Encapsulation And Locality

A Foundation for Concurrency Support in Multi-Language Virtual Machines?

Stefan Marr[1]

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

Abstract

We propose to search for common abstractions for different concurrency models to enable high-level language virtual machines to support a wide range of different concurrency models. This would enable domain-specific solutions for the concurrency problem. Furthermore, advanced knowledge about concurrency in the VM model will most likely lead to better implementation opportunities on top of the different upcoming many-core architectures. The idea is to investigate the concepts of encapsulation and locality to this end. Thus, we are going to experiment with different language abstractions for concurrency on top of a virtual machine, which supports encapsulation and locality, to see how language designers could benefit, and how virtual machines could optimize programs using these concepts.

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

General Terms Experimentation, Languages, Performance

Keywords Multi-language virtual machines, concurrency, many-core, abstraction

1. Problem Statement

High-level language virtual machines (VMs) have become a standard platform for software development. They provide abstraction from the underlying system in terms of hard- and software. Furthermore, they provide highly optimized just-in-time compilers and garbage collection to provide performance characteristics comparable to classic low-level system programming languages. Recent improvements and additions to VMs like the Java Virtual Machine or Common Language Runtime enable efficient execution of a wide range of dynamic languages. These dynamic capabilities are used increasingly to build domain-specific languages (DSLs) on top of these VMs. In return, DSLs enable developers to tackle their problems at an even higher level of abstraction. However, VMs have not made the step into the many-core era by supporting language designers to utilize concurrency and parallelism.

The main question here is, are there common abstractions for the various concurrency models? To provide the necessary flexibility to language designers, a VM should support a wide range of different concurrency models. Shared memory with threads and locks is the standard, transactional memory promises to handle some of the software engineering challenges, actor-like message passing systems avoid typical low-level concurrency issues, and data-flow programming is a good fit for a number of computing problems. They all have application where they shine, and use-cases which are not supported that well. For a multi-language VM, choosing a single model does not seem to be appropriate.

Implementing an unsupported models on top of a VM comes usually with significant additional complexity as well as performance disadvantages. Furthermore, the language specific implementation of a model on top of a VM is typically not reusable for other languages. Examples are actor languages for the JVM, which according to Karmani et al. either do not support reliable encapsulation to avoid performance problems, or have to introduce complex type systems to achieve their goal[7]. The research in the field of software transactional memory (STM) suggest direct changes to the runtime systems to achieve the desired performance, too[1, 12].

To achieve real abstraction instead of merely adding one concurrency model after another, we are searching for fundamental commonalities. These commonalities should allow language developers to implement their ideas more easily, while providing the VM with additional opportunities for optimizations to gain performance.

By approaching the question of a common abstraction, the problem of integrating different concurrency models becomes most relevant, too. One fundamental questions is, how a module, that is written with shared-memory libraries, interacts with a module, which is based on the assumptions of non-shared memory and needs to enforce strong encapsulation. A typical scenario is legacy code. For instance, legacy programs written with shared-memory have to integrate seamlessly with new modules, which could utilizes an actor-based model that requires strict encapsulation. Other relevant scenarios include building domain-specific concurrency abstractions, which also require interaction between modules that use different concurrency models.

2. Goal

The concepts we currently see as most relevant to provide such an abstraction are encapsulation and locality. 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 encapsulated by an actor could be grouped together in memory, which belongs to the same location. To find a suitable representation for a VM for the notion of locality is part of this

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.

research. Thus, our goal is to experiment with virtual machine support for these concepts and evaluate their usefulness in a multi-language VM for many-core architectures.

To this end, concrete incarnations of the different concurrency models, i.e., languages need to be analyzed to see how they could benefit from these two concepts. Furthermore, these abstractions have to be evaluated with respect to the different upcoming many-core architectures like Tilera's tile architecture[11].

These experiments will either indicate the applicability of the chosen concepts to reach the desired goal, or will provide the necessary indications to choose more suitable ones. In case of a positive result, we expect to gain arguments which allow us to design concrete low-level constructs, which need to be supported by multi-language virtual machines to allow an implementation of a wide range of different concurrency models on top of them. Furthermore, these results should provide us with the understanding to propose concrete optimization strategies for the VM implementations on the different many-core architectures.

To facilitate interoperability between different concurrency models on top of the same VM, a framework needs to be provided, which either predefines certain rules, or gives the language designers the means to specify how other modules, libraries, or languages are supposed to interact with this language. Currently, the literature discusses a number of pair-wise combinations of different concurrency models to propose possible semantics for their interaction[4, 10]. However, for the proposed abstraction such a semantics has to be more general and go beyond the approach of finding semantics which hold only for pairs of models.

