Many-Core Virtual Machines

Decoupling Abstract from Concrete Concurrency

Stefan Marr[1] Theo D'Hondt

Software Languages Lab
Vrije Universiteit Brussel, Belgium
stefan.marr@vub.ac.be

Abstract

We propose to search for common abstractions for concurrency models to enable multi-language virtual machines to support a wide range of them. This would enable domain-specific solutions for concurrency problems. Furthermore, such an abstraction could improve portability of virtual machines to the vastly different upcoming many-core architectures.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors; D.1.3 [*Programming Techniques*]: Concurrent Programming

General Terms Design, Languages, Performance

Keywords Multi-language virtual machines, concurrency, many-core, abstraction, machine model, parallel programming models

1. Problem Statement

Since the processor manufacturers reached the boundaries of what is feasible to achieve computational speedups in terms of increased clock rates, they changed their scaling dimension from *clock rate* to *core count*, i. e., the number of computing units on a single chip. With this change, they are still able to deliver more computing power with every new processor generation by shifting the burden of realizing speedups to the software developers[9]. However, for the development of end-user applications, today's systems still lack comprehensive support to make concurrency accessible and its complexity manageable.

The most widely used programming model for concurrency is shared memory with threads and locks. Unfortunately, this model has very narrow limits. Even though fine-grained locks have proven to allow high-performance concurrency, programming complexity increases fast with a rising number of threads and shared resources. Therefore, this model does not scale up to the degree of concurrency in many-core system. The use of disciplined concurrency models avoiding shared state, is almost the only choice to cope with the inherent complexity. With the actor model and software transactional memory, there are promising candidates available. However, the general forms of these models also have their problems, thus, we expect domain-specific solutions for the different application domains to become an important approach to handle concurrency.

Another aspect of the changing processor designs is an increase in the diversity of processor architectures. Processor designers experiment with various different approaches to arrange cores on the chip and to connect them to the memory system, using various different caching strategies, and possibly explicit inter-core communication. For instance, the Cell B.E., Intel's Larrabee[6], and Tilera's TILE architecture[11] are fundamentally different especially with respect to their memory architecture.

High-level language virtual machines (VMs) like the *Java Virtual Machine* (JVM) and *Microsoft's Common Language Runtime* (CLR) have become the major means to tackle these kind of problems and are used as multi-language runtime environments. Ongoing efforts like the introduction of the `invokedynamic` bytecode extend the capabilities of this platform to handle a large number of different programming paradigms[5].

However, concurrency is a concept for which VMs do not provide sufficient abstraction. The JVM and CLR have rudimentary support for threads and locks in their instruction set architectures, and Erlang as notable exception provides explicit support for the actor model in its BEAM opcodes set[3]. But there is no VM which exposes more than one concurrency model to the programmer or provides means to abstract from the different concrete concurrency models provided by the hardware. We expect that VMs have to

SPLASH'10, October 17–21, 2010, Reno/Tahoe, Nevada, USA.
ACM 978-1-4503-0240-1/10/10.

[1] Supported by a doctoral scholarship of the Institute for the Promotion of Innovation through Science and Technology in Flanders, Belgium.

handle both aspects to be able to provide software developers with the necessary, possibly domain-specific tools to cope with concurrency.

2. Research Goal

Our main goal is to identify common abstractions of different abstract concurrency models which are also appropriate to be mapped efficiently onto the various upcoming many-core architectures[4]. Thus, the concurrency models on the different levels of implementation are to be decoupled. For this, we will experiment with an VM with explicit support for some form of disciplined concurrency. The VM has to provide appropriate abstractions to generalize the broad range of abstract concurrency models to be usable as a foundation for new language designs.

The analysis and design is approached iteratively to find an appropriate compromise between the different instances of the currently most important concurrency models as well as between these models itself. Currently, actors[1, 8], software transactional memory[7], and shared-memory models are most important. The analysis and design process will iterate over the different concurrency models and chose a suitable compromise for the different instances of the current model under investigation. The compromise will be guided from the viewpoint of the language designer as well as from the viewpoint of the VM implementer.

Important is also the mapping on a concrete concurrency model provided by a specific hardware architecture. The simplest, but still important one is an intra-core communication. This is the standard case for single-core processors, possible with multiple hardware threads. The next step is a uniform memory access model like it is used for current multi-core systems and could be used for subsets of cores on many-core systems as well. For real many-core systems only a non-uniform memory access model is feasible. At least this three concrete models have to be considered in the iterations to be able to provide a suitable mapping from the VM to different hardware architectures. Distributed systems, i. e., systems based on a number of physical nodes connected by a network are not regarded by this project. Instead, the main focus of this research are on the challenges with respect to processor internal communication and the influence of the different many-core architectures.

The results of this research should enable us to decouple abstract and concrete concurrency models by using an VM with inherent concurrency support. New abstract concurrency models can be implemented on top of the VM and a new concrete concurrency model or a new many-core architecture can be supported by the generalization the VM provides.

3. Current State and Future Work

Currently, we completed the initial phase of literature studies and prototyping of ideas. Thus, we investigated the state of the art in concurrency support for virtual machines[3] and experimented with support for threads and locks, as well as actor abstractions on the instruction set level[4]. Our experiments included also an analysis of high-level concurrency constructs with a focus on barrier synchronization. However, they turned out to be to divers and their high-level character did not match the requirements for a concept that needs to be directly supported by virtual machines.

Our future work will be based on the work of Ungar and Adams[10]. With this multi- and many-core virtual machine, we have a foundation for experimenting on the TILE architecture as well as commodity multi-core systems.

Furthermore, we will investigate the notions of locality and encapsulation as fundamental concepts to concurrency. Encapsulation refers to the guarantee given to an entity, for instance an object or an actor, that its internal state is only accessible by itself. Locality refers to the notion of a spacial relation between entities. For instance, the objects grouped together in an partitioned global address space model. We aim to evaluate their capabilities to facilitate virtual machine support for different concurrency models[2].

References

[1] G. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.

[2] S. Marr. Encapsulation and locality: A foundation for concurrency support in multi-language virtual machines? In *Proc. of SPLASH 2010 - Doctoral Symposium*, 2010. (to appear).

[3] S. Marr, M. Haupt, and T. D'Hondt. Intermediate language design of high-level language virtual machines: Towards comprehensive concurrency support. In *Proc. of VMIL'09*, pages 3:1–3:2. ACM, October 2009. (extended abstract).

[4] S. Marr, M. Haupt, S. Timbermont, B. Adams, T. D'Hondt, P. Costanza, and W. D. Meuter. Virtual machine support for many-core architectures: Decoupling abstract from concrete concurrency models. In *Prof. of PLACES'09*, 2010.

[5] J. R. Rose. Bytecodes meet combinators: Invokedynamic on the jvm. In *Proc. of VMIL'09*, pages 1–11. ACM, 2009.

[6] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: A many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27(3):1–15, 2008.

[7] N. Shavit and D. Touitou. Software transactional memory. In *Proc. of PODC'95*, pages 204–213. ACM, 1995.

[8] S. Srinivasan and A. Mycroft. Kilim: Isolation-typed actors for java. In *Proc. of ECOOP'08*, pages 104–128, 2008.

[9] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobbs Journal*, 30(3), 2005.

[10] D. Ungar and S. S. Adams. Hosting an object heap on many-core hardware: An exploration. In *Proc. of DLS'09*, 2009.

[11] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, 2007.