Towards Practical Heterogeneous Virtual Machines

DOI:
10.1145/3191697.3191730

Document Version
Accepted author manuscript

Towards Practical Heterogeneous Virtual Machines

James Clarkson, Juan Fumero, Michail Papadimitriou, Maria Xekalaki, Christos Kotselidis
The University of Manchester
United Kingdom, M13 9PL
first.last@manchester.ac.uk

ABSTRACT

Heterogeneous computing has emerged as a means to achieve high performance and energy efficiency. Naturally, this trend has been accompanied by changes in software development norms that do not necessarily favor programmers. A prime example is the two most popular heterogeneous programming languages, CUDA and OpenCL, which expose several low-level features to the API making them difficult to use by non-expert users.

Instead of using low-level programming languages, developers tend to prefer more high-level, object-oriented languages typically executed on managed runtime environments. Although many programmers might expect that such languages would have already been adapted for execution on heterogeneous hardware, the reality is that their support is either very limited or totally absent. This paper highlights the main reasons and complexities of enabling heterogeneous managed runtime systems and proposes a number of directions to address those challenges.

CCS CONCEPTS

• **Software and its engineering → Virtual machines**;

KEYWORDS

Virtual Machines, Java, OpenCL, GPU, FPGA

ACM Reference Format:
James Clarkson, Juan Fumero, Michail Papadimitriou, Maria Xekalaki, Christos Kotselidis. 2018. Towards Practical Heterogeneous Virtual Machines. In *Proceedings of 2nd International Conference on the Art, Science, and Engineering of Programming (<Programming'18> Companion).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3191697.3191730

1 INTRODUCTION

Heterogeneous computing, initially introduced in the form of GPGPUs, has recently become very popular as a means to accelerate various applications from different domains including HPC and Big Data. Towards the transition to heterogeneous computing a number of new programming languages has emerged with CUDA and OpenCL being the most prevalent. Both languages enable the execution of high performance code on a variety of devices including GPUs and FPGAs, and have been designed at a lower level than non-expert programmers would expect. Both the prerequisite in-depth understanding of the architectural characteristics of the underlying heterogeneous hardware, and the exposure of low-level primitives to the API, make their use more challenging by developers, who typically write code in more widely used high-level programming languages such as Java, C#, Python, R, and JavaScript [14]. These languages typically run on top of a Managed Runtime Environment (MRE) and, in order to exploit a heterogeneous hardware resource (e.g. GPU), wrapper libraries are required to direct execution to pre-compiled OpenCL/CUDA kernels. Such an approach not only "violates" the semantics of the high-level programming language, but also requires from developers to gain expertise on new low-level programming environments.

Recently, a significant amount of work [1–9, 12, 13, 15] has been dedicated to enable dynamic JIT compilation of high-level code (e.g. Java, JavaScript, R) to heterogeneous hardware via OpenCL or PTX/CUDA. These approaches partially solve the challenge of heterogeneous execution since they mainly focus only on the aspect of compilation. MREs are complex systems encompassing a number of interconnected software components such as compilers, runtimes, memory managers, and garbage collectors. Consequently, all of these subsystems have to be re-engineered to enable performance and usability from interpreted programming languages. The remainder of this paper discusses the challenges of enabling heterogeneous execution of MREs and presents our proposal and work in progress towards addressing those challenges.

2 CHALLENGES

In this section we highlight the most significant challenges in designing heterogeneous MREs.

Programmability: High-level features of managed languages such as dynamic memory allocation and management, virtual calls and exception handling, although supported by Java and other languages, they are not present in CUDA or OpenCL. Therefore, they are not naturally supported on heterogeneous devices.

Transparency: Platform portability is a key design principle of Java that has been achieved through virtualization and abstraction of the underlying hardware architecture by the JVM. Similarly, the integration of hardware accelerators in the JVM should also follow the same principle. Developers should be able to run their code on the underlying device without having to explicitly manage memory, parallelism, code placement and other added complexities.

Adaptability: The existing compilers of JVMs have been tuned throughout the years for CPU execution exploiting Instruction Level Parallelism (ILP). Heterogeneous hardware accelerators, however, are built for different execution scenarios such as data parallelism (GPUs) or task pipelining (FPGAs). In addition, different

<Programming'18> Companion, April 9–12, 2018, Nice, France

ACM ISBN 978-1-4503-5513-1/18/04.
https://doi.org/10.1145/3191697.3191730

GPUs have different characteristics and capabilities that compilers must accommodate. Therefore, JVMs must be able to dynamically adapt the generated code based on the particular hardware device transparently to the user.

