Abstract

An instruction level simulator called MILL (Multiprocessor Instruction Level simuLator) is proposed for the evaluation of advanced snoop cache systems. In MILL, every instruction of a processor in the target multiprocessor is directly interpreted, and other components of the multiprocessor such as caches, shared memory and bus are also fully simulated by software. Various kinds of cache protocol can be defined just by describing the state transition of the cache line. In this paper, the mixed-protocol cache is simulated with MILL on parallel applications from SPLASH benchmark programs. Through the simulation, it appears that the performance of the Illinois/Dragon mixed protocol is from 10% to 25% better than those of simple Illinois or Dragon protocol.

1 Introduction

A bus connected multiprocessor is one of the most promising types of small scale parallel machines. A small number of processors are connected with a shared memory through a bus. In order to reduce the access latency and the bus congestion, each processor provides a private cache with a snoop mechanism.

Since the control algorithm for maintaining the coherency of the snoop cache is a key of the performance, many kinds of coherency protocols[1][2] are proposed. Recently, advanced techniques including multi-protocol cache, prefetch, injection[3], and combination with synchronization mechanisms[4] have been researched for further enhancement of the performance.

However, it is difficult to demonstrate which techniques are really efficient because the performance of the snoop cache is severely influenced by the detail of the machine (eg. cache size, line size, detail of the cache protocol and detail of the implementation) and the behavior of the applications.

Simulations using probabilistic models are not useful for such evaluations. Trace driven simulations support better results [5]. However, in this method, the target system is severely limited by the machine from which the trace data is obtained.

In the execution driven simulations[6],[7], the code of the application program is directly executed, and when the shared memory is accessed, the simulator is called. Precise evaluation results can be obtained with this method if the target processor is similar to the machine on which the simulator is executed. However, since the processor and memory system of recent workstations are highly optimized, it is sometimes differs from the processing unit in a multiprocessor.

The full instruction level simulation[8], in which every CPU instruction, cache system, bus system, and shared memory are simulated by software, supports the maximum flexibility and accuracy. This method takes huge simulation time. However, recent advances of high performance workstation enables to use this method if the size of target multiprocessor is not so large.

We have developed such an instruction level simulator called MILL (Multiprocessor Instruction Level simuLator) for the evaluation of advanced snoop cache systems. In MILL, various kinds of cache protocol can be defined just by describing the state transition of the cache line. In this paper, the simulation model of target multiprocessor and cache are described in Section 2. The implementation of MILL is introduced in Section 3. In Section 4, the evaluation of the multi-protocol cache is presented.

2 Simulation model

The target machine of MILL is a bus connected multiprocessor shown in the Figure1. In order to maximize the flexibility, each processor provides not only the snoop cache, but also a local memory system and a distributed synchronization mechanism. The local memory system is effective for reducing the bus congestion if a user or sophisticated compiler can allocate the private data onto it. The distributed synchronization mechanism which consists of broadcast memory with tags [9] efficiently supports both the synchronization based on shared variables and message passing which can cause the interrupt.

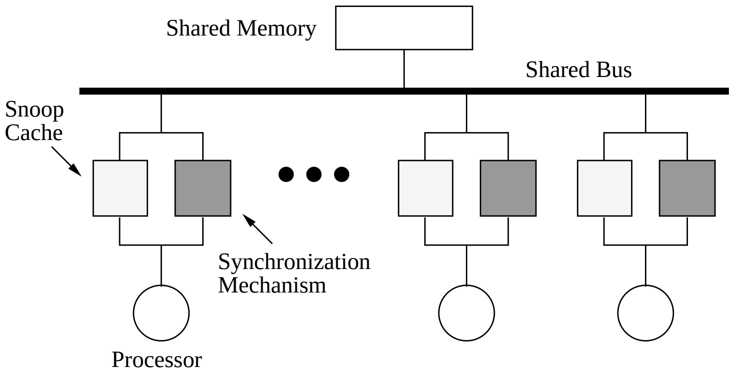

Figure 1: Target multiprocessor model

These facilities can be disabled for simulating simple multiprocessors. Simple synchronization using only shared variables on the shared memory are also available.

2.1 Specification of the snoop cache system

In multiprocessors shown in Figure1, copies of the data on the shared memory are distributed into private caches each of which is attached to a processor. In order to maintain the coherency of the copies, the cache "snoops" into transactions on the shared bus and invalidates or updates the appropriate cache line or data. This mechanism is called the cache coherency protocol and the cache with such a mechanism is called the snoop cache.

Like the cache for uniprocessors, coherency protocols of snoop caches are classified by the write policies: the write through and the write back. Another important factor is the policy to maintain the coherency between caches. In the **invalidate protocol**, when data is written into a cache line which is shared with other caches, all copies are invalidated. On the contrary, in the **update protocol**, the data is broadcast to update the copies in other caches.

