Evaluating The Performance of Non-Blocking Synchronisation on Shared-Memory Multiprocessors*

Philippas Tsigas
Department of Computing Science
Chalmers University of Technology
SE-412 96 Göteborg,Sweden
tsigas@cs.chalmers.se

Yi Zhang
Department of Computing Science
Chalmers University of Technology
SE-412 96 Göteborg,Sweden
yzhang@cs.chalmers.se

ABSTRACT

Parallel programs running on shared memory multiprocessors coordinate via shared data objects/structures. To ensure the consistency of the shared data structures, programs typically rely on some forms of software synchronisations. Unfortunately typical software synchronisation mechanisms usually result in poor performance because they produce large amounts of memory and interconnection network contention and, more significantly, because they produce convoy effects that degrade significantly in multiprogramming environments: if one process holding a lock is preempted, other processes on different processors waiting for the lock will not be able to proceed. Researchers have introduced non-blocking synchronisation to address the above problems. Non-blocking implementations allow multiple tasks to access a shared object at the same time, but without enforcing mutual exclusion to accomplish this. However, its performance implications are not well understood on modern systems or on real applications. In this paper we study the impact of the non-blocking synchronisation on parallel applications running on top of a modern, 64 processor, cache-coherent, shared memory multiprocessor system: the SGI Origin 2000. Cache-coherent non-uniform memory access (ccNUMA) shared memory multiprocessor systems have attracted considerable research and commercial interest in the last years. In addition to the performance results on a modern system, we also investigate the key synchronisation schemes that are used in multiprocessor applications and their efficient transformation to non-blocking ones. Evaluating the impact of the synchronisation performance on applications is important for several reasons. First, micro-benchmarks can not capture every aspect of primitive performance. It is hard to predict the primitive impact on the application performance. For example, a lock or barrier that generates a lot of additional network traffic might have little impact on applications. Second, even in applications that spend significant time in synchronisation operations, the synchronisation time might be dominated by wait time due to load imbalance and lock serialisation in the application, which better implementations of synchronisation may not be helpful in reducing. Third, micro-benchmarks rarely capture (generate) scenarios that occur in real applications.

We evaluated the benefits of non-blocking synchronisation in a range of applications running on top of modern realizations of shared-memory multiprocessors, a 64 processor SGI Origin 2000. In this evaluation, i) we used a big set of applications with different communication characteristics, making sure that we include also applications that do not spend a lot of time in synchronisation, ii) we also modified all the lock-based synchronisation points of these applications when possible. The goal of our work was to provide an in depth understanding of how non-blocking can improve the performance of modern parallel applications. More specifically, the main issues addressed in this paper include: i) The architectural implications of the ccNUMA on the design of non-blocking synchronisation. ii) The identification of the basic locking operations that parallel programmers use in their applications. iii) The efficient non-blocking implementation of these synchronisation operations. iv) The experimental comparison of the lock-based and lock-free versions of the respective applications on a cache-coherent non-uniform memory access shared memory multiprocessor system. v) The identification of the structural differences between applications that benefit more from non-blocking synchronisation than others. We selected to examine these issues, on a 64 processor SGI Origin 2000 multiprocessor system. This machine is attractive for the study because it provides an aggressive communication architecture and support for both in cache and at memory synchronisation primitives. It should be clear however that the conclusions and the methods presented in this paper have general applicability in other realizations of cache-coherent non-uniform memory access machines. Our results can benefit the parallel programmers in two ways. First, to understand the benefits of non-blocking synchronisation, and then to transform some typical lock-based synchronisation operations that are probably used in their programs to non-blocking ones by using the general translations that we provide in this paper.

*This work is partially supported by: i) the national Swedish Real-Time Systems research initiative ARTES (www.artes.uu.se) supported by the Swedish Foundation for Strategic Research and ii) the Swedish Research Council for Engineering Sciences.

Experiments and Main Results

The SGI Origin 2000 that we used has 64 195MHz MIPS R10000 CPUs with 4MB L2 cache and 15.5GB main memory. We used a large group of applications, some of which are from the SPLASH-2 [3] suite, and some of which were developed more recently and constitute the shared memory part of the Spark98 kernels suit [1]. More specifically from slash we used the following applications: i) Ocean, ii) Volrend, iii) Radiosity, iv) Water-Nsquared, v) Water-Spatial. Because we wanted to make the evaluation on realistic problem sizes for these multiprocessors, we selected large problem sizes that do not favour synchronisation, but still as we will show later the improvements were significant for most applications. Generally, the larger the problem size the lower the frequency of synchronisation relative to computation. After studying the applications that we had selected, we identified the lock-based high level synchronisation operations that they use. As a next step, we proposed a set of efficient lock-free implementations for these synchronisations. The description of the implementations are general enough and can be used in other parallel applications. These implementations together with the detailed modifications for each application can be found in the full paper [2].

