Validating a token coherence protocol for scientific workloads

Ricardo Fernández-Pascual
r.fernandez@ditec.um.es

José M. García
jmgarcia@ditec.um.es

Manuel E. Acacio
meacacio@ditec.um.es

Departamento de Ingenier´ıa y Tecnolog´ıa de Computadores
Universidad de Murcia, 30071-Murcia (Spain)

ABSTRACT

Token coherence provides a flexible framework for designing new coherence protocols which decouples performance from correctness, easing the design of efficient coherence protocols. In this work, we have implemented a coherence protocol for a cc-NUMA architecture based on TokenB using the RSIM performance simulator to validate previous claims about token coherence performance and viability. Unlike previous works, we have used scientific workloads to evaluate the protocol performance against a directory based coherence protocol and we have found that it provides a significant speedup on average. Also, we have created our own simulation environment based on RSIM which can perform faster simulations of larger systems for scientific benchmarks. Our results show that token coherence protocols are a good alternative to directory based coherence protocols which can provide improved performance and reduced complexity. Also, we have compared the results obtained with our simulation environment with those obtained using GEMS for the same benchmarks and we have found them similar, giving further credibility to both our simulation environment and GEMS.

Categories and Subject Descriptors

C.1.4 [**Computer Systems Organization**]: Parallel Architectures—*Distributed architectures*

General Terms

Performance,Verification

Keywords

Cache coherence, Token protocol

1. INTRODUCTION

Shared-memory machines are the most flexible architecture currently used for scientific, technical and commercial workloads due to the convenience of the programming model. In particular, cc-NUMAs are shared memory machines where the memory is physically distributed among processor nodes. These machines add additional hardware to provide the illusion of a single shared memory common to all processors, avoiding or minimizing the problem of manual data distribution. Programmers must still divide their computation into parallel tasks, but all the tasks can work with a single common dataset resident in memory. This model significantly reduces the difficulty inherent in parallel programming, especially for applications that exhibit dynamic communication patterns.

To provide the illusion of a shared memory while keeping high performance, shared-memory machines use private caches whose coherency must be kept through hardware coherence protocols. Until recently, there were two main kinds of cache coherency protocols: snoopy protocols (which rely on a logical shared and totally ordered bus) and directory protocols (which keep track of each copy of a data block).

The scalability of SMP machines, whose coherency protocols are based on *bus snooping* is limited by their need of a totally ordered interconnection network like a shared bus. In this kind of network, the demand of bandwidth for the shared bus grows very quickly as the number of processors increases. This makes impractical and cost-ineffective to build machines of this kind with a great number of processors.

On the other hand, cc-NUMA machines use coherence protocols based on directories [1, 2]. These protocols do not rely on a totally ordered network and require only point-to-point communications which allow the use of a number of different network topologies. Hence they are significantly more scalable, because although the total bandwidth must scale too, the necessary bandwidth between each pair of nodes does not increase as much. However, the directory is used as the sequencing point for requests to the same memory block and it introduces additional latency, specially in cache-to-cache transfer misses [3], which are quite frequent. Another major constraint of cc-NUMA scalability is the memory overhead incurred by the directory information.

Coherence protocols for shared-memory machines are hard to design and verify, specially in the case of cc-NUMA machines. The fine grain and asynchronous nature of communications along with the low-latency requirements lead to complex and infrequent corner cases which are hard to detect and deal with.

Token coherence [9] is a novel approach to design cache coherence protocols for distributed shared memory machines which has been recently proposed. Token coherence protocols avoid the need of a totally ordered network and the introduction of additional latency in the common case. Several variants of token coherence protocols are possible assuming different tradeoffs of interconnection bandwidth and request latency.

Until now, token coherence protocols have been evaluated using commercial workloads like SPECjbb, Apache or OLTP using full system-simulation. It has been evaluated against traditional snooping and directory designs and against a coherence protocol modeled after that of the AMD's Hammer processors.

In this paper, we provide an independent evaluation of a token coherence protocol with a completely different simulation environment and using scientific instead of commercial workloads to validate previous claims.