Various types of cache coherency protocols have been proposed [1]. Sweazy and Smith [2] represent them with matrices of cache line states, CPU transactions and bus transactions. Although this definition, which is called the MOESI protocol class, can specify bus and cache operations exactly and systematically, it is hard to specify and understand.

In MILL, the cache protocol can be specified with a simple and practical way based on the state transition diagram. Unlike the MOESI protocol class, first, the write policy is chosen between the **write through** and **write back**. Then, the coherency policy is chosen between the **invalidate** and **update**.

If the write through policy is chosen, the management of the cache is determined here. Otherwise, the state of the cache line must be specified. Like the MOESI protocol class specification, the cache line state is represented with the following three bits:

- V/I : Valid/Invalid,
- S/E : Shared/Exclusive,
- C/D : Clean/Dirty.

Since the S/E bit and the C/D bit mean nothing if the line is invalid, the following five states are possible:

- I : Invalid
- CE : Clean Exclusive
  The line is consistent with the shared memory and is not shared with any other caches.
- CS : Clean Shared
  The line is consistent with the shared memory and shared with other caches.
- DE : Dirty Exclusive
  The line is not consistent with the shared memory (i.e. write back is necessary) and is not shared with any other caches.
- DS : Dirty Shared
  The line is not consistent with the shared memory and shared with other caches.

In MILL, a protocol is specified with the state transition between these states. Figure2 shows the diagram of Illinois protocol, which is the typical write back/invalidation type protocol. The state transition is caused by either the bus transaction or the access requests from the processor. In MILL, the states and the conditions of the state transition are specified. The state transition may cause other bus transactions and data transfer. However, these operations are mostly determined when the source/destination states and the conditions which activates the transitions are known. Therefore, in MILL, the operations caused by a state transition are decided automatically. Only the following issues must be chosen to decide the detail of the operations:

- Intervene:
  When a line is required, it is usually carried from the shared memory. However, if there is a cache which holds the copy, it can be a supplier. This operation is easily implemented with a bus transaction called the "intervene". The user must specify the intervene is used or not.

- Reflection:
  When a cache supplies a line in the DE state to the other cache, in some protocols, it is often written into the shared memory. In such a case, there are two implementations: (1) the supplier cache writes back the line into the shared memory first, then the line is transferred to the cache which requires the line. (2) the line is transferred to both the shared memory and the cache which requires the line in the same bus transaction. This operation is called the "reflection".

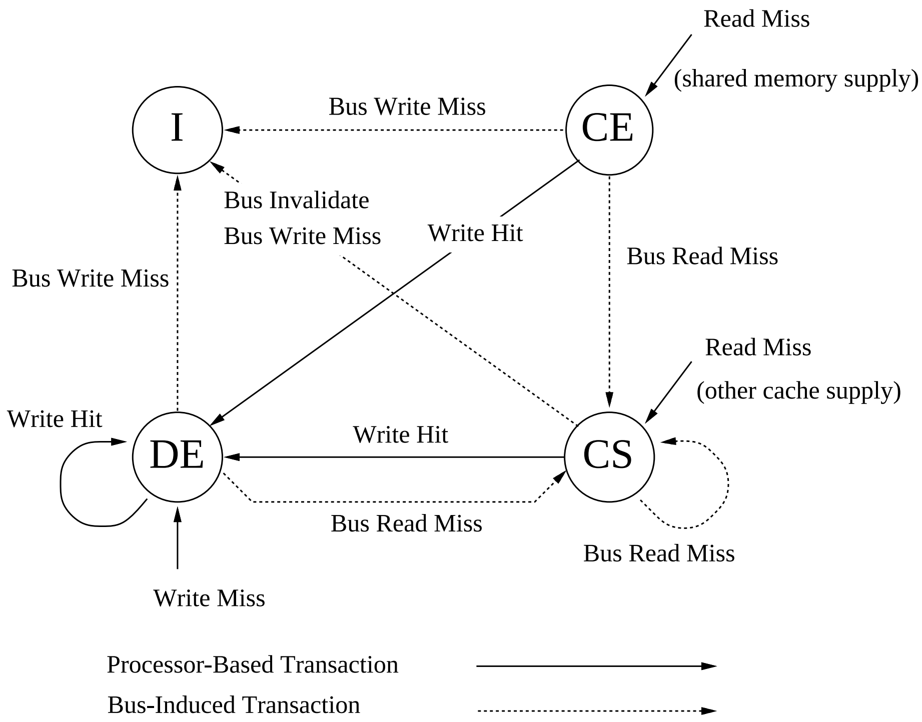

Figure 2: State transition diagram of Illinois protocol

In MILL, the tag memory for status bits are assumed to be the dual port memory, and it can be read out from the bus and the processor simultaneously and independently. The cache memory itself is single port, and the simultaneous accesses from the processor and the bus are prohibited.

Other cache issues, cache memory cycle time, cache size, and line size can be set freely. The replacement policy is fixed to be the LRU because other policies are rarely used.