The results from our experiments show that: 1) For Ocean there was no significant improvement after the modification, but, the non-blocking synchronisation do not hamper the performance of Ocean. Ocean is a regular application with very regular communication patterns and below 32 processors, the synchronisation time does not contribute much to the total execution time. Because the ocean application requires the number of processes to be power of 2, we could only do the experiments for up to 32 processors. 2) For Radiosity there was no big difference between the two versions (lock based one and non-blocking one) until we reached 32 processors where synchronisation became a significant part of the total computing time. With 32 processors, the non-blocking version is about 34% faster than the lock-based one and as the number of processors increases the improvement on the performance also increase reaching a 93% better performance when using 60 processors, the maximum number of processors that we could use exclusively for running this application. The access patterns to shared data structures in Radiosity are highly irregular. 3) For Volrend the performance advantages of the non-blocking synchronisation start to show as the number of processors becomes greater than 8. The performance of the non-blocking one is close to optimal since its speed up is very close to the theoretical limit. Volrend's inherent data referencing pattern on data that are written is migratory, while its induced pattern at page granularity involves multiple producers with multiple consumers. 4) For the Spark98 applications, due to the limited time for exclusive use that we had we performed the experiments for up to 28 processors for this application. The results, clearly show the power of non-blocking synchronisation for unstructured applications like this one. The speedup of the lock-based programs stops when we go above 16 processors while the non-blocking one continues to scale uniformly. This allows us to conjecture that non-blocking will dramatically increase the performance of these applications as the number of processors increases. 5) In Water-nsquared and Water-spatial the communication patterns and the sharing of the data is very simple: A process updates a local copy of the particle accelerations as it computes them, and accumulates into the shared copy once at the end. This simple communication pattern does not give the opportunity to lock-free synchronisation to show its power. On the other hand, the experiments show that lock-free synchronisation does not harm the performance of the applications. The lock-free versions of both applications perform as well as the respective lock-based ones.

To conclude: i) For the fairly wide range of applications examined, non-blocking synchronisation performs as well, and often better than the respective blocking synchronisation. ii) For certain applications, the use of non-blocking synchronisation yields great performance improvement. Figure 1 describes graphically the maximum speedup of the lock-free and the respective lock-based implementation for each of our implementations. With 60 processors, the non-blocking version of radiosity is about two times faster than the lock-based one; non-blocking Volrend is about 7 times faster that the lock based one. Irregular applications benefit the most from non-blocking synchronisation. Since the importance of such applications is likely to increase in the future, the importance of lock-free synchronisation in high-performance parallel systems is also expected to increase. iii) The methods that we introduced to replace lock based synchronisations are quite simple and general to be used in many parallel applications.

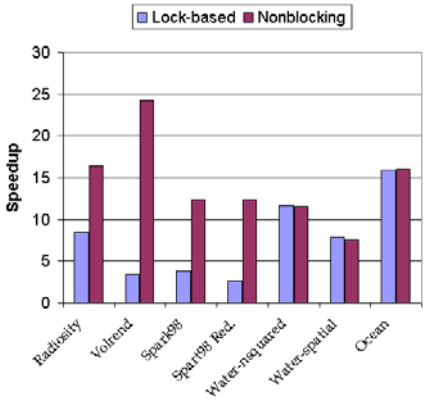

Figure 1: Speedup

1. REFERENCES

[1] D. R. O'Hallaron. Spark98: Sparse matrix kernels for shared memory and message passing systems. Technical Report CMU-CS-97-178, CMU, 1997.

[2] P. Tsigas and Y. Zhang. Evaluating the performance of non-blocking synchronisation on modern shared-memory multiprocessors. Technical Report 2000-2, Department of Computing Science, Chalmers University of Tech., 2000 (http://www.cs.chalmers.se/~tsigas/papers/TR2000-02.pdf).

[3] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedings of ISCA '95*, pages 24–36, 1995.