Available M.Sc. Thesis Project;

Weak Memory Model Error Detection in the SIMICS Simulator

Project Description

The overall motivation for the project is to develop automated techniques for finding errors (bugs) in concurrent programs that execute on multicore (and multiprocessor) shared-memory architectures. In this particular project, the goal is to develop automated techniques for detecting bugs that are caused by the incorrect use of a weak memory model. In particular, bugs caused by the potential reordering and different local orderings of memory operations as seen by different processors.

In a multiprocessor, read and write operations that are performed by different processors, may not be executed on the shared memory in the same order as one would expect by looking at the program (technically, this means that the execution is not "sequentially consistent"). A careful programmer can avoid such situations by inserting global synchronization operations, such as memory system barriers, atomic instructions, and synchronizing instructions. Using such instructions will limit the achievable parallelism, and hurts performance. Therefore, when performance is important, programmers will try to use as few synchronization instructions as possible. Overly aggressive optimization can lead to subtle errors, as the memory system is not told to synchronize enough to maintain correct program semantics under all conditions. Writing correct and high-performance shared-memory code at this level is known to be very hard and intellectually challenging.

Code that is aware of weak memory models is most commonly found in low-level operating system code and optimized concurrency libraries, e.g., the Intel Threading Building Blocks or the `java.util.concurrent` package. Such frameworks support the programmer by providing concurrent implementations of familiar data abstractions such as locks, queues, sets, or maps. Since these algorithms use subtle synchronization, and in addition are exposed to a weak memory model, they are notoriously hard to get correct. Therefore, bug-detecting techniques are important.

In some cases, weak memory models can also be exposed to user-level code. It is in general not possible to entirely hide the memory model of the underlying machine to user-level code that uses shared memory. Typically, errors caused by incorrect use of weak memory results in rare, impossible-to-reproduce bugs that only show up under high processing loads or unusual execution patterns.

In this particular project, the aim is to develop techniques by which bugs are detected using a full-system simulator. The full-system simulator runs the complete software stack of the target system, including operating system, libraries, and user code. The simulator has full control over executions of the system, and can non-intrusively and deterministically observe and analyze all aspects of an execution, including the interaction between processors and the shared memory. We want to use this obsevation power to detect code that has incorrect or insufficient synchronization code, and which could fail to work correctly.

We will use the Simics full-system simulator, originally developed by Virtutech, and now developed by Intel and sold commercially by Wind River (Wind River is an Intel wholly-owned subsidiary).

The task of the project is to develop techniques for monitoring Simics simulations using additional bookkeeping machinery, in order to detect situations where weak memory system ordering could result in an incorrect value being used by code on a processor in the system. Tentatively, such an extension to Simics could report whenever code fails to apply the necessary synchronizations to ensure a deterministic program execution.

Project Organization

The project will be carried out at Uppsala University, in the *Algorithmic Program Verification* group (supervisors Prof. Bengt Jonsson and M.Sc. Carl Leonardsson), and at the *Intel* Simics development center in Stockholm (supervisor TBD) , as well as in cooperation with Wind River (supervisor Dr. Jakob Engblom).

Contact: Bengt Jonsson, Uppsala university: `bengt@it.uu.se`

Potential Background Reading

The following could be useful reading when starting the project, or for finding out whether you might be interested.

- *Weak Memory Models:* Getting familiar with the problems of weak memory reorderings. There are several tutorials available. Examples of papers are
  Sarita V. Adve, Kourosh Gharachorloo: Shared Memory Consistency Models: A Tutorial. IEEE Computer 29(12): 66-76 (1996)
  Dennis Shasha, Marc Snir: Efficient and Correct Execution of Parallel Programs that Share Memory. ACM Trans. Program. Lang. Syst. 10(2): 282-312 (1988)
- *Monitoring Techniques for Races and Reorderings:* There are some recent papers that should be read. For example, a comparatively recent work is in *Jacob Burnim, Koushik Sen, Christos Stergiou: Sound and Complete Monitoring of Sequential Consistency for Relaxed Memory Models. Proc. TACAS 2011: LNCS Vol. 6605, pp. 11-25.*
- Simics. There is material available in the department.