We show that token coherence is a viable alternative to traditional directory based coherence protocols for designing efficient coherence protocols. For doing this, we implement a coherence protocol very similar to TokenB using the RSIM simulator [6] and compare it with a directory based protocol. We have performed simulations of both a 16 processor system and a 32 processor one finding that token coherence improves the execution time of the applications in both cases.

For comparison purposes, we have evaluated the original TokenB protocol using Multifacet GEMS simulator [10] with a configuration as close as possible to our simulator and running the same benchmarks with both TokenB and a directory protocol and we have obtained similar results, although not identical. The gaps are due to differences in the simulated architectures which could not be easily solved, like different available network topologies and distinct processor models which are described below.

The rest of this paper is organized as follows: section 2 describes the token coherence framework as used in this work. Section 3 details the coherence protocol that we have implemented and evaluated. Section 4.1 describes our simulation environment and the simulated architecture. Section 4.2 shows and evaluates our results. Finally, section 5 presents the conclusions of this work.

2. TOKEN COHERENCE

Token coherence [8, 9] is a framework for designing coherence protocols whose main asset is that it decouples the correctness substrate from several different performance policies. This allows a great flexibility, making possible to adapt the protocol for different machines efficiently using a wide range of interconnection topologies and different number of processors [7]. Thanks to decoupling correctness from performance, the different performance policies do not have to deal explicitly with all the infrequent corner cases and race conditions, allowing to make the common case as fast as possible and rely on the correctness substrate to handle the less important cases.

Decoupling different aspects of a problem is a trend in computer architecture that allows new, simpler and more aggressive solutions for different problems. Token coherence uses decoupling at several levels:

- Firstly, coherence and consistency are decoupled since token coherence provides a simple interface that allows implementing serial consistency. This makes possible to implement serial consistency or any relaxed consistency model at the processor level.
- Secondly, the protocol is decoupled from the interconnection network, being able to run on any interconnection system as long as it guarantees the eventual correct delivery of messages, without requiring any ordering constraint, not even point to point ordering.
- Thirdly, it decouples coherence from performance, since it provides a correctness substrate and a performance policy that builds upon it and does not need to be correct. This provides great flexibility to design a performance policy which is fast for the most common cases and does not need to worry about the infrequent corner cases which are usually hard to deal with correctly. Hence, coherence protocols based on token coherence can be adapted for different requirements and machine sizes more easily than snoopy or directory protocols [7].
- Finally, the correctness substrate also decouples safety from starvation. First, it enforces safety and correct semantics of cache coherence using *token counting* (it ensures that performed memory operations are correct). On the other hand, it avoids starvation using an infrequently invoked mechanism called *persistent requests* (it ensures that memory operations are eventually performed).

2.1 Ensuring safety

The main observation of the token framework is that simple token counting rules can ensure that the memory system behaves in a coherent manner. *Token counting* specifies that each block of the shared memory has a fixed number of tokens and that the system is not allowed to create or destroy tokens. A processor is allowed to read a block only when it holds at least one of the block's tokens and has valid data, and a processor is allowed to write a block only when it holds all of its tokens and valid data. These simple rules prevent a processor from reading the block while another processor is writing it, ensuring coherent behavior at all times.

One of the tokens is distinguished as the *owner token*. The processor or memory module which has this token is responsible for providing the data when another processor needs it or write it back to memory when necessary. The owner token can be either clean or dirty, depending whether the contents of the cache block are the same as in main memory or not, respectively. In order to allow processors to receive tokens without receiving data, a *valid-data bit* is added to each cache block (independently of the usual valid-tag bit).

In [7], basic token counting rules are introduced and then the rule set is extended to avoid always sending data with tokens and to support an *exclusive* state. These local rules lead to several global system invariants which are maintained by induction and allow to guarantee a correct behavior without reasoning about the interactions among intermediate protocol states and associated races.

We can relate token protocols with traditional MOESI protocols and define each of the states depending on the number of tokens that a processor holds:

0:	**I**nvalid state.
1 to $N-1$, but not the *owner token*:	**S**hared state.
1 to $N-1$, including the *owner token*:	**O**wned state.
N, dirty bit inactive:	**E**xclusive state.
N, dirty bit active:	**M**odified state.