Device Portability: Breaking away from the norm that applications always execute on CPUs, heterogeneous MREs must accommodate device portability. For example, a vanilla Java application should be able to exploit any hardware device, even if unknown during development time. In addition, in cloud deployed Big Data applications where fault tolerance is important, MREs should be able to adapt the code (through de-optimization and re-compilation) if a node with a specific device fails.

Performance Portability: The main motivation behind hardware accelerators is increased performance. A heterogeneous MRE should aim in delivering the same performance when executing the code on different accelerators. The trade-offs between peak performance, compilation time, and heterogeneous execution should always yield the best possible performance regardless of the type of accelerator that exists on our system. Therefore, a more sophisticated decision-making model is required compared to the existing counter-based compilation of "hot" methods.

3 PROPOSAL

This section outlines our proposal for heterogeneous MREs with some initial results described in [10, 11].

Task-Based API. To address **programmability**, we propose a task-based API for heterogeneous programming that augments existing Java APIs in a seamless manner. Developers create tasks by invoking existing Java methods that will be executed on a device with minimal changes in the source code. Listing 1 shows a pseudocode of the task API. As shown, a group of tasks (t1 and t2) is created in the form of a task group (lines 6-8). Either the whole task group or individual tasks can be scheduled on the same, or different, heterogeneous device(s). Also, they can execute in parallel and all data dependencies (and data copying) is handled transparently by the VM with the help of the task scheduler and the runtime.

Tasks in our system encapsulate existing Java methods. In the example shown in Listing 1, the task `t1` refers to the `Compute.bfs` Java method while the task `t2` refers to the `Compute.mapReduce` Java method. Those methods contain legal Java code that can include Java objects, exceptions, etc. The runtime system, in combination with the JIT compiler and the heterogeneous VM, compile and execute the existing Java code to the target accelerator via OpenCL.

```
1  public class Compute {
2      public static void bfs(in, out) { ... }
3      public static void mapReduce(in, out) { ... }
4  }
5  public static void main(String[] args) {
6      Schedule TaskGroup "g1" {
7         task "t1" :: Compute::bfs, inA, outA
8         task "t2" :: Compute::mapReduce, inB, outB }
9  }
```

Listing 1: Example of the Task Parallel API.

Runtime System. The runtime system performs type inference and obtains sizes and meta-data needed for compiling and optimizing tasks. It will also build a Data Flow Graph (DFG) to optimize data dependencies between tasks and generate new bytecodes for orchestrating their execution on the heterogeneous device.

Heterogeneous VM. This component will execute the new bytecode generated by the runtime system in a bytecode interpreter, resulting essentially in a "VM-in-a-VM" aiming at heterogeneous hardware virtualization. The main VM is the Java Virtual Machine (JVM) while the secondary is the VM that virtualizes the heterogeneous devices. Our runtime system generates, at runtime, new bytecodes to execute tasks on heterogeneous devices. These new bytecodes are interpreted in the heterogeneous VM that optimizes and orchestrates the execution across the accelerators. The heterogeneous VM will also manage memory between Java and the device while inspecting hardware features of the underlying device. This allows **adaptability and transparency** of applications when running on heterogeneous hardware.

Listing 2 shows an example of this new bytecode where two kernels (k1 and k2) are launched for execution on two devices (*DEV*0 and *DEV*1) in lines 5 and 9. These two tasks correspond to the generated bytecode of Listing 1. Before running the kernel, on-device memory is allocated and data is copied from the VM (output data are also copied back to the host's memory). Allocating, copying memory, and running kernels can be synchronous or asynchronous operations, allowing more task-level parallelism. Just before running a kernel, the heterogeneous VM compiles the input Java methods (tasks) for a specific device and executes the resulting code.

Heterogeneous JIT Compilation. The JIT compiler and the VM work together for bringing **performance** and **portability** using iterative compilation. The compiler will optimize the input tasks for the selected device using its hardware information during runtime. It will also report to the VM profiling information to improve thread scheduling, bringing **adaptability** of high-level code to the target hardware.

```
1  start           // start the Heterogeneous VM
2  mem_alloc inA   // memory alloc on the device
3  mem_alloc outA  // memory alloc on the device
4  copy_in inA     // data transfer Host->Device
5  runParallel t1 DEV0 // run task t1 on device 0
6  mem_alloc inB   // memory alloc on the device
7  mem_alloc outB  // memory alloc on the device
8  copy_in inB     // data transfer Host->Device
9  runParallel t2 DEV1 // run task t2 on device 1
10 copy_out outA   // data transfer Device->Host
11 copy_out outB   // data transfer Device->Host
12 finish          // finish the VM
```

Listing 2: Bytecode example of heterogeneous MRE.

