Performance Benefits of Relaxed Memory Consistency for Process Network Applications

Ronald Bos[1,2] Ben Juurlink[2]

[1]Philips Research Laboratory
The Netherlands

Delft University of Technology
The Netherlands

Abstract—**This work is the first to investigate whether using a relaxed consistency model has performance benefits over a sequential memory model when a multiprocessor executes a process network application.**

A trace-driven simulator, developed using SystemC, is used to model the distributed shared memory system of a prototype multiprocessor developed at Philips. The simulator offers two consistency models: Sequential Consistency (SC) and a generalized Relaxed Consistency (RC) model. Input traces are generated by running a process network application in a cycle-accurate simulator of the prototype multiprocessor.

The results show that relaxed consistency has marginal performance benefits (6.13 to 14.9 percent execution time drop) over sequential consistency.

I. INTRODUCTION

Philips is developing a scalable, parallel computer architecture that can be used as a computing engine in the full range of their electronics products. The multimedia applications which will run on this single chip multiprocessor exhibit streaming-like memory access patterns. Therefore, one of the programming paradigms for this Computer Architecture for Killer Experience (CAKE) is Kahn process networks [4] because multimedia applications can be implemented efficiently as process networks. High performance of this multiprocessor is important for a seamless user experience. Previous work has shown that the performance of a shared memory multiprocessor can be greatly influenced by its memory consistency model.

In this paper, results are presented of a study of the performance benefits of relaxed memory consistency over sequential consistency for process network applications running in CAKE. Our limited investigations show that for process network applications, relaxed memory consistency has marginal benefits over sequential consistency.

This paper is organized as follows. In Section II, the Philips multiprocessor is described. Also, a concise introduction to memory consistency is given. Section III-A then describes the experimental method used in this work. The results of the experiments are given in Section IV. Conclusions are drawn in Section V.

II. BACKGROUND

In this Section, the Philips multiprocessor is described. Also, a concise introduction to memory consistency is given.

A. Philips CAKE

The Philips CAKE is a nonuniform, distributed shared memory multiprocessor designed for scalability in terms of number of nodes but also in terms of effortless and seamless integration of (third party) intellectual property. One node (or *tile*) of a CAKE system is a single chip mul-

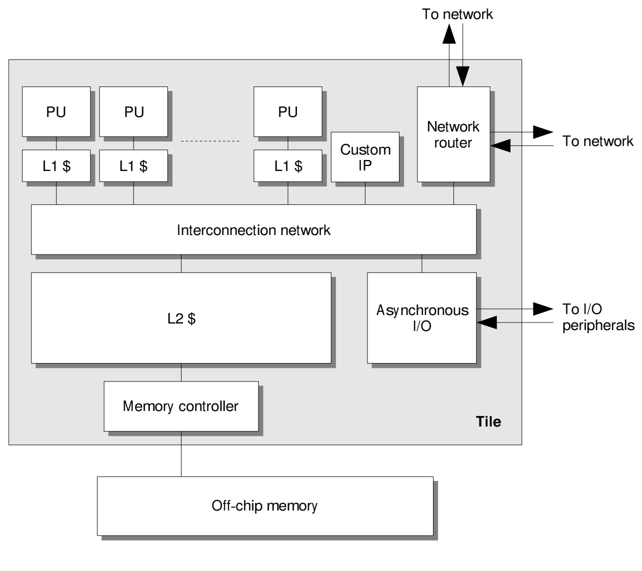

Fig. 1. Contents of one CAKE tile.

tiprocessor. Figure 1 shows the contents of one CAKE tile. A tile contains TriMedia and MIPS processors with their private L1 caches, custom IP blocks, a network interface, a shared, banked L2 cache, input/output circuitry, and a memory controller. All these elements are connected by a central, scalable interconnection network which also provides invalidation based, snoopy coherence. Off-chip DRAM (Dynamic Random Access Memory) may be connected to the tile. Multiple tiles may be connected to create an even bigger multiprocessor. In a multi-tile CAKE system, the inclusive L2 caches are kept coherent by an invalidation and home based directory protocol.

