A Comparison of SCI and Typhoon

Emmanuel Ackaouy Eric Rotenberg

May 8, 1995

Abstract

The debate over the roles of software and hardware in providing a cache coherent shared memory programming model is still open. In this paper we compare two coherence philosphies which lie at opposite ends of the spectrum: strictly hardware-based approaches and user-level protocols. The implementations chosen for this comparison are the SCI coherence protocol and Typhoon.

Our preliminary results show that Typhoon running the Stache user-level protocol performs comparably to a base SCI system.

1 Introduction

Perhaps the single most important problem in multiprocessing is the communication of data between processors. This problem is traditionally cast into the "shared memory versus message passing" debate. However, this black-and-white distinction of the problem has proven fruitless, and consequently it has evolved into a new debate – that of "communication policies versus mechanisms".

In this paper we begin a comparative study of two approaches which exemplify the debate of policies and mechanisms. The Scalable Coherent Interface (SCI) coherence protocol [12] is a hardware based policy for communicatiing in large-scale multiprocessors. Typhoon [1] is a proposed high-end (hardware-assisted) implementation of the Tempest Interface [3], a portable parallel programming interface which provides a diverse set of mechanisms for communicating in large-scale multiprocessors. The sections which follow briefly summarize the two approaches.

1.1 SCI

SCI is a distributed directory-based hardware coherency protocol. The objective of SCI is to provide transparent coherent shared memory that scales well both in terms of hardware cost and parallel performance. This objective led to the following key features of the SCI coherency protocol:

- directory-based coherency
- distributed directories
- directories maintain doubly linked-list of nodes caching a block
- hardware queue-based locks, or QOLB [15]

The first two features, directory-based coherency and distributed directories, is a first step in scaling systems from the traditional 4-way bus-based multiprocessors to large-scale distributed shared memory machines [7]. Using a directory protocol instead of a snoopy protocol allows for point-to-point messages between only those processors that need to be involved in a coherency transaction. Distributing these directories with the memory is a natural extension to provide scalable bandwidth.

The last two features, doubly linked-list of sharers and hardware queue locks, are unique to SCI. In other directory protocols such as Dash [8], the state of a given block is maintained in only one physical directory. Since any processor in the system can have the block cached, a directory entry must use either a bit vector as long as the number of nodes in the system, or a small number of node pointers combined with a limited broadcast mechanism. In one case, the hardware overhead required to maintain the state of a block does not scale linearly with the number of processors, and in the other there is the possible performance impact due to broadcast messages. SCI provides an alternative to these problems by distributing the caching state of a block among all of the caches that have that block. This is done by maintaining linked-lists of processors which share a block (each cache tag entry has two pointers, one to each neighbor in the sharing list). Using this linked-list data structure has two benefits: the hardware directory overhead is a fixed percentage of the total shared memory (cost scales well), and it provides the base hardware on top of which QOLB can be implemented.

Queue-based locking is an efficient mechanism for implementing spin-locks in multiprocessors. QOLB maintains the queue in hardware and consequently does not incur the overhead of the all-software MCS algorithm [16]. QOLB also has the distinct advantage of being able to pass a data structure along with the lock when it is relinquished.

1.2 Typhoon

The Typhoon architecture approaches the problem of cache coherent shared memory in a novel way, drawing from both software and hardware paradigms. The designers of Typhoon worked from the following premises [1]:

- There is no single coherence protocol that works optimally over a broad range of applications. Therefore, an architecture should provide mechanisms instead of policies (protocols) to the user for handling communication. This leads to the concepts of *user-level shared memory* and the Tempest Interface [3].
- Certain communication – namely, fine-grained sharing in the form of coherence messages – requires the ability to communicate at the cache block level. This leads to the concept of hardware-assisted *fine-grained access control.*
- Since it is sometimes possible to overlap communication with computation, it might be a good idea to free the CPU from protocol processing.

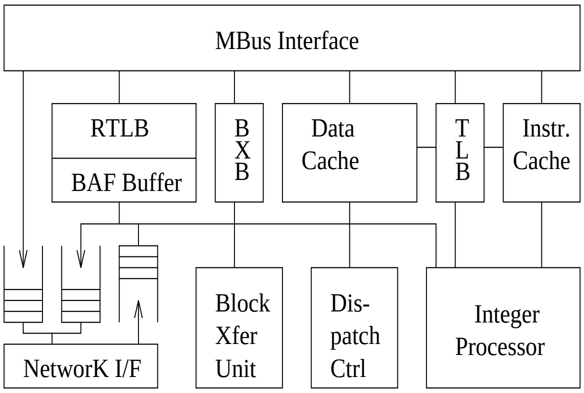

Figure 1: Block diagram of the Typhoon network processor.