2.2 Avoiding starvation

When a processor detects potential starvation, it issues a persistent request. Persistent requests, unlike transient requests, are guaranteed to eventually succeed. To ensure this, each token protocol must define how it deals with several pending persistent requests. Such requests may queue in a dedicated virtual network or at a queue in a persistent request arbiter, as long as they are served in a starvation free way (like a FIFO queue).

There are several ways to implement the arbitration for persistent requests: a single centralized arbiter, an arbiter for each home node, a banked arbiter to reduce contention, or distributed persistent request activation. However, ideally the performance policy should deal with most requests and avoid the necessity for persistent requests most of the time, making their implementation less critical from the point of view of performance. Persistent requests can deal with write and read requests using the same mechanism for simplicity.

2.3 Performance policies

Token coherence provides the framework for designing several particular coherence protocols. Building upon the correctness substrate, a variety of *performance policies* may be designed specifying the precise behavior of each processor and memory module to different coherence messages.

Performance policies are responsible for defining when and what transient requests are issued and how each component should react to them. Since transient requests are not guaranteed to succeed, every token protocol must have provisions to eventually resort to the persistent request mechanism when necessary. Each performance policy may target different aspects of efficiency: shorter latency for cache-to-cache transfer misses, better bandwidth efficiency, less power consumption, etc. Until now, three different performance policies have been described for non-hierarchical cc-NUMA systems:

Token-using-broadcast (TokenB) is a performance policy to simultaneously achieve low-latency cache-to-cache transfer misses and avoid an ordered interconnect. TokenB is faster than both traditional snooping protocols and directory protocols, although it requires more bandwidth [9]. Like traditional snooping protocols, TokenB broadcasts every transient request, but instead of using a shared bus like snooping protocols, it uses an interconnection network without any ordering guarantees. Due to races between processors some of these requests may fail to collect sufficient tokens. To handle those occasional situations TokenB reissues the transient request after a timeout period, ultimately relying on the correctness substrate's persistent request mechanism to prevent starvation after several retries. TokenB can resolve most misses using only two hops, including cache-to-cache transfer misses.

Token-based-directory (TokenD) emulates a directory based protocol using the token framework. Its main advantage is the bandwidth efficiency obtained by avoiding broadcasting requests. Instead of sending the transient request to every other component, the faulting cache sends only a request to a directory-like structure located at the home memory module and then the memory module sends transient requests to only the necessary nodes. Unlike in a traditional directory, the directory information does not need to be accurate, since the correctness substrate guarantees a correct execution anyway. In TokenD, most misses will be resolved using three hops. This extra indirection means that TokenD will perform worse than TokenB when there is enough bandwidth.

TokenM is a performance policy that seeks a compromise between bandwidth usage and latency. In TokenM, the requester processor uses destination set prediction to decide the recipients of the transient requests and uses multicast to send the requests to all of them. TokenM also has a soft directory at the home node like TokenD which receives all the requests and forwards them to the components missing in the predicted destination set that, according to its information, should have received the request too. TokenM has a sightly larger latency than TokenB and uses sightly more bandwidth than TokenD.

3. AN IMPLEMENTATION OF TOKEN COHERENCE ON RSIM

For our simulations we have used a modified version of RSIM (Rice Simulator for ILP Multiprocessors), a detailed execution driven simulator which models an out-of-order superscalar processor pipeline, a two-level cache hierarchy, a split-transaction bus on each processor node, and an aggressive memory and multiprocessor interconnection network subsystem [5].

In our target system, each node consists of a processor, a two-level cache hierarchy, a portion of the system's physical memory and its associated token count information, and a network interface. The network interface connects the node to a multiprocessor interconnection network for remote communication. We use a two-dimensional mesh network and wormhole-routing. We are restricted to employ this interconnection network since it was the only one implemented in RSIM.

Since the memory is physically distributed, data placement in our programs is either done explicitly by the programmer or implicitly using a first-touch policy on cache-line granularity. That is, each cache-line sized block of physical memory will be allocated at the first node that accesses it, unless the programmer cares to distribute the data manually to improve locality. Data distribution in most of our benchmarks is done manually. On the other hand, previous works based on GEMS did not use a physically distributed memory.