B. Memory Consistency Models

Culler et al. give the following definition of memory consistency in [2]: "A memory consistency model ... specifies constraints on the order in which memory operations must appear [to the processors in the system] to be performed with respect to one another". Memory consistency, in contrast to coherence, reigns the order between *any* pair of memory references. An important aspect of memory consistency is *visibility*. The order between the memory references that is promised by a memory consistency model has to be visible only whenever a processor bothers to look. This gives the system designer freedom to overlap and reorder (parts of) memory operations in order to enhance the system performance. As long as the *visible* order is in agreement with the promised memory consistency model, these behind-the-scene reorderings are perfectly valid.

The most important memory consistency model is *sequential consistency* (SC). It was introduced in 1979 by Lamport in [5]. A popular graphical model of SC was introduced by Adve et al. in [1] and is reproduced in Figure 2. Every processor P_i issues memory operations in

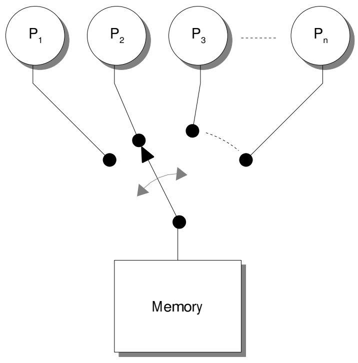

Fig. 2. Programmer's view of sequential consistency.

program order. Only one of the processors at a time can be connected to the memory. It then performs one or more atomic operations on the memory. After this, the switch is set randomly. Thus, under sequential consistency, all processors observe the same global ordering between the memory operations of the individual processors.

Researchers realized that a strict ordering between *any* pair of memory references is usually not needed. In a lot of programs, only a strict ordering between synchronizing memory accesses and the block of memory operations between the synchronizing memory accesses is needed. This led researchers to invent relaxed memory consistency models. Relaxed memory consistency models have in common that they allow the processors in the system to see different orderings between all or selected categories of memory operations. A relaxed memory consistency allows the system designer to expose reorderings, that occur as a result of architectural optimizations, to the programmer. This usually removes any overhead needed to provide the illusion of SC when the system would implement SC instead of a relaxed memory consistency model.

A machine that implements a relaxed memory consistency model always provides some kind of *safety net* which allows a programmer to enforce a sequential order between (sets of) memory operations. The safety net allows correct execution of parallel programs on multiprocessors that have a relaxed memory consistency model. A commonly used safety net is some form of *memory barrier* instruction. Depending on its type, certain or all types of memory operations may not be reordered with respect to a memory barrier.

III. Experimental Method

A. Overview of the Experimental Method

In order to determine whether a relaxed memory consistency model has benefits over sequential consistency for process network applications, we wanted to simulate a process network benchmark application in the cycle accurate simulator of CAKE, `cakesim`. However, the directory system in this simulator was not finished at the time our research started. Therefore, we used the approach as shown in Figure 3.

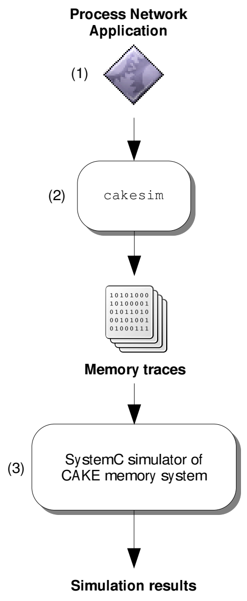
Process Network Application
(1)
(2)
cakesim
Memory traces
(3)
SystemC simulator of CAKE memory system
Simulation results

Fig. 3. Experimental setup.

Memory trace files are generated by running the process network benchmark application (1) in the execution based `cakesim` simulator (2). A custom built SystemC simulator (3) runs those memory traces through a model of CAKE's DSM system and returns simulation results.

B. *Custom Built* CAKE *Simulator*

Figure 4 shows one simulator tile, which contains models of processors, a shared L2 cache and shared memory with its directory controller. Multiple tiles may be connected to form a massive, coherent multiprocessor.

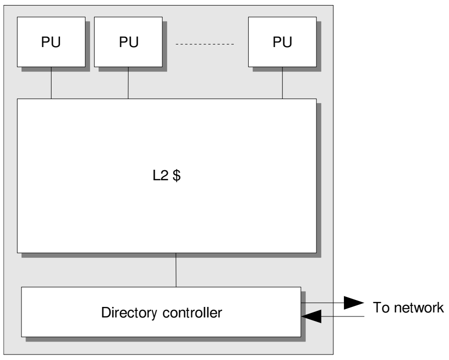

Fig. 4. Contents of one CAKE simulator tile.

