawer, Will Toms, James Clarkson, Cosmin Gorgovan, Amanieu d'Antras, Yaman Cakmakci, Thanos Stratikopoulos, Sebastian Werner, Jim Garside, Javier Navaridas, Antoniu Pop, John Goodacre, and Mikel Luján

Advanced Processor Technologies Group
The University of Manchester,
`first.last@manchester.ac.uk`

Abstract. The end of Dennard scaling combined with stagnation in architectural and compiler optimizations makes it challenging to achieve significant performance deltas. Solutions based solely in hardware or software are no longer sufficient to maintain the pace of improvements seen during the past few decades. In hardware, the end of single-core scaling resulted in the proliferation of multi-core system architectures, however this has forced complex parallel programming techniques into the mainstream. To further exploit physical resources, systems are becoming increasingly heterogeneous with specialized computing elements and accelerators. Programming across a range of disparate architectures requires a new level of abstraction that programming languages will have to adapt to. In software, emerging complex applications, from domains such as Big Data and computer vision, run on multi-layered software stacks targeting hardware with a variety of constraints and resources. Hence, optimizing for the power-performance (and resiliency) space requires experimentation platforms that offer quick and easy prototyping of hardware/software co-designed techniques. To that end, we present Project Beehive: A Hardware/Software co-designed stack for runtime and architectural research. Project Beehive utilizes various state-of-the-art software and hardware components along with novel and extensible co-design techniques. The objective of Project Beehive is to provide a modern platform for experimentation on emerging applications, programming languages, compilers, runtimes, and low-power heterogeneous many-core architectures in a full-system co-designed manner.

1 Introduction

Traditionally, software and hardware providers have been delivering significant performance improvements on a yearly basis. Unfortunately, this is beginning to change. Predictions about "dark silicon" [2] and resiliency, especially in the forthcoming exascale era [1], suggest the traditional approaches to computing

problems are impeded by power constraints; saturation on architectural and compiler research; and process manufacturing. Mitigation of these problems is likely to come through vertical integration and optimization techniques; or bespoke solutions for each or a cluster of problems. However, whilst such an approach may yield the desired results it is both complex and expensive to implement. At the current time only a handful of vendors, such as Oracle, Google, Facebook, etc., have both the financial resources and engineering expertise required to deliver on this approach.

Co-designing an architectural solution at the system-level[1] requires significant resources and expertise. The design-space to be explored is vast, and there is the potential that a poor, even if well intentioned, decision will propagate through the entire co-designed stack; amending the consequences at a later date may prove extremely complex and expensive if not impossible.

Project Beehive aims to provide a platform for rapid experimentation and prototyping, at the system-level, enabling accurate decision making for architectural and runtime optimizations. The project is intended to facilitate:

– Co-designed research and development for traditional and emerging workloads such as Big Data and computer vision applications.
– Co-designed compiler and runtime research of multiple languages building on top of Truffle [5], Graal, and Maxine VM [4].
– Heterogeneous processing on a variety of platforms focusing mainly on ARMv7, Aarch64, and x86.
– Fast prototyping and experimentation on heterogeneous programming on GPGPUs and FPGAs.
– Co-designed architectural research on power, performance, and reliability techniques.
– Dynamic binary optimization techniques via binary instrumentation and optimization on both at the system and chip level.

The following subsections describe the general architecture of Project Beehive and its various components. Finally, some preliminary performance numbers along with the short-term and long-term plans are also presented.

2 Beehive Architecture

2.1 Overview

Beehive, as depicted in Figure 1, targets a variety of workloads spanning from traditional benchmarks to emerging applications from a variety of domains such as computer vision and Big Data. Applications can execute either: directly on hardware, in-directly on hardware using our dynamic binary optimization layer (MAMBO and MAMBO64) or inside our simulator.

---

[1] In this context we refer to architectural solution as a co-designed solution that spans from a running application to the underlying hardware architecture.
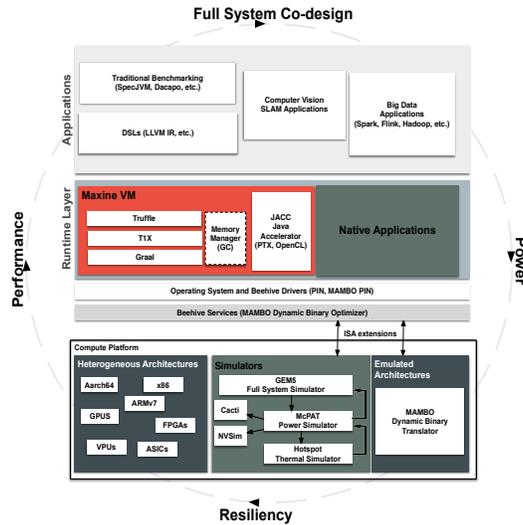
Fig. 1. Project Beehive architecture overview.

