

Many-Core Virtual Machines

Decoupling Abstract from Concrete Concurrency

Stefan Marr 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[12]. In the fields of operating systems, middleware, databases, and distributed systems, solutions have been found to provide support for concurrency. 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 the shared memory model with threads and locks. Unfortunately, this model has very narrow limits. Even though threading and fine-grained locking 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, but currently not part of mainstream software de-

velopment. 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. uses a distributed memory model with an inter-core ring network for explicit memory transfers[3]. Intel's Larrabee uses also a ring for inter-core communication, but provides shared memory with cache-coherency[8]. A different approach was taken with MIT's RAW processor which was commercialized by Tiler. Its TILE architecture combines cores in a 2D mesh with five special-purpose inter-core networks[14]. Thus, in the age of many-core systems, processors do not only vary over instruction sets, but they provide fundamentally different inter-core communication characteristics which need to be accounted for.

High-level language virtual machines (VMs) have become the major means to tackle these kind of problems. They are used increasingly as multi-language runtime environments. Typical examples are the *Java Virtual Machine* (JVM) and *Microsoft's Common Language Runtime* (CLR). Even so the JVM was originally not designed as a multi-language VM, today it hosts, similar to the CLR, various different languages and programming models. Ongoing efforts like the introduction of the `invokedynamic` bytecode extend the capabilities of this platform to handle a large number of different programming paradigms[7].

However, concurrency is a concept for which VMs do not provide sufficient abstraction. They abstract from memory management, language specific, and platform specific characteristics by providing an abstract machine model represented by an instruction set architecture (ISA) which brings portability as well as it allows to build highly optimizing just-in-time compilers for the different supported hardware architectures. Yet, these machine models do not provide abstractions for different concurrency models[5]. The JVM and CLR have rudimentary support for threads and locks in their ISAs, and Erlang as notable exception provides explicit support for the actor model in its BEAM opcodes set. 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. With respect to the many-core evolution, we expect that VMs have to handle both aspects to be able to provide software developers with the necessary, possibly domain-specific tools to cope with concurrency.

2. Research Goal

The main goal of our research 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[6]. Thus, the concurrency models on the different levels of implementation are to be decoupled. For this, we experiment with an VM instruction set architecture, i. e., a VM model with explicit support for some form of disciplined concurrency. The ISA has to provide appropriate abstractions to generalize the broad range of abstract concurrency models to be usable as a foundation for new language designs. Furthermore, it is necessary to devise a methodology which enables language-designers to map their abstract concurrency model onto such an ISA. The methodology also needs to provide means to VM developers to map from this ISA to concrete concurrency models, since the differences in the hardware design will be significant.

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, 11], software transactional memory[9], and shared-memory models are most important. Even if the latter have conceptual difficulties with respect to many-core concurrency, because the programming model with fine-grained locking does not scale, they are important to allow a transition from these models to more disciplined models. Furthermore, recent research in the area of partitioned global address space (PGAS) languages indicates that their might be disciplined shared memory approaches[2, 10]. 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. After each iteration, the insights will be consolidated to a methodology which then will be applied and refined in the following iteration.

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. Like mentioned before, inter-core ring or mesh networks with or without shared-memory are currently proposed[3, 8, 14]. At least this three concrete models have to be considered in the iterations to be able to provide a suitable mapping from the ISA 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 ISA with inherent concurrency support. New abstract concurrency models can be implemented on top of the ISA and a new concrete concurrency model or a new many-core architecture can be supported by the generalization the ISA 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[5] and experimented with support for threads and locks, as well as actor abstractions on the

instruction set level[6]. 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[13]. 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[4].

Acknowledgments

Stefan Marr is supported by a doctoral scholarship of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen), Belgium.

References

- [1] G. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.
- [2] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: An object-oriented approach to non-uniform cluster computing. In *Proc. of OOPSLA'05*, pages 519–538. ACM, 2005.
- [3] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26(3):10–23, 2006.
- [4] 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).
- [5] 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).
- [6] 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 *Proc. of PLACES'09*, volume 17 of *Electronic Proceedings in Theoretical Computer Science*, pages 63–77, February 2010.
- [7] J. R. Rose. Bytecodes meet combinators: Invokedynamic on the jvm. In *Proc. of VMIL'09*, pages 1–11. ACM, 2009.
- [8] 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.
- [9] N. Shavit and D. Touitou. Software transactional memory. In *Proc. of PODC'95*, pages 204–213. ACM, 1995.
- [10] J. Shirako, D. M. Peixotto, V. Sarkar, and W. N. Scherer. Phasers: A unified deadlock-free construct for collective and point-to-point synchronization. In *Proc. of ICS'08*, pages 277–288. ACM, 2008.
- [11] S. Srinivasan and A. Mycroft. Kilim: Isolation-typed actors for java. In *Proc. of ECOOP'08*, pages 104–128, 2008.
- [12] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobbs Journal*, 30(3):202–210, 2005.
- [13] D. Ungar and S. S. Adams. Hosting an object heap on manycore hardware: An exploration. In *Proc. of DLS'09*. ACM, 2009.
- [14] 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.