The private L1 caches have been left out of the model because of time constraints. Because the L2 cache is inclusive, this will have minimal influence on the number of L2 coherence actions via the directory protocol. However, the absence of the L1 caches will increase the execution time of the simulated system.

The simulated processors support a configurable maximum number of outstanding references and include a memory barrier instruction. A processor stalls when the maximum number of outstanding references is reached or when a memory barrier is detected (details are explained in Section III-C). No data dependency checks are done when the processor issues a memory reference. The instruction window has an unlimited size, which means that memory references that are hundreds of places apart in the trace file may be concurrently outstanding due to a very long latency of the 'oldest' memory reference. The processors support semaphore based synchronization which allows the modeling of a FIFO with a limited size. No process migration is supported, so any producer or consumer process is tied to one specific processor during a simulation run.

The L2 cache is banked and supports an infinite number of simultaneous requests from the processors. When any new request maps to a cache block that is currently in transaction, this new request is buffered until the transaction has been finished.

The directory protocol is a simple, strict request–response, invalidation and home based protocol. The home replies with a negative acknowledgment (NACK) to requests for memory blocks that are already in a transaction. This idea is taken from the directory protocol used in the Silicon Graphics Origin2000.

The shared memory that is being kept coherent by the directory protocol consists of the concatenation of all FIFO buffer memory, thus forming a single, contiguous range of shared memory. The rest of the memory is not shared and therefore not handled by the directory protocol. The block of shared memory is distributed evenly over the tiles; each tile is assigned an equal portion of the shared memory with a size of

$$M_i = \frac{M}{T} \tag{1}$$

bytes, where M_i is the memory size for tile i, M is the total memory size and T is the number of tiles.

The network models a router with constant delay. It does not model contention.

Implementation of the simulator took approximately six months, during which 5,490 lines of code[1] were written.

C. *Simulator Memory Consistency Models*

The simulator implements two memory consistency models: (1) straightforward sequential consistency and (2) a generic relaxed consistent model. The sequential model is based on three sufficient conditions which, when implemented in a multiprocessor, guarantee a sequential consistent system. These three conditions can be derived from work done by Dubois et al. in [3] and [7] and are adapted by Culler et al. in [2, p.289]; they are (begin quote):

1. Every process issues memory operations in program order.
2. After a write operation is issued, the issuing process waits for the write to complete before issuing its next operation.
3. After a read operation is issued, the issuing process waits for the read to complete, and for the write whose value is being returned by the read to complete, before issuing its next operation. That is, if the write whose value is being returned has performed with respect to this processor (as it must have if its value is being returned), then the processor should wait until the write has performed with respect to all processors (end quote).

In correspondence with these conditions, under sequential consistency, a processor issues the references from the trace file in sequential order and waits for each reference to complete before it issues the next reference. A reference is considered complete when the cache returns the value (for a read) or signals write completion. The third condition is guaranteed by the directory protocol, which does not allow any read to an address for which the write (whose value is being returned by the read) has not yet globally performed.

The generic relaxed model is implemented by allowing a processor to simultaneously issue r_{max} references

[1]Line count obtained with `sloccount`.

to the cache. As soon as any of these outstanding references completes, a new reference is issued to the cache. Because the outstanding references may have different latencies, in the long run the sequential ordering is violated. When the processor detects a memory barrier, it stops issuing new references to the cache and waits until all currently outstanding references have completed. Only then the memory barrier is retired and new memory references are issued to the cache.

D. *Process Network Application Model*

Figure 5 shows the process network benchmark application used in this research. It consists of only a producing and a consuming node which are connected by an 8 kilobyte FIFO. The FIFO buffer memory is located in

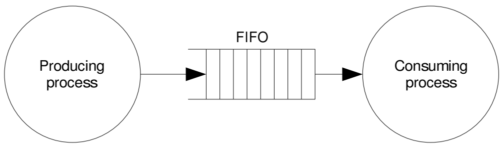

Fig. 5. Process network benchmark application.

shared memory and the processes directly manipulate this memory. The processes communicate 256 byte FIFO elements. Inside these elements, they read or write 4 byte integers. Because these memory accesses are to sequential, monotonously increasing memory addresses, we expect that multiple outstanding references from any processor will have a high chance of mapping to the same cache block. Therefore, we expect that relaxing the memory model will not increase performance spectacularly.