The high-level view of the Typhoon hardware is shown in Figure 1. The generality required by the first premise – user-level shared memory – is implemented by incorporating a processor core in the network interface chip. This processor executes software handlers developed by the user. Note that the "user" can mean the programmer, the compiler (in theory), or libraries linked in with the application. Having a separate processor in the network interface also satisfies the third premise, allowing the central CPU to do useful computation in parallel with coherence processing. The second premise – fine-grained access control – requires the ability to recognize the state of a cache block at the time of a cache miss. This is achieved through custom hardware in the network interface. The RTLB (reverse translation lookaside buffer) snoops the address which is put out by the central processor on the memory bus, and examines the state of that block. Depending on the state, the RTLB may let the processor proceed with the memory access, or possibly stall the request and invoke a user-level access fault handler.

Typhoon builds upon previous software coherence schemes in several ways. First, unlike the first virtual shared memory architectures such as Munin [10] and IVY [9], coherence can be maintained at finer granularities than a page. This requires special hardware support in the form of the RTLB, but the benefits are three-fold: faster protocol processing, independence from the operating system (state not maintained in PTEs), and a significant reduction in false sharing. Second, Typhoon generalizes the concept of choosing protocols as first put forth by the Munin project. Munin implements a base set of protocols such as delayed-update, invalidate, and update-with-timeout, and allows the programmer to annotate variables with the desired protocol. Tempest and Typhoon go one step further by allowing the user to implement any handlers.

1.3 Qualitative Comparison

Typhoon addresses the fact that while a fixed coherency protocol implemented in hardware is generally much faster than software based coherency, the hardware protocol cannot possibly work optimally over a broad range of applications. For example, certain classes of applications may perform significantly better with a delayed update protocol than with an invalidate protocol, as demonstrated with the *em3d* application [2]. Supporting both protocols in hardware is virtually impossible – design complexity and chip realestate does not allow for it.

The basic performance tradeoff when comparing SCI and Typhoon is between flexibility of the coherence mechanism and the time required to execute protocol transactions. A flexible architecture can run different protocols for different classes of applications. If the data sharing pattern of an application is well understood, it may be possible to design a custom protocol which minimizes the communication overhead (coherency traffic) required to execute the program correctly. Alternatively, there may be portions of the program which do not contribute much to overall execution time, and the programmer can choose not to tune this code. However, this generality incurs a cost – a software handler must be invoked, instructions fetched and executed, *handler* cache misses resolved, etc. – overhead which is absent in the hardware protocols. The main idea behind user-level protocols is to tolerate longer access fault processing, but reduce the frequency of coherency transactions and perhaps the cost of hardware.

The basic cost tradeoff when comparing SCI and Typhoon is between custom protocol development cost and hardware cost. The principle problem with the "mechanisms" (Typhoon) approach is that the development cost of user-level protocols may outweigh any performance advantage they have over a fixed hardware policy. On the other hand, SCI hardware costs in terms of design time (incuding hardware verification) and silicon cost may be prohibitive in large-scale multiprocessors. (Note that Typhoon also requires some special hardware support – in the network interface.)

2 Methodology

In this section, we outline the simulation platform used, simulation parameters, the experiments performed, and the primary measure of performance.

2.1 Simulators

The common platform used for simulating both architectures is the Wisconsin Wind Tunnel [14]. Existing parallel event-driven simulators developed by the SCI and Typhoon groups at Wisconsin were used to run experiments. Using a common underlying simulation platform allowed us to abstract technology and architectural factors which are inconsequential to cache coherence. The WWT is also an attractive simulation platform because it approximates cycle-by-cycle simulation for the duration of an actual benchmark run.

Only the Typhoon simulator required modifications. This included changing the block size from 32 bytes to 64 bytes, fixing bugs, and changing timing parameters to more closely match those of SCI. Also, a non-trivial amount of code was added to support the latest Tempest interface, but this effort was set aside due to complications. Support for varying the Stache block size was also added.

2.2 Simulation Parameters

Hardware configuration and timing parameters are shown in Figure 2.

