A Power-Aware Prediction-Based Cache Coherence Protocol for Chip Multiprocessors

Ehsan Atoofian and Amirali Baniasadi
ECE Department, University of Victoria
{eatoofian, amirali}@ece.uvic.ca

Abstract

Snoopy cache coherence protocols broadcast requests to all nodes, reducing the latency of cache to cache transfer misses at the expense of increasing interconnect power. We propose speculative supplier identification (SSI) to reduce power dissipation in binary tree interconnects in snoopy cache coherence implementations.
In SSI, instead of broadcasting a request to all processors, we send the request to the node more likely to have the missing data. We reduce power as we limit access only to the interconnect components between the requestor and the supplier node. We evaluate SSI using shared memory applications. We show that SSI reduces interconnect power by 23% in a 4-way multiprocessor. This comes with negligible performance cost and hardware overhead.
SSI does not change existing coherence protocols and is completely transparent to software and the operating system.

1. Introduction

Exploiting thread-level parallelism is believed to be a reliable way to achieve higher performance improvements in the future. Moreover, as technology advances provide microprocessor design with more options, finding new solutions to use the possible capabilities is necessary. Chip multiprocessing offers an attractive solution as using multiple cores makes efficient execution of parallel threads possible. Accordingly, chip multiprocessors (CMPs) are expected to become more popular as the number of on-die transistors continue to increase.
Currently available CMPs (*e.g.,* Sun's Niagara) exploit as many as eight cores. However, it is expected that the increasing demand for higher speed in multiprocessors would require using higher number of cores, effectively increasing interconnect complexity and power dissipation. This is due to the fact that multithread workloads which run simultaneously on multiprocessor nodes communicate through interconnects and dissipate significant power. As a result, inter-processor communication has become one of the bottlenecks in multiprocessor systems consuming a considerable share of the overall power (*i.e.*, up to 15% [1]).
In a shared memory multiprocessor system, the processors have to access interconnects upon frequently occurring local cache misses [3, 6, 8, 15, 17, 18]. Each local cache is responsible for maintaining the correct state for its local data and responding to requests made by other processors when necessary. Upon a cache miss, several power dissipating transactions occur on interconnects including snoop requests, invalidate messages, and block writebacks. In a write-invalidate protocol, a snoop request is broadcasted to all nodes, substantially increasing interconnect power dissipation.
Our study shows that processors providing a missing data show very high locality. In other words, for a cache miss occurring in processor A, if the required data is residing in and provided by processor B's local cache, there is a high probability that next time A is missing a data, it would be provided by B again.
In this work we exploit this phenomenon and introduce speculative supplier identification (SSI) to reduce power in interconnects. In SSI, the requesting node avoids broadcasting requests to all nodes and sends the request only to the previous supplier if there is a high confidence that the previous supplier would provide the missing data. As such, interconnect activity is reduced and only necessary links and switches (which connect the requestor and supplier) are accessed. This eliminates unnecessary activities not only in interconnects and internal switches but also in tag arrays of non-supplying processors.

In summary, we make the following contributions:

- We show supplier nodes have very high locality, i.e., for any processor there is a high probability (85% for the configuration and applications studied here) that two consecutive misses are handled by the same remote node.

- We show that a simple predictor can predict the supplier processor for a local cache miss with high accuracy. Our study shows that a single entry 4-bit predictor can identify remote suppliers with an average accuracy up to 95%.
- By limiting sending requests only to the predicted remote nodes, we reduce interconnect activity by 23% in a 4-way chip multiprocessor system. This comes with negligible impact on execution time.

It should be noted that while in this work, and as a case study, we use a binary tree, similar to Sun Fireplane [2], our method can also be used for other alternative non-bus interconnects (*e.g.*, benes and fat-tree [21]).

The rest of the paper is organized as follows. In section 2, we discuss our motivation. In section 3, we review the related background. In section 4, we discuss SSI and explain implementation details. In section 5, we discuss methodology and evaluate SSI. In section 6, we review related work. Finally, in section 7 we offer concluding remarks.

2. Motivation

In Figure 1, we show how a snoop request is handled. As presented, N processors sharing a single L_2 cache are connected through a network interconnect. Suppose that processor P_0 is about to read the elements of a shared array for the first time. Meantime assume that P_{n-1} has already read the array, and all array elements are available in P_{n-1}'s local cache. A miss occurs as soon as P_0 reads the first array element. To find the missing data, P_0 broadcasts snoop requests to all the other nodes (Figure 1.a). P_1, P_2, …, and P_{n-1} receive snoop requests and lookup their tag arrays. P_{n-1} finds the element and sends it to P_0. The system goes through the same procedure every time P_0 reads a new (missing) element.