E. *Communication Patterns*

The way the processes and FIFO buffers are distributed over the tiles influences the network communication characteristics of the simulated system. For example, if two processes and the FIFO buffer they access are assigned to the same tile, all inter-process communication will happen inside the tile (via the cache) and no network communication will occur. Another extreme would be to locate the producer and consumer process on different tiles and their corresponding FIFO buffer on a third tile. This situation would lead to maximum network communication and cache usage.

Figure 6 shows the choice for the arrangement of the processors and FIFO buffer memory that has been made for the experiments in this work. In this configuration, minimal network communication occurs because the processes that communicate via a slice of shared memory reside on the same tile as the shared (FIFO) memory itself. For an

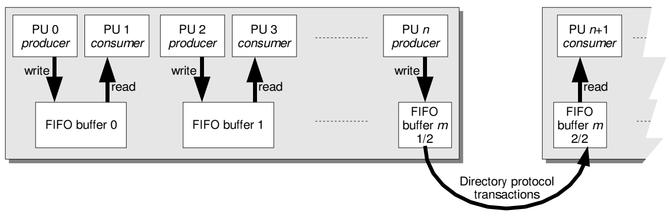

Fig. 6. Distribution of processes and FIFO buffers over the simulator tiles (for an uneven number of processors per tile).

uneven number of processors per tile, only the first or second half of one process' FIFO buffer is assigned to the tile. The other half is assigned to the neighboring tile. These two processes therefore communicate via the network.

IV. Experimental Results

In this Section, we describe the results of two experiments. In the first experiment, the simulator was configured to model a CAKE system in order to investigate the benefits of relaxed memory consistency for CAKE. In the second experiment, the simulator was configured to maximize network communication in order to investigate the influence of the network latency on the performance of the memory system. For both experiments, the execution time of the processors was measured. The execution time of a processor is defined as the cycle in which the last outstanding reference completes after the trace file has been depleted. During one simulation run, the process networks communicate 50,000 FIFO blocks of 256 bytes.

A. *Benefits of Relaxed Memory Consistency for CAKE*

Figure 7 shows the simulator configuration for this experiment. Like a real CAKE system, this simulator con-

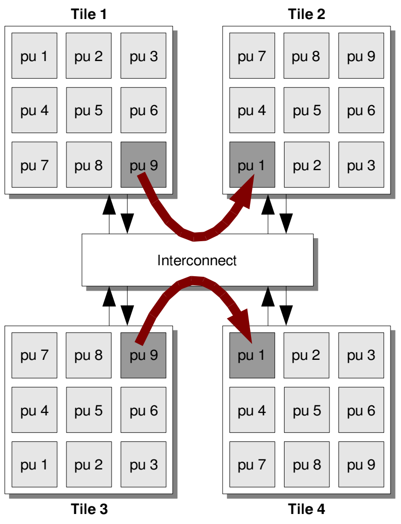

Fig. 7. CAKE-like simulator configuration.

figuration consists of 4 tiles and each tile contains 9 processors. The tiles are connected by the network. Because of the arrangement of the processes and shared memory over the tiles (which is described in Section III-E), only the dark gray processes communicate via the directory protocol. The light gray processes communicate via the shared cache (not shown in the figure).

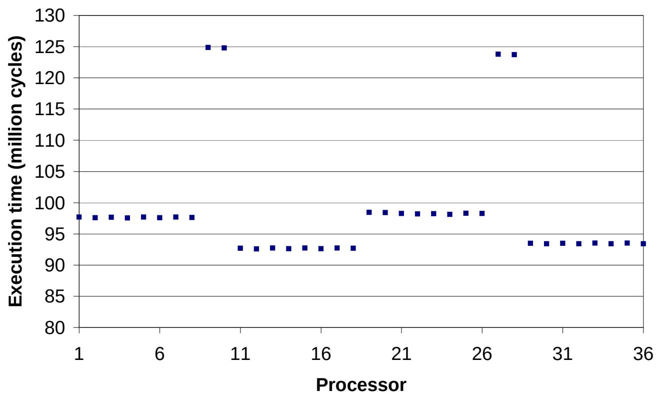

Fig. 8. End times of individual processors in the CAKE-like simulator configuration (shown in Figure 7) for $r_{max} = 3$.