The runtime layer centers around an augmented Maxine Research VM and MAMBO components. The VM provides the capability to target both high-performance x86 and low-power ARM systems, in addition to heterogeneous architectures. Our enhanced capability is made possible via a range of compilers: the T1X and Graal compilers support ARMv7, Aarch64, and x86 architectures, while our Jacc compiler can target GPGPUs, FPGAs, and SIMD units. Moreover, by replacing the C1X compiler with Graal it is also possible to fully benefit from Truffle AST interpreter on our VM.

Beehive offers the ability to perform architectural research via an integrated simulation environment. Our environment is built around the gem5, Cacti, NvSim, and McPat tools. By using this environment, novel micro-architectures can be simulated and performance, power, temperature, and reliability metrics to be gathered.

2.2 Applications

Beehive targets a variety of applications in order to enable co-designed optimizations in numerous domains. Whilst compiler and micro-architectural research traditionally uses benchmarks such as SPEC and PARSEC, Beehive also considers complex emerging application areas. The two primary domains considered are Big Data software stacks such as Spark, Flink, and Hadoop along with computer vision SLAM (Simultaneous Localization and Mapping). In the vision arena SLAMBench [3] will be the main vehicle of experimentation. SLAMBench currently includes implementations in C++, CUDA, OpenCL, OpenMP and Java allowing a broad range of languages, platforms and techniques to be investigated.

2.3 Runtime Layer

Some of the key features of Beehive are found in its runtime layer, which provides capability beyond simply running native applications. For instance, our MAMBO64 component is able to translate ARMv7 binaries into Aarch64 instructions at runtime, whilst MAMBO enables binary translation/optimization in a manner similar to PIN [2].

Despite being able to execute native C/C++ applications, Beehive has been designed to target languages that utilize a managed runtime system. Our managed runtime system is based on the Maxine Research VM which has been augmented with a selection of state-of-the-art components. For example, we have properly integrated and increased the stability of both the template, T1X, compiler and the Graal compiler which also allows Project Beehive to utilize the Truffle AST interpreter. Moreover, this work has required us to undertake extensive infrastructure work to allow us to easily downstream the Graal and Truffle code bases in order to provide Beehive with the state-of-the-art components on a regular basis.

The VM is designed to enable execution across a range of hardware configurations. To that end, we introduce support for low-power ARM systems, by extending the T1X and Graal compilers to support both the ARMv7 and Aarch64 architectures, along with continuing the existing x86 support. Additionally, the VM supports heterogeneous execution via the Jacc (Java accelerator) framework. By annotating source code using Jacc's API, which is similar to OpenMP/OpenAcc, it is possible to execute performance critical code on specialized hardware such as GPGPUs and FPGAs.

Regarding the memory manager (GC), various options are being explored ranging from enhancing Maxine VM's current GC algorithms to porting existing state-of-the-art memory management components.

2.4 Hardware Layer

As depicted in Figure 1, Project Beehive targets a variety of hardware platforms and therefore significant effort is being placed in providing the appropriate support for the compilers and runtime of choice.

In addition targeting conventional CPU/GPU systems, it is also possible to target FPGA systems, the primary target being Xilinx Zynq, ARM/FPGA. In-house tools and IP (Intellectual Property) can be used to rapidly assemble hardware systems targeted at specific applications, for example accelerators for computer vision, hardware models appropriate to system level simulation or database accelerators. The hardware accelerators have access to the processor's main memory at 10Gb/s through the processor's cache system allowing high speed transfer of data between generic and custom processing resources. The system uses an exclusively user space driver allowing new hardware to be added

---

[2] https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

and easily linked to runtimes or binary translators. Using the Zynq's ARM processors it is possible to identify IP blocks currently configured on the FPGA and if necessary reconfigure it, whilst applications continue running on the host ARM device. This allows a runtime to dynamically tune its hardware resources to match its power/performance requirements.

Typical examples of the hardware layer in use might include preprocessing image data in SLAMBench; integrating with MAMBO's dynamic binary instrumentation to provide high performance memory system simulation, using our memory system IP; or providing a small low power micro-controller which might be used for some runtime housekeeping task.
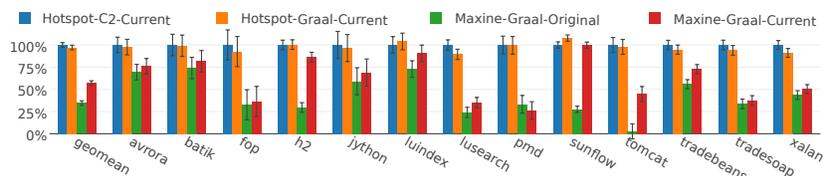
Fig. 2. DaCapo-9.12-bach benchmarks (higher is better) normalized to Hotspot-C2-Current.