3. Research Approach

As a foundation for this research, we started with an literature survey including the implementation strategies for *partitioned global address space* (PGAS) languages. They use the notion of locality to divide a global address space based on physical computational nodes. Furthermore, their implementations typically combine shared memory concurrency with a message-passing infrastructure, and thus, utilize different concurrency models to implement a language on top. The candidates for non-shared-memory, actor-like languages are are E[8] and AmbientTalk[9]. They represent a very compelling approach for object-oriented languages. For functional languages, Erlang is know for its concurrent nature. Based on this literature study we will choose a number of language concepts to experiment with. At the moment, we expect to evaluate concepts from UPC and X10 as representative models for shared-memory concurrency and E, AmbientTalk, and Erlang for actor-like concurrency.

Our goal is to implement the chosen language abstractions prototypically on top of an existing virtual machine. The next step is to incorporate support for encapsulation and locality at the VM level. Thus, we will design a virtual machine model, with an extended instruction set and presumably a sketch for a memory model semantics. Based on these facilities, we want to rebuild the same language abstractions. This will enable us to assess the different engineering efforts to build domain-specific language abstractions on top. Furthermore, we also expect to get an initial intuition about the performance related aspects on a Tilera 64-core processor.

With respect to the interoperability between different concurrency models, we plan to investigate how VM support could be designed to provide enough flexibility to language designers but also provide the necessary guarantees to the different concurrency models. Currently, we plan to provide the concept of encapsulation in a flexible and possibly configurable way. Thus, it could be enabled selectively only depending on the guarantees a module/language requests. One way of doing this, would be in a way comparable to X10, in which all shared-memory access from a model are transformed to messages and handled safely, iff the accessed module requires strict encapsulation.

Furthermore, we will investigate how the notion of locality can facilitate VM implementations on many-core architectures. Here a relevant question is how can the concept of locality be represented flexibly in the VM. The problem we like to address is, that PGAS languages have the notion of locality on the level of computation nodes, which means large object heaps, where AmbientTalk typically uses smaller groups of objects as an actor, and in Erlang as the other extreme, actors are typically on the level of very small groups of objects down to one object per locality.

To investigate the generality of our approach, the experiments need to be extended to at least one other model. STM seems to be a relevant candidate. While researchers struggles to reduce the overhead to an acceptable level, hardware vendors start to provide the foundations for hardware-assisted TM. Thus, they provide mechanisms which are itself limited, but improve the performance of STM systems on top[2, 3]. Based on their proposals, we see a strong correlation of STM to the notion of locality. Usually, CPUs leverage cache-line locking or marking as an implementation strategy that provides only very restricted freedom in what can be done in a single transaction. At the moment, we believe that such a mechanism could be provided based the VM support for locality and encapsulation. This would allow to implement sophisticated STM systems on top of the restricted model the VM would offer. Even such a restricted system would directly facilitate the implementation of concurrent data structures[5, 6].

References

[1] A.-R. Adl-Tabatabai, B. T. Lewis, V. Menon, B. R. Murphy, B. Saha, and T. Shpeisman. Compiler and runtime support for efficient software transactional memory. In *Proc. of PLDI'06*, pages 26–37. ACM, 2006.

[2] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum. Hybrid transactional memory. In *Proc. of ASPLOS'06*, 2006.

[3] D. Dice, Y. Lev, M. Moir, and D. Nussbaum. Early experience with a commercial hardware transactional memory implementation. In *Proc. of ASPLOS'09*, pages 157–168. ACM, 2009.

[4] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 1992. ISBN 1558601902.

[5] D. Hendler, N. Shavit, and L. Yerushalmi. A scalable lock-free stack algorithm. *J. Parallel Distrib. Comput.*, 70(1):1–12, 2010.

[6] M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*, 21(2):289–300, 1993.

[7] R. K. Karmani, A. Shali, and G. Agha. Actor frameworks for the jvm platform: A comparative analysis. In *Proc. of PPPJ'09*. ACM, 2009.

[8] M. S. Miller, E. D. Tribble, and J. Shapiro. Concurrency among strangers: Programming in e as plan coordination. In R. D. Nicola and D. Sangiorgi, editors, *Symposium on Trustworthy Global Computing*, volume 3705 of *LNCS*, pages 195–229. Springer, April 2005.

[9] T. Van Cutsem, S. Mostinckx, E. G. Boix, J. Dedecker, and W. De Meuter. Ambienttalk: Object-oriented event-driven programming in mobile ad hoc networks. *Proc. of SCCC'07*, pages 3–12, 2007.

[10] T. Van Cutsem, S. Mostinckx, and W. De Meuter. Linguistic symbiosis between event loop actors and threads. *Computer Languages, Systems & Structures*, 35(1), June 2008.

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

[12] P. Wu, M. M. Michael, C. von Praun, T. Nakaike, R. Bordawekar, H. W. Cain, C. Cascaval, S. Chatterjee, S. Chiras, R. Hou, M. Mergen, X. Shen, M. F. Spear, H. Y. Wang, and K. Wang. Compiler and runtime techniques for software transactional memory optimization. *Concurrency and Computation: Practice & Experience*, 21(1), 2008.