2.2 Bus model

The system bus used in MILL is based on the IEEE standard Futurebus [10]. The Futurebus is designed as a system bus of multiprocessors, and provides facilities for implementing various cache protocols (eg. intervene and reflection).

The following steps are necessary to send the data through the bus:

- Bus arbitration:
  The winner is chosen randomly from competitors for keeping the fairness between processors. The "parking" mechanism is available. This mechanism gives the mastership without arbitration to the processor which just used the bus in case such a processor consequently requires the mastership with no other processors requesting the mastership.

- Bus transaction:
  In MILL, a bus transaction consists of the connection phase, the address/data transfer phase, and the disconnection phase. The consuming time for every phase can be set freely. If continuous transactions are required for implementing a cache coherency protocol, the mastership is not moved. Split transaction is not supported in the current state.

The arbitration and the bus transaction are executed in the overlapped manner.

The IEEE Futurebus is an asynchronous bus. However, since the loss caused by the handshake only influences the consuming time of each phase in MILL, it can be used as the synchronous bus. The IEEE Futurebus+[11], which is less flexible than original Futurebus from the viewpoint of the cache control, can be easily emulated except split bus transaction.

3 Simulator Implementation

3.1 Implementation overview

MILL is implemented on a SUN workstation using the light-weight processes library package.

MILL reads the simulation configuration file when it starts simulation. Figure 3 shows the overview of MILL. MILL consists of four kinds of modules: i.e. processors, caches, synchronizers, a bus and a shared memory. The shared memory module is just a passive data structure shared by other modules. Other modules are implemented as threads of a MILL process so as to enable easy sharing of data structures. The synchronizers are provided to support the architecture of ATTEMPT-0 (A Typical Testing Environment of MultiProcessor sysTem version 0) [9].

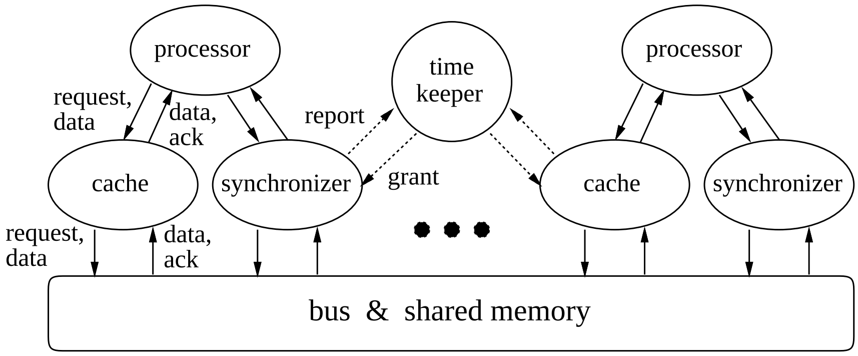

Figure 3: The organization of MILL

The default threads scheduler of the light-weight process package is used to schedule threads. If a thread send a message, the thread is suspended until a reply to the message is returned.

The time ordering of the interactions among modules are kept on the right track by a thread called timekeeper. Each module maintains its own clock. As far as the actions of a module use only the data structures local to the module, the module continues its action advancing its own clock. When a module accesses a shared data structure, it sends a registration request to the timekeeper and then suspends waiting a reply from the timekeeper. Upon receiving a request, the timekeeper register the request to its time-ordered events list, and activates the module at the top of the list by sending a reply to the corresponding thread. This time ordering management scheme enables consistent data sharing among threads.

Next, we describe the implementations of the modules.

3.2 Components of MILL

Processor

The processor module includes *CPU emulator* which interprets and executes the codes of the target programs . Currently, a CPU emulator of MC68000 family processors is provided. Altering the CPU emulator, machines made with any processor can be simulated.

In our model, each processor has some amount of local memory on which texts, data or stack of the application programs can be located. This local memory is implemented as a local data structure of a processor module thread. Therefore no interactions with other modules occur during local memory access. When a processor accesses cache memory or synchronizer, a message is sent to the cache or synchronizer module.

Cache

Cache coherency protocols can be divided into some primitive operations. Combining these primitives, MILL can support essentially any protocol. Users can specify their desired protocol in the simulation configuration file. For convenience, some well known protocols such as Illinois or Dragon can be specified by their names. Different protocols can be used for different memory locations in order to evaluate effects of the multiple cache protocols.

Data exchanges between a cache and a processor is implemented using messages. When a cache module receives a request from the processor, it first checks the conflict between the access from the processor and the bus (i.e. other caches). If there is a access from the bus, it suspends the request from the processor.

If a bus transaction should be issued to support the request from the processor, the cache asks the bus module for a right to use a bus, and after granted, issues an bus transaction. Since the cache directories or the cache memories of all cache modules and the shared memory are implemented as shared data among the threads, a cache which starts bus transaction can modify them directly.