Figure 8 shows the execution time of the individual processors for the first experiment for $r_{max} = 3$ (r_{max} is the maximum number of outstanding references a processor is allowed to have). In this Figure, processors 1–9 reside on tile 1, processors 10–18 reside on tile 2, and so on. The Figure shows that the processes that communicate via the network (8, 9, 26 and 27) take 26 to 35 percent longer to finish than the processes that do not communicate via the network. The difference between the execution times of the sets of intra-tile–communicating processors is only at most 6.1 percent. Therefore we define the execution times of respectively the cache- and network-communicating processes as the average of the end times of the corresponding subsets of processors.

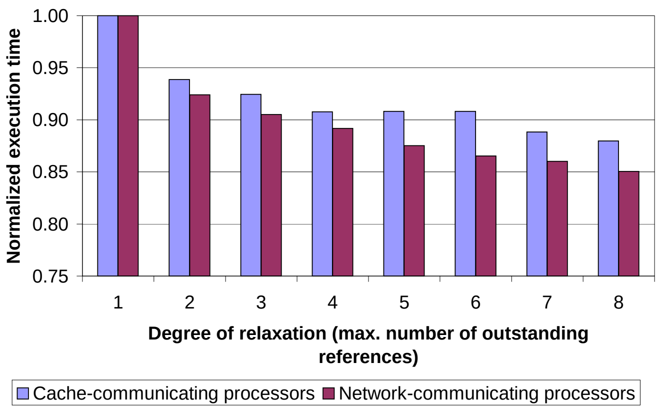

Fig. 9. Normalized execution times for the CAKE-like simulator configuration.

Figure 9 shows the normalized execution times of the cache- and network-communicating processors versus r_{max}, the number of outstanding references a processor is allowed to have; $r_{max} = 1$ is equivalent to sequential consistency. The first thing we conclude from this graph is that relaxing the memory model reduces the execution time by 6.13 to 12.0 percent for intra-tile communicating processes and by 7.59 to 14.9 percent for inter-tile communicating processes, so relaxing the memory model has performance benefits. We do not know which value of r_{max} is realistic for the process network workload; this issue is left for future investigation. The second conclusion is that the largest performance increase is observed when r_{max} is raised from 1 to 2 outstanding references. In this case, the decrease in execution time is 6.13 percent for cache-communicating processors and 7.59 percent for network-communicating processors. Increasing r_{max} further only gives relatively small performance improvements.

The overall conclusion we draw from the first experiment is that for our small process network benchmark application, relaxed memory consistency has marginal benefits over sequential consistency in a CAKE-like system.

B. *Network Influence on the Benefits of Relaxed Memory Consistency*

In order to investigate the influence of the network latency on the simulation results, experiments have been done with a simulator configuration as shown in Figure 10. The system consists of eight tiles. Because each tile contains only one processor, the influence of the cache on the simulation results is minimized.

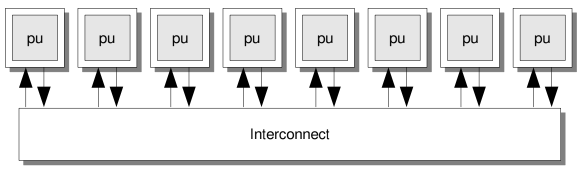

Fig. 10. Simulator configuration that emphasizes the influence of the network and directory protocol on the process network execution time.

Figure 11 shows the normalized execution times of the processors versus r_{max}; nine series are shown for nine values of L, the network round trip latency. This chart shows that the network latency has a strong influence on the speedup achievable by relaxing the memory model: the lower the network latency L, the higher the speedup that can be gained by raising the maximum number of outstanding references r_{max}. In other words, the higher the network latency, the lower the advantage of relaxing the memory model.

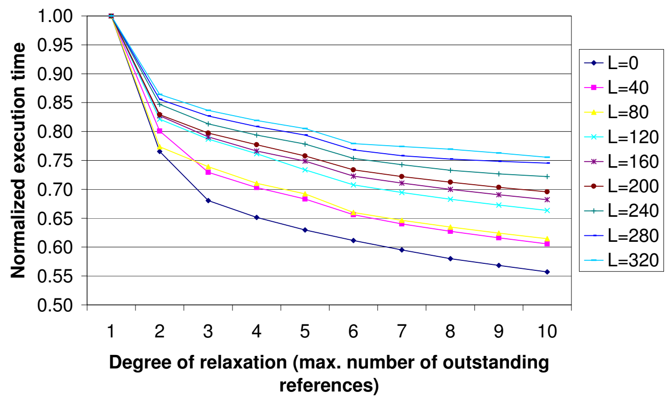

Fig. 11. Normalized process network execution time for a multiprocessor with 8 tiles and 1 processor per tile for varying degrees of relaxation.