Token coherence defines a flexible framework for designing coherence protocols. In this work we evaluate a TokenB based protocol because we target medium sized machines (from 16 to 32 processors) and we think that the interconnection bandwidth will not be a problem for implementing TokenB in these machines. Since TokenB achieves the lowest miss latency of the polices described above when bandwidth is unconstrained, it is the best option from the point of view of performance. We have been able to create a coherence protocol which is very similar to TokenB, but may have small differences due to several implementation decisions and the target architecture.

We use token coherence to keep the coherence at the L2 cache level, using traditional states for L1, which uses a write-through and non-allocate policy on writes (previous works on token coherence assume a write-back L1). The L2 cache is write back with write allocate policy. The L2 cache maintains inclusion with respect to the L1 cache.

Our cache coherence protocol supports read, write and upgrade transient requests. Read requests are satisfied by the component (it can be a L2 cache or memory controller) which has the owner token when it receives the request, sending a reply with data and a single token. Write requests are satisfied by every node having at least a token when they receive the request, sending a message carrying all the tokens that the node had. Also, the message will include the data only if the replying node had the owner token. Upgrade requests are satisfied by every node, sending all their tokens without data. The original TokenB protocol did not support upgrade requests, but they were less important since the protocol included a migratory sharing optimization.

Since transient requests can fail to obtain enough tokens or data due to races between requests from different processors, the protocol reissues each request that has gone unsatisfied a certain amount of time. This timeout is adjusted dynamically using the technique described in [7]. Each transient request will be reissued up to three times. If the timeout expires even after the third retry, a persistent request will be raised.

Our version of TokenB uses a centralized persistent request activation policy with an arbiter at each node. Persistent requests are directed only to the persistent request arbiter responsible for the requested block, which is the one at the home node.

The persistent request arbiter state machine activates at most one request each time by sending a persistent activation message to each node. Each node responds with an acknowledgment (to avoid races) and remembers all the persistent requests that are active using a hardware table (there may be only one active persistent request for each memory block, but several persistent requests for different memory blocks may be active at the same time).

While a persistent request is active, the nodes must forward all tokens (and data, if they have the owner token) to the requester. They will also forward tokens and data that arrive later, because the request persists until the requester explicitly deactivates it. Note that every node but the requester will be invalidated by a persistent request even if the original request was a read request.

Table 1: Benchmarks and input sizes used in this work

Benchmark	Input Size
FFT	256K complex doubles
Ocean	258 × 258 ocean
Radix	2M keys, 1024 radix
Water-SP	512 molecules, 4 time steps
Water-NSQ	512 molecules, 4 time steps
Tomcatv	256 elements, 5 iterations
Unstructured	Mesh.2K, 5 time steps

Once the requester receives enough tokens and data, it satisfies the miss and sends a message to the arbiter at the home memory module to deactivate the request. The arbiter deactivates the request by informing all the nodes, who delete the entry from their table and send an acknowledgment. The arbiter considers the request deactivated once it has received all the acknowledgments.

The persistent request arbiter acts as the serialization point for persistent requests and ensures that at most one persistent request is active at any given time and that every persistent request is eventually activated using a FIFO policy.

Our token protocol does not include the migratory sharing optimization [11], since we wanted to compare it against a directory based protocol which does not include it either.

For being able to implement the described protocol, we have had to extend the network simulation layer of RSIM to support efficient broadcast (using dual-path routing as described in [4]). Notice that our directory based protocol does not use multicast.

4. VALIDATING THE TOKEN PROTOCOL FOR SCIENTIFIC WORKLOADS

We have used a set of scientific applications which cover a variety of computation and communication patterns to evaluate our protocol. The applications and the input sizes used are summarized in table 1.

FFT, Ocean, Radix, Water-SP, and Water-NSQ are from the SPLASH-2 benchmark suite. Tomcatv is a parallel version of a SPEC benchmark and Unstructured is a computational fluid dynamics application. The experimental results reported here correspond to the parallel phase of each program only.