Bus

The main role of the bus module is arbitration. When a cache or synchronizer module wants to issue a bus transaction, it first asks the bus module for a right to use the bus. The bus module arbitrates such requests according to the algorithm defined by the Futurebus standard. Then it grants the use of the bus to a module by replying the message from the module. The module which get the right then issues a bus transaction simply accessing data structures such as the shared memory or other caches' directory or memory. When the bus transaction is finished, the module returns the right to the bus module by sending a message.

3.3 Gathering statistics of MILL

Following statistics can be gathered by MILL:

Each processor's cache:

- number of hits, coherence misses (misses due to invalidations), capacity misses (due to short of cache size), cold misses (due to yet unloaded data at the initial state), replaces, writeback operations and access conflicts occurred by the bus snooping

Each processor's synchronizer:

- number of read, write and Fetch&Decrement[9] accesses

Bus:

- number of bus requests and arbitrations, time consumed in waiting for the right to use the bus, bus occupation time, parking ratio of each processor(cache)
- Total number of arbitrations

Precise log of the behaviors of the processors, caches, synchronizers, bus and timekeeper are recorded if specified. Since the processor modules provide some debugging functions such as break points and step execution, MILL can be used for debugging of target programs.

4 Evaluation

4.1 The mixed protocol cache

The efficiency of a cache coherency protocol varies with the way to access cached variables. If variables are accessed continuously by a processor, the invalidation protocol is suitable. On the contrary, the update protocol can improve the performance if cached variables are frequently updated by different processors. In order to maximize the performance, a protocol should be selected according to characteristics of the cached variables. In the mixed-protocol cache, different protocols can be assigned to the separate address spaces.

The simple mixed-protocol cache consists of two representative protocols: invalidation (Illinois) and update (Dragon). Variables used for the synchronization are mapped to the memory space with the update protocol, the Dragon. The other data is mapped to the separate memory space with the write invalidate protocol, the Illinois. For comparison, we also simulate the cases when either the Illinois or the Dragon protocol is used. We refer these three protocols as **mixed**, **all-Illinois** and **all-Dragon**. The parameters of the cache are shown in the following table.

access	time	trans. size
Read hit	119ns	4byte
Read miss(Illinois)	694ns	16 byte
Read miss(Dragon)	694ns	16 byte
Read miss with writeback	1133ns	32 byte
Write hit to exclusive	119ns	4byte
Write hit to shared (Illinois)	450ns	4byte
Write hit to shared (Dragon)	511ns	4byte
Write miss (shared memory supply)	694ns	16byte
Write miss with writeback(shared memory supply)	1133ns	32byte
Write miss to shared(Illinois)	694ns	16byte
Write miss to shared with writeback(Illinois)	1133ns	32 byte
Write miss to shared(Dragon)	936ns	16byte
Write miss to shared with writeback(Dragon)	1375ns	32 byte

Table 1: Simulation parameters (line size = 16byte)

4.2 Protocol description

As described before, cache protocols can be described with state transition diagrams. Here, we show the description of the Illinois protocol used in the mixed-protocol.

```
# write policy
CACHE_WRITE_POLICY = WRITEBACK
# policy
COHERENCE_ENFORCE_POLICY = INVALIDATE
# intervene
INTERVENE = YES
# reflection
REFLECTION = NO
# possible cache line states
STATES = CE CS DE I
# state transition
#   ~I means all states except for I,
#    * means all defined states
~I, DE: WRITE_HIT
 I, DE: WRITE_MISS
 I, CE: READ_MISS && NO_HOLDER
 I, CS: READ_MISS && HOLDER_EXIST
~I, CS: BUS_READ_MISS
 *, I : BUS_INVALIDATE || BUS_WRITE_MISS
```

In this example, the protocol shown in Figure 2 is described. In this protocol, a cache (which is selected by a certain algorithm) supplies the line which is required by another cache. Therefore, the intervene is set to "YES". On the other hand, the reflection is set to "NO" because there are no DS state in this protocol. While this method gives a way to specify any protocol, some well known protocols can be specified by their names. Currently, Illinois, Berkeley, Symmetry's, Dragon, and Firefly are supported. Thus, the mixed cache protocol used here can be specified as follows:

```
# cache coherency protocol
C_PROTOCOL = illinois
CACHE_SYNC_PROTOCOL = dragon
```

4.3 Application programs

The evaluation results are much influenced with the application programs. The following three parallel applications are used.

- Eratosthenes' sieve

  Eratosthenes' sieve for finding prime numbers. In this algorithm, initially an array of numbers up to 10000 is prepared in the shared memory, then the first processor eliminates numbers which are multiple of two, the second eliminates ones which are multiple of three and so on. In order to avoid the unbalanced load, processors finished eliminating multiples of one number are dynamically assigned the next number.

- MP3D

  MP3D is one of the SPLASH benchmark[14] developed at Stanford University. This program simulates the hypersonic rarefied fluid flow using Monte Carlo method. The simulating space is a rectangular tunnel with opening at each end and reflecting walls on the remaining sides. The object under study is represented as a set of boundaries in the tunnel. Particles enter the tunnel at one end, collide with other particles or boundaries, and exit from another end. We execute the code compiled without LOCKING option for 100 steps with 400 particles at `test.geom`.

