Analysis and optimization of Synchronization Algorithms for Multicore Architectures

Masoud Hemmatpour, Renato Ferrero, Bartolomeo Montrucchio, Maurizio Rebaudengo
Dipartimento di Automatica e Informatica, Politecnico di Torino
{masoud.hemmatpour, renato.ferrero, bartolomeo.montrucchio,
maurizio.rebaudengo}@polito.it

Multicore design is a major issue in modern computer architectures. Programmers are urged to design innovative algorithms by exploiting multicore facilities. Since synchronization affects the performance of multithread algorithms, the selection of an effective synchronization mechanism is critical for multicore environments. Modern computers provide special hardware instructions that allow to atomically read and modify the content of a word (e.g., the *cmpxchg* instruction in Intel x86 CPUs), so they can be used for synchronization of threads. Moreover, software techniques can synchronize threads without any dependency on hardware instructions [1]. This study considers the main synchronization techniques [1, 2], such as *Ticket lock*, which guarantees fairness execution to all threads, *Filter lock*, which is intended for multiple threads, *Readers-writer lock*, which aims to solve the readers-writers problem and *Read-Copy Update* (RCU), which reduces the overhead in readers-writer lock. The first contribution of this study is to evaluate the costs of the mentioned synchronization techniques, due for example to *memory access*, *system call*, and *spinning*, i.e., the act of querying (or in some cases modifying) an object in memory and waiting for its content. In order to reduce the costs of the mentioned synchronization mechanisms, state-of-the-art approaches exploit hardware or software techniques. The second contribution of this study is the analysis of both hardware and software solutions to reduce the synchronization costs. Moreover, a comparative study to highlight benefits and drawbacks of the different synchronization mechanisms has been performed.

Different software solutions such as *backoff*, a waiting time to reduce the bus traffic, *non blocking algorithm*, a synchronization mechanism without blocking primitives, and *compiler barrier*, a compiler directive to avoid reordering of the instructions, are well-known techniques which are investigated in this study.

Beside software solutions, hardware manufacturers introduce various facilities in shared memory or distributed environment to enhance the performance of synchronization mechanisms. Examples of hardware solutions are *hardware message passing* and *different layer of caches* in the shared memory environment, and *Remote Direct Memory Access* (RDMA) in distributed environment.

Experimental benchmarks have been executed on a node of cluster (Opteron 6276 2.3 GHz CPU with 16 cores and running CentOS 6.3 Operating System2). The experiments, which are intended to represent a useful aid for researchers and practitioners interested in optimization of parallel algorithms, show that:

1. The update rate directly impacts on performance, even if a non blocking algorithm is exploited.
2. The cost of keeping data locality should not exceed the cost of cache misses.
3. Exploiting a non blocking synchronization algorithm (i.e., RCU) leads to a better performance.
4. Critical section length should be reduced as much as possible in order to increase the performance.
5. In order to reduce the bus traffic, it is better to avoid spinning.
6. Hardware message passing can increase the performance of shared memory synchronization model.
7. Synchronization methods with heavy instructions should be avoided.

References

[1] Herlihy and Shavit. "The art of multiprocessor programming." Elsevier, 2008.

[2] Is Parallel Programming Hard, And, If So, What Can You Do About It? https://www.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook-e1.pdf: Accessed: 2016-07-25.