4.1 Simulation environment

The modeled systems are cc-NUMA with 16 or 32 uniprocessor nodes. Table 2 summarizes the relevant parameters of the simulated systems. These values have been chosen to be similar to the parameters of the multiprocessors built in the near future considering current trends.

Table 2: Characteristics of simulated machine

16 or 32-Node System	
ILP Processor Parameters	
Processor speed	5 GHz
Max. fetch/retire rate	4
Instruction window	128
Branch predictor	2 bit agree, 2048 count
Cache Parameters	
Cache block size	64 bytes
L1 cache:	write-through
Size, associativity	32 KB, direct mapped
Hit time	2 cycles
Request ports	2
L2 cache:	write-back
Size, associativity	512 KB, 4 ways
Hit time	15 cycles
Request ports	1
Directory Parameters (when applicable)	
Directory controller cycle	1 cycle (on-chip)
Directory access time	6 cycles (L2 tag)
Message creation time:	
First coherence message	4 cycles
Next coherence messages	2 cycles
Memory Parameters	
Memory access time	300 cycles
Memory interleaving	4-way
Internal Bus Parameters	
Bus width	8 bytes
Bus cycles	1 cycle
Network Parameters	
Topology	2-dimensional mesh
Non-data message size	2 flits
Channel bandwidth	4 GB/s

To perform our simulations, we have used RSIM which provides a directory based coherence protocol. For the simulations using a token protocol, we have developed a new version of RSIM which models a protocol similar to TokenB as described above.

Recently, the GEMS simulator has been released by the Winsconsin Multifacet Project. GEMS is a timing simulator based on the Simics full system simulator and includes implementations for a TokenB protocol and a directory based protocol, amongst others. We have run our simulation using GEMS too for comparison purposes using the supplied TokenB and directory protocols. However, due to current limitations in Simics, we have only been able to use 16 processors for these simulations.

The configuration of the system simulated in GEMS is as close as possible to the one used in RSIM. However, differences between the simulators and the simulated architectures are inevitable (for example, GEMS uses a torus instead of a mesh for the interconnection topology). Also, we have not used Opal (the detailed processor model provided by GEMS) and we have used the in-order processor model provided by Simics. Using the simpler processor model allows much shorter simulation times.

RSIM is a much faster simulator than GEMS. This makes possible to scale up the problem size or the number of processors in the system while keeping reasonable simulation times. Currently, RSIM supports simulating systems with up to 64 processors.

For the same problem size, the simulation of a 32-processor

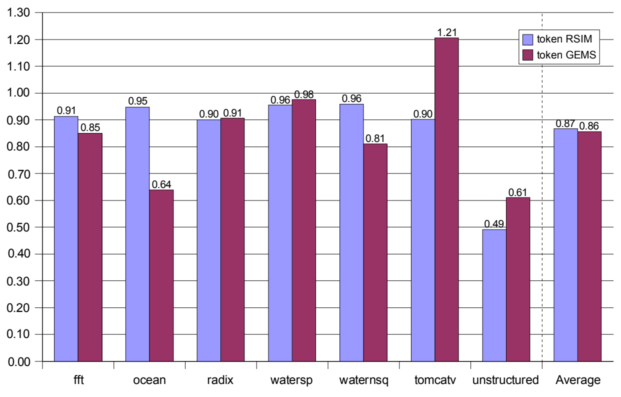

Figure 1: Normalized execution time of the token protocol with respect to the directory protocol for 16 processors using RSIM and GEMS.

system using RSIM which models a detailed ILP processor takes less time than the simulation of a 16-processor system, even when GEMS does not use its detailed processor model (Opal) but an in-order processor.

4.2 Results

In figure 1 we show the execution time of each benchmark for 16 processors using our token based protocol simulated with RSIM normalized with respect to the execution time using a directory based protocol. We see that every benchmark improves its execution time obtaining speedups from 4% for Water-NSQ and Water-SP up to 51% for Unstructured. The average improvement in execution time is 13%.

Figure 1 shows also the execution times of our benchmarks using GEMS simulating a 16 processors systems using TokenB compared to a directory based protocol. These results show higher speedups for most applications than those obtained using our protocol simulated with RSIM, although one of the applications obtains a much worse result, being actually slower when using the TokenB protocol. The average speedup obtained is 14%, which is similar to our results.