Note that all interconnect components are accessed every time that an array element is accessed in P_0. However, only one of the many processors (P_{n-1} in the example) provides the data. This approach provides fast access to the missing data but is inefficient from the energy point of view [6].

In this work, we address this design inefficiency and introduce speculative supplier identification (or simply SSI) to reduce power in snoop-based chip multiprocessor systems.

SSI relies on the fact that there is a high chance that two consecutive cache missies in a local cache are supplied by the same remote node. We refer to this phenomenon as supplier locality. Figure 2 reports supplier locality for the Splash-2 benchmarks used in this study (see section 5.1 for methodology). Locality shows how often the current supplier of a missing data in a local node is the same as the previous supplier. Except for *barnes*, all benchmarks have a locality higher than 70%. On average, supplier locality is 85%.

SSI exploits this locality and uses a simple predictor to speculate the remote node providing the missing data. SSI sends the associated request only to the speculated node. As such, only the path between the requestor and supplier is accessed (Fig. 1.b). This can reduce power compared to a conventional snoop-based system where all nodes are accessed uniformly and regularly.

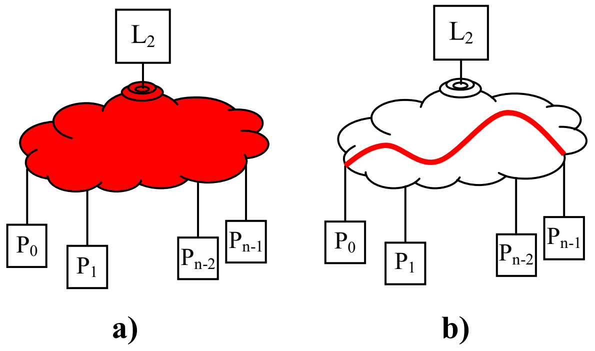


Figure 1. a) Conventional snooping. **b)** SSI snooping.

Power savings is possible if the supplier node is predicted accurately. In the event of a misprediction, however, snoop requests have to be broadcasted to all nodes resulting in energy and latency penalty. This penalty can negate our savings if the necessary behavior is not there. However, as we show later in this work, savings outweigh the associated overhead.

Note that although speculation has been used in directory-based cache coherence protocols (*e.g.,* to reduce cache to cache transfer latency [12]), to the best of our knowledge, this is the first time it is being used in snoopy cache coherence protocols.

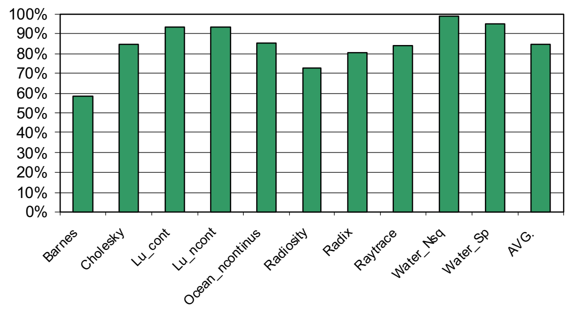


Figure 2. Supplier locality for the applications and the configuration used in this study.

3. Background

In section 3.1, we review a basic write-invalidate snoop protocol. We discuss the interconnect architecture used in this work in section 3.2.

3.1 Write-invalidate snoop protocol

The snoop protocol is used to maintain cache coherence in shared memory multiprocessors. A cache coherence protocol is a collection of finite state machines that change their states in response to their local processors' requests and messages received on the bus. Each finite state machine is distributed over nodes with each local cache maintaining the state of its local data. All caches connected to the bus monitor bus transactions. Caches update the state of their data and reply to requests whenever necessary.

On a cache read miss, a request is broadcasted on the bus. All caches lookup their tag arrays with the address of requesting message. If a cache has the requested (valid) data, it sends the data to the requestor. In a write-invalidate protocol, when a write miss occurs, an invalidate request is sent over the bus. All processors that have a copy of the message address invalidate the corresponding entry in their local caches.

3.2. Tree-base interconnect structure

The address interconnect used in this paper is similar to Sun Fireplane interconnect[2]. Figure 3 shows the structure of the address interconnects. Address interconnect is implemented using two level switches. Processors are located at the leaves of the tree, and the L_2 cache is connected to the root. Memory is off the chip and is connected to the L_2 cache. At any moment, at most one message exists in the tree. It should be noted that from a processor viewpoint, the tree structure is similar to a bus [6].

Upon broadcasting a request, the request is first sent to the root switch. In the next step, the root switch sends copies of the request down to all processors. Processors use the received data and lookup their tag arrays before replying to the root switch. If any of the processors has the data, the root switch selects the closest processor to the requestor and forwards the processor's message. If none of the processors hold the requested data, the root switch sends a request to the L_2 cache. If the data is not found in the L_2 cache, the processor sends an off the chip request to the memory.