In order to precisely compare the execution time of programs run on either simulator, the timing mechanisms have to match as closely as possible. Timing can be divided into two modes:

Number of Nodes	32
Processors/Node	1
Cache Block Size	64
Page Size	4096
CPU Cache	256K/1M, 4-way assoc.
NI D-Cache	16K, 4-way assoc.
NI I-Cache	8K, direct mapped
CPU TLB	64, fully assoc., FIFO repl.
NI TLB	64, fully assoc., FIFO repl.
RTLB	64, fully assoc., FIFO repl.
Castout Buffer	"ideal"
Memory Bus	64 bit, 2x clock
Network	Arbitrary

CPU virtual time	1 cycle/instr
NI virtual time	1 cycle/instr
Cache Access (L2)	3
Memory Directory Access	10
Memory Data Access	10
CPU Cache Fill	16
NI Cache Fill	16
CPU TLB Miss Penalty	25
NI TLB Miss Penalty	25
RTLB Miss Penalty	25
NI Initiate (RTLB + handler dispatch)	2
Fixed Network Latency	91
Hardware Barrier Latency	91

Figure 2: Hardware and Timing parameters used in simulations.

- While a target application is hitting in the target cache, each instruction is counted as 1 cycle. The target application code is instrumented in order to count instructions. Note that in the Typhoon simulator, handler code executing on the target network processor is counted in exactly the same way as code executing on the target CPU. Further, the CPU and NI execution times may overlap, and are resolved only when they synchronize.
- If the target application misses in the cache, the simulators add on fixed hardware costs based on current technology assumptions. These costs are listed in Figure 2.

Some reasonable assumptions were made which merit some explanation:

- The processor-DRAM gap is assumed to be on the order of 10 cycles, that is, DRAM access time is 10 processor cycles; DRAM *cycle time* is not modelled.
- In SCI, the data and directory lookup are assumed to proceed in parallel at the memory controller.
- The memory bus is 64 bits, clocked at $\frac{1}{2}$ the processor speed; the memory bus latency shows up in the "fill cost" terms.
- Cache hits are assumed to be on average 1 cycle, since an on-chip L1 cache is assumed; a cache miss must always go through the larger L2 cache, which has a cost of 3 cycles.

2.3 Experiments

2.3.1 Benchmarks

We chose 4 benchmarks from the SPLASH [13] benchmark suite: *barnes-hut*, *mp3d*, *ocean*, and *water*. Also, *em3d* with an application-specific protocol is included in the experiment. All benchmarks were compiled using the gcc library version 2.6.2 of sparc-sun-sunos4.1.3, optimization level -O3. We used the default SPLASH input datasets.

The benchmarks for SCI are tuned for SCI-specific features (e.g. QOLB), and likewise for Typhoon. Padding of data structures is used where available.

2.3.2 Implementation Design Space

Both the SCI and Typhoon architectures provide a rich design space for study. However, the scope of this project is limited by time constraints as well as numerous simulator problems. Therefore, we chose to look at extreme implementations of either architecture as a starting point for comparison – these two endpoints are called BASE and SUP(er).

The systems which we studied are as follows:

- BASE_SCI: "typical set" coherence (fresh and dirty lists), sequential consistency, MCS locking
- SUP_SCI: "typical set" coherence (fresh and dirty lists), sequential consistency, pairwise sharing, QOLB
- BASE_TYPHOON: Typhoon running the Stache protocol [1]

- SUP_TYPHOON: Typhoon running an application-specific protocol

We also chose to run with 2 realistic L2 cache sizes, 256K and 1M, to determine the effect of capacity misses in either implementation. Note that sequential consistency is used throughout the simulations – although SCI can relax the ordering constraints on write operations, we do not present numbers for this feature because writes in Typhoon are strictly blocking.

2.4 Performance Metric

The primary measure of performance is *virtual time*, the number of simulated clock cycles required to execute an application.

3 Results

In this section we present the results of our simulations of the 4 system configurations, BASE_SCI, BASE_TYPHOON, SUP_SCI, and SUP_TYPHOON.

3.1 256K vs 1M Level-2 Cache

One of the obvious advantages that Stache has over the SCI system is that capacity and conflict misses on a node are essentially eliminated. This is due to the fact that local memory serves as a level-3 cache for remote data. Consequently, only coherency invalidations can remove a block from a node.

It is important to isolate this effect from other factors when comparing the SCI and Typhoon protocols, for the sake of useful protocol comparisons. Therefore simulations were performed on all configurations with both a 256K and 1M CPU cache. The speedup achieved by using a 1M cache over a 256K cache is less than 3% – from this we conclude that the application datasets fit in the SCI cache, which in turn factors out the Stache effect.