When we increase the number of processors up to 32, the average improvement in execution time is also 13% but now not every benchmark obtains better time using our token based coherence protocol. In figure 2 we can see the normalized execution time of each benchmark. Some applications improve their execution time even more than using 16 processors, but other improve less or even obtain worse execution times than the directory version. Unstructured is still the benchmark which obtains the best improvement, but it is now only 26%. FFT and Tomcatv obtain a slightly worse time too (1% and 3% worse respectively).

Looking at both figures we can see that the token based coherence protocol does not scale as well as the directory based protocol for some benchmarks. This is not totally unexpected, since the token based protocol relies on broadcasting every request and targets small and medium sized systems, unlike the directory based protocol which is more concerned with scalability. Hence, bigger machines using token based protocols may need to implement optimizations to

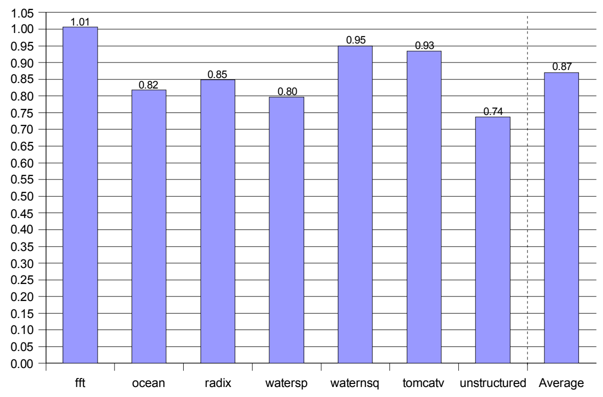

Figure 2: Normalized execution time of the token protocol with respect to the directory protocol for 32 processors.

Table 3: Percentage of reissued requests for 16 processors using RSIM

	Number of retries			
Benchmark	0	1	2	Persistent
FFT	99.68%	0.28%	0.01%	0.02%
Ocean	99.13%	0.72%	0.02%	0.13%
Radix	99.44%	0.55%	0.00%	0.01%
Water-NSQ	97.88%	1.56%	0.23%	0.34%
Water-SP	97.95%	1.19%	0.31%	0.56%
Tomcatv	96.32%	2.85%	0.17%	0.66%
Unstructured	97.40%	2.10%	0.20%	0.30%
Average	98.26%	1.32%	0.13%	0.29%

reduce bandwidth usage and use other performance policies like those described in [7]. However, we see that for medium sized machines like the simulated systems, the network is not a bottleneck in general and token coherence scales on average just as well as a directory based protocol.

In tables 3, 4 and 5 we see the percentage of misses satisfied with one, two or three transient requests or using a persistent request for GEMS and RSIM using 16 processors. First, we see than in every case the percentage of requests satisfied using only a single transient request is very high (more than 95% in the worst case). However, we see that as the number of processors increases to 32, the number of requests that need at least one retry more than doubles. This is because since there are more processors, it is more likely that two or more processors request the same line at once. This issue could limit the scalability of this token coherence protocol when using even more processors.

When comparing the behavior of RSIM and GEMS using 16 processors with respect to the number of retries, we see that our protocol makes less than half as many retries. This difference is probably due to the small differences in parameters like initial timeouts and the different behavior of the network. More retries do not necessarily imply worse performance, since some retries are not actually necessary[1] and in that case they do not hurt performance unless the network is congested.

[1]The answer for the first request may be already traveling through the network when the retry is issued.