- Cholesky

  Cholesky is also from the SPLASH. This program calculate Cholesky factorization of a sparse positive definite matrix. Ordinary, Cholesky factorization has three steps of execution: that is, ordering, symbolic factorization and numerical factorization. In this program, the last step, which most consumes time, is parallelized. Supernodal elimination techniques are used for numerical factorization. We chose `lshp.O` as an example.

The SPLASH programs are written using ANL macros[15] to hide machine dependent parts of programs such as lock operations and fork calls. Users can use the SPLASH programs just writing a macros for their machine. MP3D and Cholesky are using the macros developed for ATTEMPT-0, because the current version of MILL can directly execute the codes generated for ATTEMPT-0 .

SPLASH programs begin parallel execution through multiple calls of `fork()`. In ATTEMPT-0, `fork()` is achieved by passing pointers to the functions to processors through the synchronizer. Worker processors are busy waiting until master calls `fork()`s. Variables used for synchronizations (barriers) are assigned to the memory space of Dragon protocol. Lock and unlock are implemented using Fetch&Decrement operations of the synchronizers. Thus, these operations do not affect the cache.

4.4 Simulation results

Speed-up ratio and analysis

Figure 4 shows the speed up ratio of Eratosthenes' sieve (*eratos*), MP3D (*mp3d*) and Cholesky (*cholesky*) when the number of processors increases. For *eratos*, another version of implementation which placed all data including text, private data and stacks on shared memory is also shown. The execution speed of *eratos* saturated or degraded if the number of the processors are beyond 8. For the all programs, mixed protocol achieves the best performance. The all-Illinois sometimes yield sometimes better and sometimes worse performance compared with the all-Dragon depending on the target program.

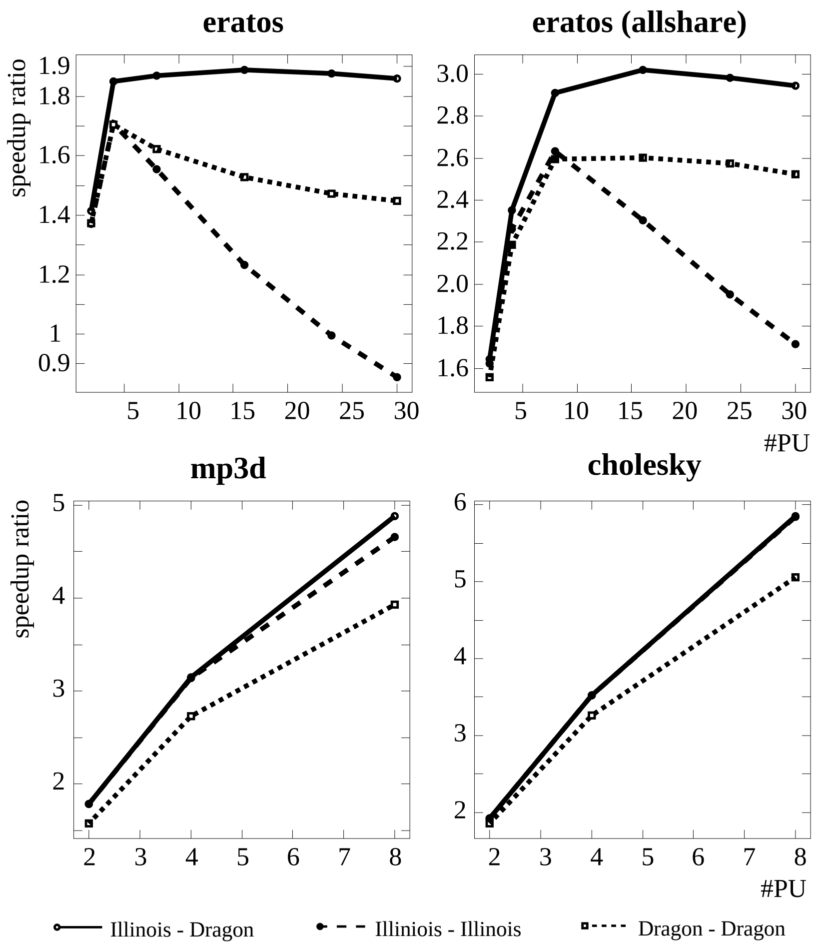

Figure 4: The speedup of 3 applications

In order to analyze the bus congestion, the bus utilization ratio (Figure 5) and the cache hit ratio (Figure 6) are measured in 8 processors (4 way set associative, cache size = 128Kbyte, line size = 16byte). Comparison of the bus utilization ratio for the mixed and all-Illinois in *eratos* shows that in the all-Illinois cache, line transfers due to invalidations of synchronization variables account for a large part of bus utilization. Therefore, in this case, assignment of the synchronization variables to the Dragon protocol memory space in the mixed works effectively. Since the program size of *eratos* is small, all shared data can be stored in the cache. Therefore, the capacity miss never occurs.