Benchmark	BASE_SCI	BASE_TYPHOON	$\frac{BASE\_SCI}{BASE\_TYPHOON}$
barnes	76,795,770	67,100,124	1.145
mp3d	187,702,521	154,552,126	1.214
ocean	10,368,386	14,867,166	0.697
water	207,243,600	213,207,287	0.972

Table 1: Cycles executed on BASE_SCI and BASE_TYPHOON using a 1M cache. The last column is the ratio of SCI cycles to Typhoon cycles.

3.2 BASE_SCI vs BASE_TYPHOON

The data in Table 1 show that BASE_SCI performs better on the *water* and *ocean* benchmarks, yet BASE_TYPHOON performs better on *mp3d* and *barnes*.

The latter result is somewhat surprising. Since Stache and SCI are both fixed invalidate protocols, one might expect BASE_SCI to always outperform the base software counterpart. However, a brief analysis of the two protocols shows that indeed the software protocol can perform better in the base case.

The following observations help explain the data for *mp3d* and *barnes*:

1. Using a microbenchmark, we measured the write miss penalty for remote data with various numbers of read sharers; refer to Table 2. Notice that SCI hardware services misses only 40 cycles faster than Typhoon's handlers. Most of the time in either case is spent in the network. Further, as the sharing list in SCI grows, the penalty grows linearly after 1 processor, because of the sequential invalidation of readers. On the other hand, the Stache protocol miss penalty hardly grows at all beyond 1 processor due to the broadcast of invalidates.

2. The common case of local miss is serviced as quickly in Typhoon as in SCI (about 30 cycles), since the network protocol processor does not intervene in local accesses.

3. The results may actually be biased in favor of Stache, since the network bandwidth is not a constraint. In reality, broadcast invalidates would experience slower delivery due to queueing.

4. The benchmarks *ocean* and *water* exhibit greater locality than *mp3d* and *barnes* [13]. One would expect longer sharing-lists in the latter two applications as a result; longer sharing-lists take longer to break down.

# of read sharers	BASE_SCI (cycles)	BASE_TYPHOON (cycles)
none	245	286
1	641	512
2	839	540

Table 2: Round-trip remote cache miss penalty for BASE_SCI and BASE_TYPHOON.

Benchmark	cycles	speedup
barnes	53,554,744	1.434
mp3d	94,238,096	1.992
ocean	6,434,966	1.611
water	188,140,590	1.102

Table 3: Cycles executed on SUP_SCI (and speedup over BASE_SCI).

Benchmark	BASE_TYPHOON	SUP_TYPHOON	speedup
em3d	128,282,234	119,802,487	1.071

Table 4: Cycles executed on BASE_TYPHOON and SUP_TYPHOON for em3d

3.3 SUP_SCI vs SUP_TYPHOON

Unfortunately we were unable to make any direct comparisons between the two high-end configurations of SCI and Typhoon. First, there is only one application-specific protocol that runs on the WWT Typhoon simulator, *em3d*. Second, *em3d* has not yet been optimized for SUP_SCI (i.e. no QOLB or pairwise sharing directives). Third, the ported version of *em3d* on BASE_SCI is flaky.

Instead, we look at the performance gain for SUP_SCI and SUP_TYPHOON separately, and then compare these results with recent SCI [5] and Typhoon [1] publications. Our experiments with the four SCI applications (Table 3) show speedups ranging from 1.1 to 2.0, somewhat greater than the speedups from [5], 1.04 to 1.58 . The descrepancy is probably due to the difference in timing target program execution (we use 1 cycle/instruction). At any rate, QOLB is obviously very effective at reducing runtime. Our single experiment with the application-specific *em3d* protocol results in a speedup of 1.07 (Table 4) – the speedup increases with the problem size [1].

Benchmark	BASE_SCI	SCI Relaxed	speedup
barnes	76,795,770	65,025,147	1.18

Table 5: Cycles executed on BASE_SCI and SCI with fully relaxed consistency.

3.4 SCI Weak Consistency

While weak consistency models are not strictly part of the cache coherency protocol, it is interesting to see how relaxing consistency in SCI can hide the effects of its expensive write transactions [5]. Table 5 shows that the extreme case where writes never stall the processsor (as long as the write buffer is available) yields significant speedup for the benchmark *barnes*. In this case all transactions involved in a write miss – accessing the memory directory, prepending at the head of the list, and issuing invalidate requests – are performed asynchronously and out of order with respect to *other* writes. (This is not true for special synchronizing, or signalling, writes.)

As stated previously, the issue of consistency is not directly related to the coherency protocol of either architecture. The CPU in Typhoon could also implement an aggressive write buffer which supports completely nonblocking writes.