ACKNOWLEDGMENTS

This work is partially supported by the EPSRC grants PAMELA EP/K008730/1 and AnyScale Apps EP/L000725/1, and the EU Horizon 2020 E2Data 780245.

REFERENCES

[1] AMD. 2016. Aparapi. (2016). http://aparapi.github.io/.

[2] James Clarkson, Christos Kotselidis, Gavin Brown, and Mikel Luján. 2017. Boosting Java Performance using GPGPUs. In *Proceedings of the 30th International Conference on Architecture of Computing Systems (ARCS '17)*.

[3] Christophe Dubach, Perry Cheng, Rodric Rabbah, David F. Bacon, and Stephen J. Fink. 2012. Compiling a High-level Language for GPUs: (via Language Support for Architectures and Compilers). In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '12)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/2254064.2254066

[4] Juan Fumero, Michel Steuwer, Lukas Stadler, and Christophe Dubach. 2017. Just-In-Time GPU Compilation for Interpreted Languages with Partial Evaluation. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '17)*. ACM, New York, NY, USA, 60–73. https://doi.org/10.1145/3050748.3050761

[5] Juan Fumero, Michel Steuwer, Lukas Stadler, and Christophe Dubach. 2017. OpenCL JIT Compilation for Dynamic Programming Languages. In *MoreVMs Workshop. Collocated with Programing 2017 (MoreVMs '17)*.

[6] Juan José Fumero, Toomas Remmelg, Michel Steuwer, and Christophe Dubach. 2015. Runtime Code Generation and Data Management for Heterogeneous Computing in Java. In *Proceedings of the Principles and Practices of Programming on The Java Platform (PPPJ '15)*. ACM, New York, NY, USA, 16–26. https://doi.org/10.1145/2807426.2807428

[7] Juan José Fumero, Michel Steuwer, and Christophe Dubach. 2014. A Composable Array Function Interface for Heterogeneous Computing in Java. In *ARRAY'14: Proceedings of the 2014 ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*. 44.

[8] Akihiro Hayashi, Max Grossman, Jisheng Zhao, Jun Shirako, and Vivek Sarkar. 2013. Accelerating Habanero-Java Programs with OpenCL Generation. In *Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools (PPPJ '13)*. ACM, New York, NY, USA, 124–134. https://doi.org/10.1145/2500828.2500840

[9] K. Ishizaki, A. Hayashi, G. Koblents, and V. Sarkar. 2015. Compiling and Optimizing Java 8 Programs for GPU Execution. In *2015 International Conference on Parallel Architecture and Compilation (PACT)*. 419–431. https://doi.org/10.1109/PACT.2015.46

[10] Christos Kotselidis, James Clarkson, Andrey Rodchenko, Andy Nisbet, John Mawer, and Mikel Luján. 2017. Heterogeneous Managed Runtime Systems: A Computer Vision Case Study. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '17)*. ACM, New York, NY, USA, 74–82. https://doi.org/10.1145/3050748.3050764

[11] Christos Kotselidis, Andrey Rodchenko, Colin Barrett, Andy Nisbet, John Mawer, Will Toms, James Clarkson, et al. 2015. Project Beehive: A Hardware/Software Co-designed Stack for Runtime and Architectural Research. *In 9th International Workshop on Programmability and Architectures for Heterogeneous Multicores (MULTIPROG)* (2015). http://arxiv.org/abs/1509.04085

[12] Philip C. Pratt-Szeliga, James W. Fawcett, and Roy D. Welch. 2012. Rootbeer: Seamlessly Using GPUs from Java *(HPCC-ICESS)*, Geyong Min, Jia Hu, Lei (Chris) Liu, Laurence Tianruo Yang, Seetharami Seelam, and Laurent Lefevre (Eds.).

[13] Sumtra. 2015. Sumatra OpenJDK. (2015). http://openjdk.java.net/projects/sumatra/.

[14] December TIOBE List. 2017. TIOBE Programming Language Index. (2017). http://www.tiobe.com/tiobe-index/.

[15] Wojciech Zaremba, Yuan Lin, and Vinod Grover. 2012. JaBEE: Framework for Object-oriented Java Bytecode Compilation and Execution on Graphics Processor Units. In *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units (GPGPU-5)*. ACM, New York, NY, USA, 74–83. https://doi.org/10.1145/2159430.2159439