Theoretically, in the all-Illinois, coherent misses causes cache line transfers. On the contrary, all misses are cold misses in the all-Dragon and thus the amount of line transfer should be smaller. However, the results of the simulation show that the bus utilization ratio of the all-Dragon is larger than that of the mixed in which the Illinois protocol is used except for the synchronization variables. It shows that, except for the synchronization variables, the penalty caused by the update broadcasts in the Dragon protocol exceeds that of line transfers for the replaces in the Illinois.

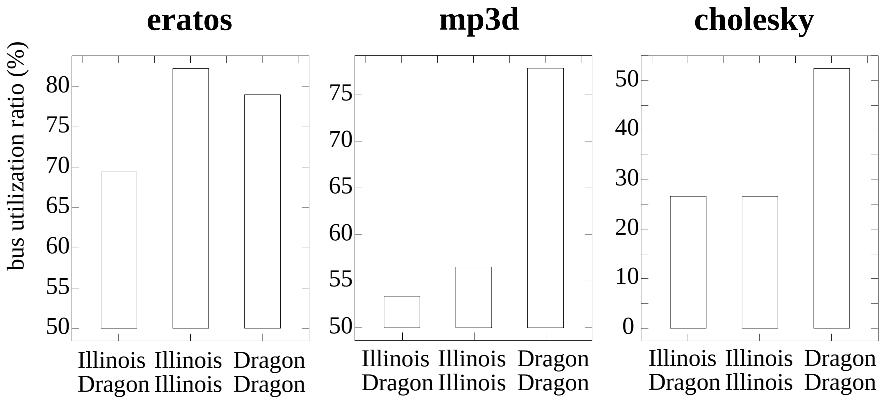

Figure 5: The bus utilization ratio of 3 applications

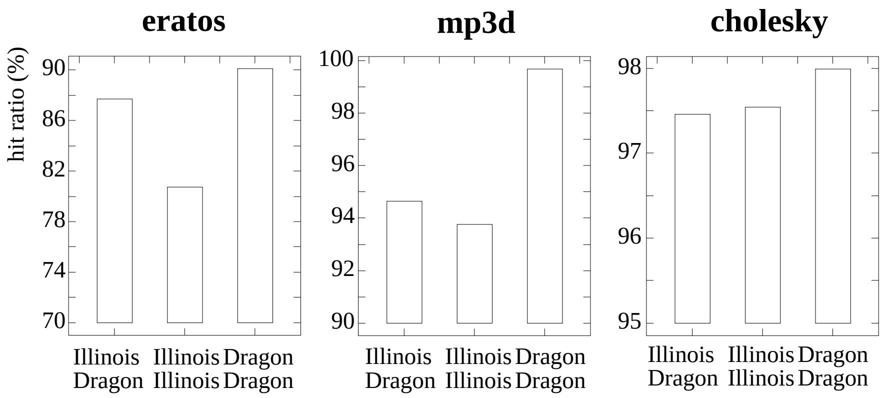

Figure 6: The cache hit ratio of 3 applications

As mentioned before, the performance of the all-Dragon is worse than others. This is an effects of the large bus utilization ratio of the all-Dragon. Especially, in *mp3d*, the amount of write access is large (read:write is about 3:1), and the waiting time for bus arbitration is almost twice as large as that of others (Figure 7). On the other hand, many read capacity misses occur in the *cholesky*. Especially, in the all-Dragon, replaces occur three times as many as that of others. Furthermore, about 70% of replaces require write back to the shared memory. This also causes three times much arbitrations and thus waiting time is three times larger than that of others (Figure 8).