Table 4: Percentage of reissued requests for 16 processors using GEMS

	Number of retries			
Benchmark	0	1	2	Persistent
FFT	95.83%	3.49%	0.47%	0.21%
Ocean	96.15%	3.52%	0.26%	0.07%
Radix	98.02%	1.86%	0.09%	0.03%
Water-NSQ	96.23%	3.35%	0.33%	0.08%
Water-SP	95.95%	3.47%	0.42%	0.16%
Tomcatv	93.50%	6.35%	0.12%	0.02%
Unstructured	96.42%	3.32%	0.20%	0.06%
Average	96.01%	3.62%	0.27%	0.09%

Table 5: Percentage of reissued requests for 32 processors using RSIM

	Number of retries			
Benchmark	0	1	2	Persistent
FFT	98.81%	0.87%	0.17%	0.16%
Ocean	96.49%	2.32%	0.35%	0.84%
Radix	98.01%	1.94%	0.02%	0.03%
Water-nsq	94.87%	3.59%	0.71%	0.84%
Watersp	93.28%	4.39%	0.96%	1.37%
Tomcatv	95.90%	3.10%	0.40%	0.60%
Unstructured	93.77%	5.14%	0.76%	0.33%
Average	95.88%	3.05%	0.48%	0.60%

Also, the persistent request mechanism implemented by the version of GEMS that we have used is different to the one provided by our protocol. Instead of centralized arbitration, it uses distributed arbitration which is more scalable and has less latency. We think that this and the fact that our implementation uses persistent requests more frequently than GEM's implementation is penalizing the performance of our implementation.

Although the average speedup obtained by both implementations of the token protocol with respect to the corresponding directory implementation is very similar (gap of 1%), there are some benchmarks that obtain very different speedups. Firstly, Unstructured obtains better improvement with our implementation than with GEM's implementation but we think that this difference is not very important since the improvement is huge in both cases (51% and 39% respectively, which are the best improvements for both simulators). On the other hand, Water-NSQ and Ocean obtain a much worse result using RSIM, while Tomcatv obtains a much worse one. The reason of these differences are hard to explain due to the many slight differences in the simulators and even in the directory protocols assumed for the base cases.

4.3 Lessons learned

Modeling the performance of multiprocessor systems is a resource hungry operation. There are several tradeoffs between accuracy, flexibility and simulation speed.

Contrary to GEMS using Simics, RSIM does not simulate a whole system, but only the few components which are interesting to measure the performance of the system running a certain kind of workloads. That is, RSIM can reasonably model the performance of cc-NUMA systems running scientific benchmarks without the burden of modeling input/output, the effect of the operating system calls, task scheduling, etc. On the other hand, these simplifications are not acceptable for running other kinds of workloads like commercial workloads for which full system simulators like GEMS can provide significantly more accurate measurements. However, these simplifications allow RSIM to scale easier than GEMS: RSIM can simulate the benchmarks presented in this work using 32 processors including a detailed processor model employing a time comparable to the time taken by GEMS using only 16 processors and an in-order processor model (without Opal) for the same benchmarks.

Our protocol does not perform exactly as the TokenB described in the literature due to differences in the architecture simulated at the processor and network levels, and differences in the protocol itself which have been already described. However, the results are relevant as a verification of the token coherence framework as a flexible tool to build efficient coherence protocols.

We have found that token coherence provides a very good framework for reasoning about the coherence protocol and significantly eases the implementation of the protocol compared to a traditional protocol. This is due to the correctness substrate that ensures that races between different processors cannot yield to an incoherence. However, those races can degrade performance due to the retries (and eventually persistent requests) which are needed to solve them. So, although it is easier to build a correct coherence protocol, there is still an important work required to fine-tune it and obtain good performance.

Also, token coherence adds some issues to the protocol by itself: most protocols based on token coherence will need a multicast enabled interconnection network to be implemented efficiently, some things are more difficult to implement or even impossible (like silent evictions), there is some overhead involved in token accounting, etc.

5. CONCLUSIONS

In this work we have presented a new and independent evaluation of a cache coherence protocol for cc-NUMAs based on token coherence. Our work validates previous claims about token coherence using both a different simulator and a different kind of benchmarks.

Firstly, unlike previous works describing token coherence protocols, we have used a different simulator than GEMS, which was developed by the same group that proposed the token coherence framework. Our simulator is based on RSIM, a performance simulator already widely used for evaluating cache-coherence protocols.

Secondly, we have used a different type of applications to evaluate the coherence protocol, namely scientific workloads (most of them from the SPLASH-2 benchmark suite) instead of commercial workloads. Scientific problems are an important use of high-performance shared memory computers and they behave quite different than commercial workloads. In particular, input/output performance is less important while the processors speed and communication among threads are more critical.

