

Devilish Execution of Multithreaded Programs

Francesco.Zappa_Nardelli@inria.fr

Parallelism is finally going mainstream, but, despite 40 years of research on concurrency, programming and reasoning about concurrent systems remains very challenging.

A key issue is that most programmers (and most researchers) assume that memory is sequentially consistent: that accesses by multiple threads to a shared memory occur in a global-time linear order. Real multiprocessors and compilers, however, incorporate many performance optimisations. These are typically unobservable by single-threaded programs, but some have observable consequences for the behaviour of concurrent code. For example, on standard Intel or AMD x86 processors, given two memory locations x and y (initially holding 0), if two processors $\text{proc}:0$ and $\text{proc}:1$ respectively write 1 to x and y and then read from y and x , as in the program below, it is possible for both to read 0 in the same execution.

iwp2.3.a/amd4	proc:0	proc:1
poi:0	MOV [x]←\$1	MOV [y]←\$1
poi:1	MOV EAX←[y]	MOV EBX←[x]
Allow: 0:EAX=0 \wedge 1:EBX=0		

One can view this as a visible consequence of *write buffering*: each processor effectively has a FIFO buffer of pending memory writes (to avoid the need to block while a write completes), so the reads from y and x can occur before the writes have propagated from the buffers to main memory. Such optimisations (of which this is a particularly simple example) destroy the illusion of sequential consistency, making it impossible (at this level of abstraction) to reason in terms of an intuitive notion of global time.

Project aim. Reorderings such as those described above happen rarely when running multithreaded programs: it is thus difficult to test that a program is robust against such behaviours. In this internship we will develop a tool to run multithreaded programs against a memory that systematically produces non-sequentially consistent reorderings of memory accesses. Since the tool will have complete control of the execution path and reorderings performed by the memory subsystem, it will simplify testing and debugging of multithreaded programs.

Prerequisites. This project requires familiarity of the C programming language, and some knowledge of assembly programming. This project will build on a semantics of parallel execution of x86 processors, called x86-TSO, that defines the view of memory that each processor can have. We will use tools such as Pin or Valgrind to perform binary code rewriting to intercept the memory accesses performed by the program being tested.

This stage will be carried out in the *Moscova Research Team* at INRIA Rocquencourt under the direction of Francesco Zappa Nardelli. To obtain more detailed informations feel free to contact the author of the proposal.

References

- [1] The Semantics of Multiprocessor Machine Code,
<http://moscova.inria.fr/~zappa/projects/weakmemory> .
- [2] Owens, Sarkar, Sewell, *A Better x86 Memory Model: x86-TSO*, TPHOLs 2009.