2.5 Simulation Layer

Despite running directly on real hardware, Beehive offers the opportunity to conduct micro-architectural research via its simulation infrastructure. The gem5 full-system simulator has been augmented to include accurate power and temperature models using the McPat and Hotspot simulators. Both simulators are invoked within the simulator allowing power and temperature readings to be triggered either from the simulator (allowing for transient power and temperature traces to be recorded) or from within the simulated OS (allowing accurate power and temperature figures to be used within user space programs) with minimal performance overhead. Furthermore, the non-volatile memory simulator NVSim has been incorporated into the simulation infrastructure. This can be invoked by McPat (along side the conventional SRAM modeling tool Cacti) and allows accurate delay, power and temperature modeling of non-volatile memory anywhere in the memory hierarchy.

3 Initial Evaluation

Project Beehive combines work conducted on various parts of the co-designed stack. Although, presently, it can not be evaluated holistically, individual components are very mature and can be independently evaluated. Due to space limitation, we present preliminary developments in two areas of interest.

3.1 Maxine VM Development

The following major changes to Maxine VM were done since Oracle Labs has stopped its active development: 1) profiling instrumentation in T1X, 2) more optimistic optimizations were enabled (including optimistic elimination of zero count exception handlers), and 3) critical math substitutions were enabled. The following configurations were evaluated on DaCapo-9.12-bach benchmarks (with the exception of `eclipse`) as depicted in Figure 2: 1) Hotspot-C2-Current (ver.1.8.0.25), 2) Hotspot-Graal-Current[3], 3) Maxine-Graal-Original[4], 4) Maxine-Graal-Current[5]. Our work on improving performance and stability of Maxine-Graal resulted in 1.64x speedup over the initially committed version. The plan is to keep working towards increasing performance and stability of all versions of Maxine-Graal; ARMv7, Aarch64, and x86.

3.2 MapReduce Use Case

Parallel frameworks, such as Flink, Spark and Hadoop, abstract functionality from the underlying parallelism. Performance tuning is therefore reliant on the capabilities provided through specializations in the API. These attempts to reduce the semantic distance between applications elements require additional experience and expertise. Furthermore, every layer in the software stack abstracts the functionality and hardware even further. Co-designing the layers in a complete application is an alternative approach that aims to maintain productivity for all.

MapReduce is a very simple framework, yet popular and a powerful tool in the Big Data arena. In multicore implementations there exists a semantic distance between the Map and Reduce methods. The method level abstraction for compilation in Java cannot span the distance and so compiles each method independently. Existing MapReduce frameworks offer the Combine method explicitly in order to compensate for this inconvenience.

By designing a new MapReduce framework, with a co-designed optimizer, it is possible to inline the Reduce method within the Map method. This allows the optimizing compiler of Java to virtualize or eliminate many objects that would otherwise be required as intermediate data. It is possible to reduce execution times up to 2.0x for naive, yet efficient, benchmarks at the same time as reducing the strain on the GC. Importantly this is possible without altering or extending the API presented to the user.

4 Conclusions

In this paper, we introduced Project Beehive: a hardware/software co-designed stack for full-system runtime and architectural research. Project Beehive builds

---

[3] http://hg.openjdk.java.net/graal/graal rev.21075

[4] http://kenai.com/hg/maxine~maxine rev.8749, ~graal rev.11538

[5] http://kenai.com/hg/maxine~maxine rev.8809, ~graal rev.11557

on top of existing state-of-the-art as well as novel components at all layers of the stack. The short-term plans are to complete the ARMv7 and Aarch64 ports of T1X and Graal compilers, while increasing confidence by achieving a high application coverage, along with establishing a high-performing GC framework.

Our vision regarding Project Beehive is to unify the platform capabilities under a semantically aware runtime increasing developer productivity. Furthermore, we plan on defining a hybrid ISA between emulated and hardware capabilities in order to provide a roadmap of movement of capabilities between abstractions offered in software that later are offered in hardware. Finally, we plan to work on new hardware services for scale out and representation of volatile and non-volatile communication services in order to provide a consistent view of platform capabilities across heterogeneous processors.

References

1. Cappello, F., Geist, A., Gropp, B., Kale, L., Kramer, B., Snir, M.: Toward exascale resilience (November 2009)
2. Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. In: ISCA '11 (2011)
3. Nardi, L., Bodin, B., Zia, M.Z., Mawer, J., Nisbet, A., Kelly, P.H.J., Davison, A.J., Luján, M., O'Boyle, M.F.P., Riley, G., Topham, N., Furber, S.: Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In: ICRA (2015)
4. Wimmer, C., Haupt, M., Van De Vanter, M.L., Jordan, M., Daynès, L., Simon, D.: Maxine: An approachable virtual machine for, and in, java (January 2013)
5. Würthinger, T., Wimmer, C., Wöß, A., Stadler, L., Duboscq, G., Humer, C., Richards, G., Simon, D., Wolczko, M.: One vm to rule them all. In: Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software. Onward! '13 (2013)