4 Conclusions

Using the Wisconsin Wind Tunnel simulation environment, we were able to study and compare the performance of SCI and Typhoon, two different philosophies for solving the cache coherence problem in shared memory multiprocessors.

Our main result is that the base configuration of SCI and Typhoon running the Stache protocol provide comparable performance over four SPLASH benchmarks. This is a surprising result considering that both use invalidate protocols, but SCI is implemented in hardware and Stache in user-level software. Stache performs better than base SCI in the case of frequent writes to relatively long sharing lists, because SCI performs serial invalidation of the list. However, the results may be skewed somewhat in favor of Stache since its broadcast invalidates see no contention in the network.

We would like to see this work continued in possibly two directions:

- A study comparing aggressive implementations of SCI against fine-tuned application-specific protocols on Typhoon is in order.
- It would be interesting to put the user-level "mechanisms" philosophy to the real test – an implementation with no hardware support (i.e. Blizzard). Typhoon relies on a custom network processor in order to implement fine-grain access control. This results in Typhoon/Stache being as fast as SCI in the common case of local misses; this may cloud any conclusions as far as the benefit of user-level shared memory.

5 Acknowledgements

We are most grateful to Babak Falsafi, Alain Kägi, Steve Reinhardt, Doug Burger, and Stefanos Kaxiras for the large amount of time they spent helping us with this project.

References

[1] Steven K. Reinhardt, James R. Larus, and David A. Wood, "Tempest and Typhoon: User-Level Shared Memory," Proceedings of the 21st Annual International Symposium on Computer Architecture, April 1994.

[2] Babak Falsafi, Alvin R. Lebeck, Steven K. Reinhardt, Ioannis Schoinas, Mark D. Hill James R. Larus, Anne Rogers, David A. Wood, "Application-Specific Protocols for User-Level Shared Memory," Supercomputing '94, Nov. 1994.

[3] Mark D. Hill, James R. Larus, and David A. Wood, "Tempest: A Substrate for Portable Parallel Programs," COMPCON Spring 1995, March 1995.

[4] Steven K. Reinhardt, "Tempest Interface Specification," February, 1995.

[5] Alain Kagi, Douglas C. Burger, and James R. Goodman, "Techniques for Reducing Overheads of Shared-Memory Multiprocessing," International Conference on Supercomputing , July 1995.

[6] Douglas C. Burger and David A. Wood, "Accuracy vs. Performance in Parallel Simulation of Interconnection Networks," ACM/IEEE International Parallel Processing Symposium, April 1995.

[7] A. Argarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," Proceedings of the 15th Annual Symposium on Computer Architecture (June 1988), pp. 280-289.

[8] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Henessy, Mark Horowitz, and Monica S. Lam, "The Stanford Dash Multiprocessor," IEEE Computer, 25(3):63-79, March 1992.

[9] K. Li, P. Hudak, "Memory Coherence in Shared Virtual Memory Systems," ACM transactions on Computer Systems, Vol. 7, No. 11, (November 1989), pp. 321-359.

[10] J.B. Carter, J.K. Bennet, and W. Zwaenepoel, "Techniques for Reducing Consistency-Related Communication in Distributed Shared Memory Systems," to appear in ACM Transactions on Computer Systems.

[11] Steven K. Reinhardt, Mark D. Hill, James R. Larus, Alvin R. Lebeck, James C. Lewis, David A. Wood, "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers," ACM SIGMETRICS, May 1993.

[12] "IEEE Standard for Scalable Coherent Interface (SCI)", IEEE Computer Society, August, 1993.

[13] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta, "SPLASH: Stanford Parallel Applications for Shared Memory," Computer Architecture News, 20(1):5-44, March 1992.

[14] Steven K. Reinhardt, Mark D. Hill, James R. Larus, Alvin R. Lebeck, James C. Lewis, and David A. Wood, "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers," Proceedings of the 1993 ACM SIGMETRICS Conference, May 1993.

[15] J. R. Goodman, M. K. Vernon, and P. J. Woest, "Efficient synchronization primitives for large-scale cache-coherent multiprocessors," Proceedings of the 3rd Symposium on Architectural Support for Programming Languages and Operating Systems, pp. 64-75, April 1989.

[16] J. M. Mellor-Crummey, and M. L. Scott, "Synchronization Without Contention," Proceedings of the 4th Symposium on Architectural Support for Programming Languages and Operating Systems, pp. 269-278, April 1991.