In SSI, we modify the baseline cache coherence protocol and reduce the number of steps involved. Instead of sending request to the root and then having the request broadcasted by the root, the request is directly sent to other nodes. This reduces the number of accesses to the links by one and results in processors receiving snoops requests at different cycles. For example in Figure 3, under our system, a request sent by P_0 is received by P_1 sooner than P_3.

In our system, and similar to the conventional snoop-based system, processors reply to the root switch after tag lookup is performed. However, in our system, the root switch does not receive replies from processors at the same time. The root switch should wait to receive all replies and then select the closest supplier to the requestor or send the request to the L_2 cache.

We use separate data and address interconnects. Data interconnect is similar to address interconnect and uses two level of switches. When the supplier is determined by the cache coherence protocol, the supplier sends data to the requestor through the data interconnect. In this work we focus on the address interconnect.

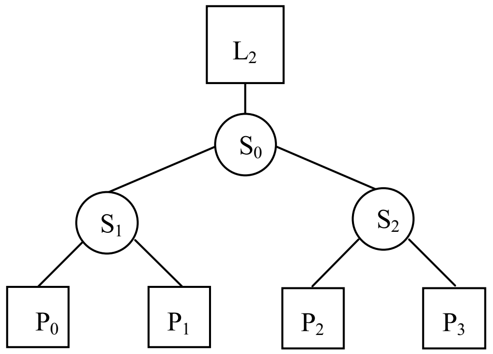


Figure 3. Address interconnect structure.

4. Implementation

In our proposed architecture, each node is equipped with a small single-entry predictor to speculate the supplier for missing data reads[1] in the local cache.

Figure 4 depicts a typical processor in our CMP configuration. Each processor includes a core, a private L_1 cache, and a supplier predictor. Each predictor entry is equipped with two fields: a $\log_2 N$ bit field, where N is the number of processors, referred to as speculated supplier or SPL (SPL has two bits in our example of four processors) and an n-bit saturating counter. SPL records the last supplier node for the processor. To achieve high accuracy, we use saturating counters. If the prediction is correct, the counter is incremented. For mispredictions, the counter is reset to zero. The predictor is trusted only if the value of the saturating counter is more than a pre-decided threshold. Note that the area and energy overhead associated with the predictor is negligible as the predictor only includes an

[1] A predictor may need to predict multiple nodes for write misses [5], since several nodes may share the missing data, and invalidation request should be sent to all of the sharers. However, our predictor can predict only one node. Extending this work to cover more complex scenarios is part of our ongoing research.

n-bit counter and a $\log_2N$ bit register. We refer to an SSI system using an n-bit counter as SSI-n (*e.g.*, SSI-2 uses a single 2-bit counter).

Initially the predictor does not include any information. Therefore, no prediction is made when the first miss occurs as there is no record of any previous supplier. Under such circumstances, the processor follows the conventional approach and broadcasts a snoop request on the interconnect. When the supplier processor responds, the predictor is updated with the supplier number. Upon the next cache miss, if the saturating counter is more than a threshold, the predictor speculates, and the request is only sent to the predicted node. The predicted node looks up for the requested address, and replies. For accurate predictions and if the valid data is found in the speculated supplier, no additional step is needed. Consequently, instead of accessing all switches, links, and tag arrays, only those components that are between the requestor and the supplier will be accessed. This reduces power in both interconnect and tag arrays.

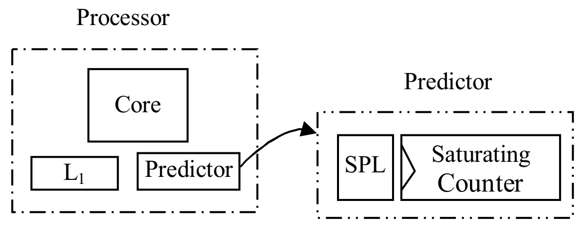


Figure 4. A processor using a node predictor: each predictor includes a speculative supplier (SPL) and a saturating counter.

In the event of mispredicting the supplier, a snoop request is broadcasted by the requestor to all the other processors. The cost associated with the misprediction includes the extra access to interconnect and an increase in data communication latency. However, as we show later, the benefits of correct predictions outweigh the associated misprediction costs.

It is important to note that SSI does not impose any changes to the underlying cache coherence protocol. In a MESI protocol [21], the state of a requested cache block in the speculated supplier node is in one of the following four states: modified, exclusive, shared, or invalid. If the state is modified, exclusive, or shared, and speculation turns out to be accurate, then both the supplier and requester will end up having the shared state. However, if the state of requested cache block is invalid, a misprediction will occur and the requester will broadcast a snoop request. Consequently, whether the prediction is right or wrong, SSI would not change any state transition in the MESI protocol. Moreover, SSI does not impose any limitation on software, and is completely transparent to operating system.