We have found that our version of the TokenB protocol performs better than a directory based protocol for our benchmarks using 16 processors, obtaining an average speedup of 13% with no slowdown in any of our applications when using 16 processors. In the case of a 32 processors machine, we have found an average speedup of 13% too, but one of the applications got 1% slower.

We think that these results could be improved tuning our protocol implementation, specially implementing a better persistent request arbitration method. Also, optimizations using prediction that are easier to implement within the token coherence framework and that would be specially helpful for bigger machines have not been explored. However, we have preferred to keep this initial implementation simple and not optimize it too much to avoid skewing the results in favor of the token based protocol.

For comparison purposes, we have evaluated the same applications using GEMS implementations of cache coherence based on tokens and directory for SMPs using a 16-processor configuration. We have found that GEMS implementation of token coherence outperforms GEMS implementation of directory coherence by 14%, which is very similar to the average speedup obtained with our implementation.

Our simulation platform based on RSIM trades flexibility for simulation speed to be able to simulate larger systems or larger problem sizes. In this way, although RSIM can only simulate scientific workloads (or workloads that do not rely very much on the operating system behavior or input/output) it can do it much faster than a full system simulator like GEMS with Simics, which can simulate other types of workloads too. We think that this is an acceptable tradeoff if we are mostly interested in one kind of applications.

Finally, we think that token based protocols are a viable alternative to traditional coherence protocols and the token coherence provides a very flexible and simple framework to develop new coherence protocol and optimizations.

Acknowledgements

We want to thank the anonymous reviewers for their insigthful comments and valuable suggestions, which have helped to improve the quality of this paper. This work has been supported by the Spanish Ministry of Ciencia y Tecnología and the European Union (Feder Funds) under grant TIC2003-08154-C06-03. Ricardo Fernández-Pascual has been supported by the fellowship 01090/FPI/04 from the Comunidad Autónoma de la Región de Murcia (Fundación Séneca, Agencia Regional de Ciencia y Tecnología).

6. REFERENCES

[1] M. E. Acacio and J. M. García. Techniques for improving the performance and scalability of directory-based shared-memory multiprocessors: A

survey. *Journal of Computer Science & Technology*, 3(2):1–8, October 2003. Invited paper.

[2] M. E. Acacio, J. González, J. M. García, and J. Duato. An Architecture for High-Performance Scalable Shared-Memory Multiprocessors Exploiting On-chip Integration. *IEEE Transactions on Parallel and Distributed Systems*, 15(8):755–768, August 2004.

[3] L. Barroso, K. Garachorloo, and E. Bugnion. Memory system characterization of comercial workloads. In *Proceedings of the 25th International Symposium on Computer Architecture (ISCA'98)*, pages 3–14, June 1998.

[4] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers, Inc., 2002.

[5] R. Fernández and J. M. García. RSIM x86: A cost-effective performance simulator. In *19th European Conference on Modelling and Simulation*, pages 774–779, Riga, Latvia, June 2005. European Council for Modelling and Simulation.

[6] C. Hughes, V. Pai, P. Ranganathan, and S. Adve. RSIM: Simulating shared-memory multiprocessors with ILP processors. *IEEE Computer*, 35(2):40–49, February 2002.

[7] M. M. Martin. *Token Coherence*. PhD thesis, University of Wisconsin-Madison, December 2003.

[8] M. M. Martin, M. D. Hill, and D. A. Wood. Token coherence: A new framework for shared-memory multiprocessors. *IEEE Micro*, 23(6):108–116, November/December 2003.

[9] M. M. Martin, M. D. Hill, and D. A. Wood. Token coherence: Decoupling performance and correctness. In *The 30th Annual International Symposium on Computer Architecture*, pages 182–193, June 2003.

[10] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, 33(4):92–99, September 2005.

[11] P. Stenström, M. Brorsson, and L. Sandberg. An adaptive cache coherence protocol optimized for migratory sharing. In *20th ACM/IEEE Annual International Symposium on Computer Architecture*, pages 109–118, May 1993.