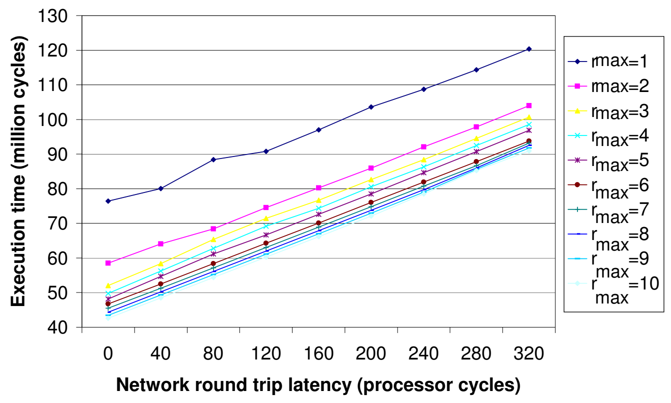

Fig. 12. Normalized process network execution time for a multiprocessor with 8 tiles and 1 processor per tile for varying degrees of relaxation.

Figure 12 shows the absolute execution times of the processors versus the network round trip latency; ten series are shown for ten values of r_{max}. First of all, the relatively big gap between the $r_{max} = 1$ and $r_{max} = 2$ series in this graph confirms the statement that increasing r_{max} from 1 to 2 gives most of the performance benefits. But there is a caveat here. The fact that the average slopes of the lines in this graph are approximately equal is at least interesting. But this is simply caused by the fact that the network does not model contention. Would the network model include contention, then the lines would have been diverging for increasing L.

The overall conclusion we draw from the second experiment is that in our experiments, the advantage of relaxed memory consistency decreases for increasing network latency.

V. Conclusions and Future Work

We have performed experiments with a model of the CAKE multiprocessor that offers two memory consistency models: (1) straightforward sequential consistency and (2) a generic relaxed consistent model. The benchmark application was a small producer–consumer process network application connected by an 8 kilobyte FIFO. The following conclusions can be drawn from the limited benchmark and simulator used in this research:

- Relaxed memory consistency has marginal benefits (6.13 to 14.9 percent execution time drop) over sequential consistency.
- The largest performance increase is observed when r_{max}, the maximum number of outstanding references a processor is allowed to have, is raised from 1 to 2 outstanding references.
- The advantage of relaxed memory consistency decreases for increasing network latency.

However, the limitations of the simulator and the benchmark prohibit final conclusions. We recommend to use either `cakesim` or RSIM [6] for more investigations and to reach more reliable and/or comparable statements about this subject. The use of `cakesim` has the benefits that it most closely resembles CAKE itself. Furthermore, because it is an execution-driven simulator, it can simulate real-world process network applications. The use of RSIM also allows us to simulate real-world applications because it is also execution-driven and very detailed. Furthermore, the use of the freely available and well-known RSIM would allow us to compare the research to other work performed with this simulator which would give us a better indication of the validity of the simulation outcome.

References

[1] S. V. Adve and K. Gharachorloo, *Shared Memory Consistency Models: A Tutorial*, IEEE Computer **29** (1996), no. 12, 66–76.
[2] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, Inc., San Francisco, 1999.
[3] M. Dubois, C. Scheurich, and F. Briggs, *Memory Access Buffering in Multiprocessors*, Proceedings of the 13th annual international symposium on Computer Architecture, IEEE Computer Society Press, 1986, pp. 434–442.
[4] G. Kahn, *The Semantics of a Simple Language for Parallel Computing*, Proceedings of IFIP Congress 74 (J.L. Rosenfeld, ed.), North-Holland, 1974, pp. 471–475.
[5] L. Lamport, *How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs*, IEEE Transactions on Computers **C-28** (1979), no. 9, 690–691.
[6] Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve, *RSIM Reference Manual. Version 1.0*, Tech. Report 9705, Rice University, 1997.
[7] C. Scheurich and M. Dubois, *Correct Memory Operation of Cache-Based Multiprocessors*, Proceedings of the 14th annual international symposium on Computer architecture, ACM Press, 1987, pp. 234–243.