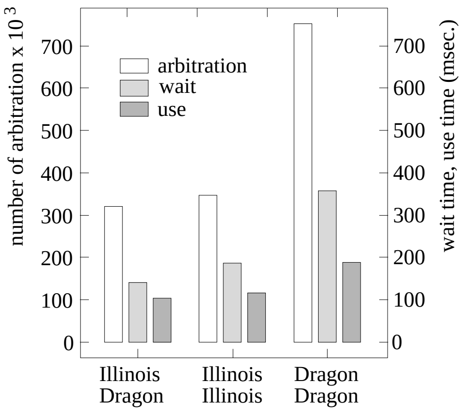

Figure 7: The number of arbitration, bus wait time, bus use time in MP3D

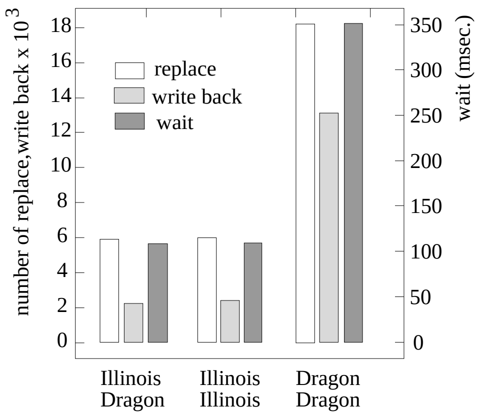

Figure 8: The number of replace, write back and bus wait time in Cholesky

Influences of the line size

Figure 9 shows the execution time, the bus utilization ratio and the cache hit ratio when the cache line size varies from 8 byte to 128 byte. The bus utilization ratio of the mixed and the all-Dragon cross each other at 32 byte. The all-Dragon shows almost constant bus utilization while the ratio is increased linearly in the mixed. This comes from the fact that in the mixed, the penalty of the line transfer increases along with the line size increases, while in the all-Dragon, the penalty of the write broadcast operation is constant. The bus utilization ratio is directly reflected in the execution time.

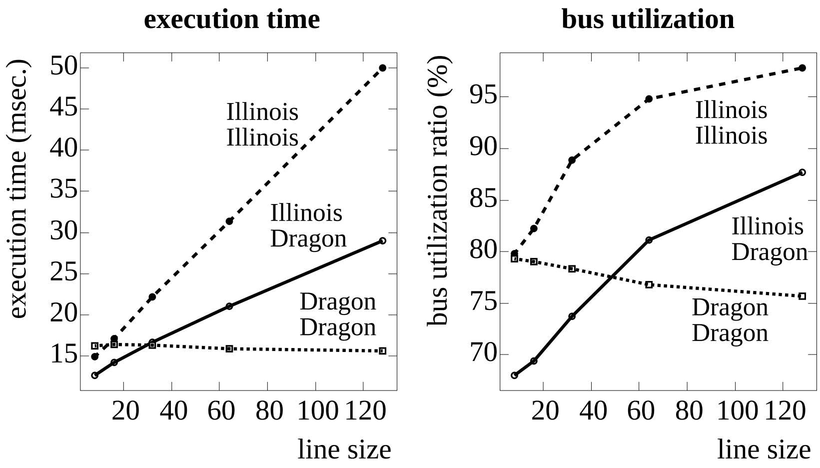

Figure 9: Line size vs. execution time, bus utilization ratio in Eratos

Influence of the cache size

In Figure 10, the results of *eratos* when cache size varies are shown. Here, (cache size)/(line size) are set to 16KB/16byte and 16KB/128byte.

When the cache size is 16KB, capacity misses occur. The all-Dragon must write back the being swapped out lines to the shared memory, since the contents of the lines are frequently valid and different from the shared memory (dirty). Figure 11 shows the ratio of the write back occurred by replaces at 16Kbyte cache size. It shows that almost 100% of replaces require write back. On the other hand, the all-Illinois has advantage of less frequent write backs invalid lines in the cache need not be written back. This is one of the reasons that the performance of the all-Dragon is poor.

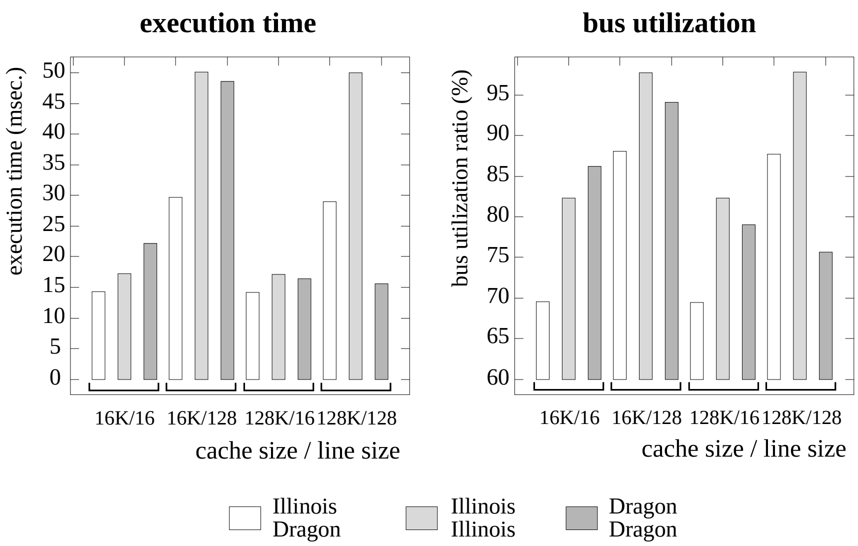

Figure 10: (cache size)/(line size) vs. exec. time, bus util. ratio in Eratos

For both cache sizes, the mixed shows the lowest bus utilization ratio and consequently the execution time is the shortest. This is the advantage to combine the two types of protocols. By combining the two, for synchronization variable, both read and write accesses mostly hit the cache, while less write backs are required for other memory spaces. These results demonstrate that if the size of problem is large enough to require replaces, the combination of the write-invalidate and the write-broadcast type protocols has certain advantages.

