Chapter 1

Efficient Hardware-Supported Synchronization Mechanisms for Manycores

José L. Abellán$^{\Phi}$, Juan Fernández$^{\Psi}$ and Manuel E. Acacio$^{\Omega}$
$^{\Phi}$*Boston University* $^{\Psi}$*Intel Labs Barcelona* $^{\Omega}$*University of Murcia*

1.1 Introduction

Data centers are evolving by hosting emerging parallel and distributed applications such as cloud computing, streaming video or social networking. These new applications demand not only more storage capacity or network bandwidth, but also higher performance and throughput thus requiring multicore processors as building blocks of every computational node or server [27].

As more and more cores are being integrated on chip, manycore architectures [17, 36, 38] have emerged as the next generation of multicores. Manycores are systems specially tailored to the exploitation of massive throughput by incorporating many simple and low-frequency computing units. This paradigm shift towards throughput-oriented servers leads to parallel workloads with ever more number of threads that need to communicate and synchronize among them, typically relying on a single shared memory domain per server.

In that context, conventional implementations of synchronization operations, such as barrier and locks, make use of shared variables which are atomically updated. In particular, when considering global barriers and highly-contended locks (i.e., a significant amount of threads requesting the lock at the same time), without the proper hardware support, typical software-based implementations cannot provide good scalability as the number of cores increases.

Regarding barrier implementations in software, as we will discuss in Section 1.3, the use of shared variables creates a performance bottleneck, and demands not only a significant amount of resources but also of energy consumption. In more depth, the cache coherence protocol must come into play to maintain memory consistency across all levels of the memory hierarchy. In turn, coherence activity translates into traffic injection in the interconnection network that may interfere with application-related traffic. On the other hand, the busy-waiting required to wait for the completion of the barrier synchronization on locally-cached shared variables has also significant implications on the energy consumed by the L1 caches.

As to the software implementations for highly-contended locks, as we will expose in Section 1.8, they are critical to performance since lock contention causes serialization. Therefore, an implementation based on the use of shared variables is not efficient enough due to the perfor-

[41] W. T.-Y. Hsu and P.-C. Yew. An Effective Synchronization Network for Hot-Spot Accesses. *ACM Transactions on Computer Systems*, 10(3):167–189, 1992. 1.7

[42] Z. Hu, J. del Cuvillo, W. Zhu and G. R. Gao. Optimization of Dense Matrix Multiplication on IBM Cyclops-64: Challenges and Experiences. In *Proceedings of the 12th International European Conference on Parallel and Distributed Computing*, 2006. 1.7