5 Conclusion

An instruction level simulator MILL is proposed for the evaluation of advanced snoop cache systems. Using MILL, the mixed-protocol cache is simulated, and its efficiency is demonstrated. It appears that the performance of the mixed-protocol is 10% to 20% better than those of simple Illinois or Dragon protocol.

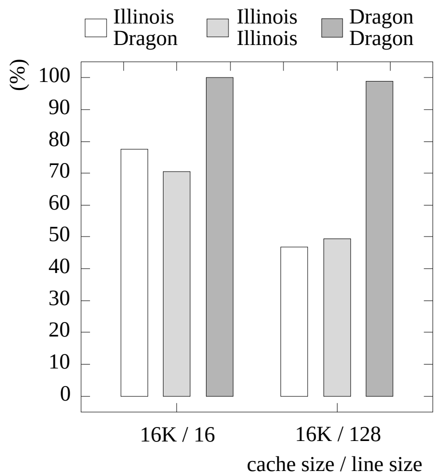

Figure 11: (average write back)/(average replace) in Eratos

The major problem of the current version of MILL is its simulation speed. It takes hours for simulating a practical application program on the target machine with more than 8 processors. In the current MILL, every part of the target multiprocessor is implemented with the thread of light-weight process for keeping the modularity. However, some part should be rewritten with function calls in order to improve the simulation speed.

The processor emulator for other family of processors (MIPS) are now under development. The NUMA multiprocessor with clusters each of which is bus connected multiprocessor is the next target machine of MILL.

6 Conclusion

An instruction level simulator MILL is proposed for the evaluation of advanced snoop cache systems. Using MILL, the mixed-protocol cache is simulated, and its efficiency is demonstrated. It appears that the performance of the mixed-protocol is 10% to 20% better than those of simple Illinois or Dragon protocol.

The major problem of the current version of MILL is its simulation speed. It takes hours for simulating a practical application program on the target machine with more than 8 processors. In the current MILL, every part of the target multiprocessor is implemented with the thread of light-weight process for keeping the modularity. However, some part should be rewritten with function calls in order to improve the simulation speed.

The processor emulator for other family of processors (MIPS) are now under development. The NUMA multiprocessor with clusters each of which is bus connected multiprocessor is the next target machine of MILL.

References

[1] J.Archibald, J.-L.Baer, "Cache-Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," ACM Trans. on Computer Systems, Vol.4, No.4, pp. 273-298, Nov. 1986

[2] P.Sweasy, A.J.Smith, "A class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus," Proc. of ISCA pp.414-423, Jun. 1986

[3] T.Matsumoto, K.Hiraki, "Dynamic Switching of Coherent Cache Protocols and its Effects on Doacross Loops," Proc. of Intl. Conf. on Supercomputing, pp. 328-337, Jul. 1993

[4] J.Lee, U.Ramachandran, "Synchronization with Multiprocessor Caches," Proc. of ISCA 1990, pp. 27-37, May 1990

[5] C.B.Stunkel, B.Janssens, W.K.Fuchs, "Address Tracing for Parallel Machines," COMPUTER Jan. 1991

[6] H.O.Bugge, E.H.Kristiansen, B.O.Bakka, "TRACE-DRIVEN SIMULATION FOR A TWO-LEVEL CACHE DESIGN IN OPEN BUS SYSTEMS," Proc. of ISCA, pp. 250-259, May 1990

[7] H.Davis, S.R.Goldschmidt, J.Hennessy, "MULTIPROCESSOR SIMULATION AND TRACING USING TANGO," Proc. of ICPP, Vol.II, pp. 99-107, Aug. 1991

[8] B.W.O'Krafka, A.R.Newton, "An Empirical Evaluation of Two Memory-Efficient Directory Methods," Proc. of ISCA, pp. 138-147, May 1990

[9] H.Amano, T.Terasawa, T.Kudoh, "Cache with synchronization mechanism," Proc. of IFIP 11th World Computer Congress, pp.1001-1006, Aug. 1989.

[10] "IEEE Standard Backplane Bus Specification for Multiprocessor Architectures: Futurebus," IEEE Jun. 1988

[11] "Draft Standard: Futurebus+," Draft 8.2, IEEE Feb. 1990

[12] T.Kudoh, T.Kimura, H.Amano, and T.Terasawa, "A parallel logic simulation algorithm based on query," Proc of ICPP, Vol.III, pp. 262-266, Aug. 1992

[13] K.Suzuki, H.Amano, Y.Takefuji, "Neural Network Parallel Computing for Channel Routing Problems," Proc. of IEEE Intl. Conf. on Neural, Robotics and Distributed Computing, 1992

[14] J.P.Singh, W-D.Weber, A.Gupta, "SPLASH: Stanford Parallel Applications for Shared Memory," Apr. 1991

[15] E.L.Lusk, R.A.Overbeek, "Use of Monitors in FORTRAN: A Tutorial on the Barrier, Self-scheduling DO-Loop, and Askfor Monitors," Tech. Report No. ANL-84-51, Rev. 1, Argonne National Laboratory, Jun. 1987