To provide better understanding, in Figure 5, we show the actions taken under SSI for the example discussed earlier in section 2. For simplicity, we assume SSI-1 with a prediction threshold equal to 0.

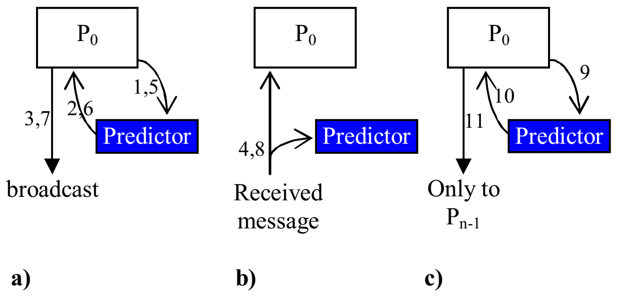


Figure 5. An SSI example: P_0 and P_{n-1} share an array. P_{n-1} has already read the elements and has them in its L_1 cache. P_0 starts reading the (missing) array elements. **a.1)** P_0 asks the predictor for the likely supplier. **a.2)** The predictor cannot make a prediction as there is no previous record. **a.3)** P_0 broadcasts snoop request to all nodes. **b.4)** P_{n-1} sends the data to P_0. Predictor is updated with the supplier processor number, and data is stored in P_0's local cache. **a.5, a.6)** Upon missing the array's second element, the predictor speculates P_{n-1} as the likely supplier. The predictor is not trusted since the saturating counter is not greater than the threshold. **a.7)** P_0 broadcasts snoop request to all nodes. **b.8)** The saturating counter is incremented as the predictor has made a correct prediction. **c.9)** For the third array element, P_0 probes the predictor. **c.10)** Predictor speculates that P_{n-1} is supplier. **c.11)** P_0 sends the request only to P_{n-1} (instead of broadcasting). P_{n-1} provides the array element.

5. Evaluation

In this section, we evaluate SSI. In section 5.1 we present the methodology. In section 5.2 we report the results.

5.1. Methodology

We use SPLASH-2 [4] benchmarks (details reported in Table 1) to evaluate our scheme. For simulation, we used the execution driven mode of Sesc[7] modeling the out of order processors and the memory subsystem presented in Table 2. In this work we focus on a 4-way CMP. Note that extending our scheme to systems with higher number of processors is possible and is part of our future work. We used MESI protocol to maintain cache coherence in L_1 caches.

In section 5.2, we report SSI accuracy and coverage. In section 5.3 and 5.4, we report how SSI impacts performance and interconnect activity. We use interconnect activity as an implementation independent and indirect power estimate. Modeling of links and switches to achieve direct power estimation is part of our ongoing research. We compare SSI with the conventional baseline cache coherence where snoop requests are broadcasted to all nodes upon any local cache miss. To make better evaluation of SSI possible, we also compare SSI to serial snooping [6]. In serial snooping, a snoop request is initially sent only to the neighbor node. The neighbor node looks up its local cache and replies to the requestor if the requested data is found, otherwise, it sends the snoop request to the next node. In both SSI and serial snooping, at any moment, at most one message exists in interconnect. As such, memory consistency is maintained accurately [20]. To the best of our knowledge, serial snooping is the only power aware snoop-based cache coherence in binary tree interconnects.

Table 1. Splash2 benchmarks and input parameters

Benchmarks	Input Parameters
Barnes	16k particles
Cholesky	tk29.O
Lu(contiguous, non-contiguous)	512×512 matrix, B=16
Ocean(non-contiguous)	258×258 grid
Radiosity	-batch –room
Radix	8M keys
Raytrace	Balls4.env
Water(nsquared, spatial)	4k molecules

Table 2. System parameters

Processor	Interconnect	Memory System
frequency: 1 GHz branch predictor:16K entry bimodal and gshare branch penalty: 17 Fetch/issue/commit: 6/4/4 RAS: 32 entries BTB: 2K entries, 2-way	bus clock cycle: 7 ns switch latency: 1 cycle link latency: 1 cycle interconnect width: 64B	cache block size: 64B split I-L1,D-L1: 32KB, 4-way L1 latency: 2 L2: 512KB/8-way L2 latency: 11 memory latency: 70 ns

5.2. Coverage and Accuracy

In this section, we report coverage and accuracy for SSI. Note that in a 4-way multiprocessor there are four predictors, one predictor for each processor. We report average data for the four predictors.

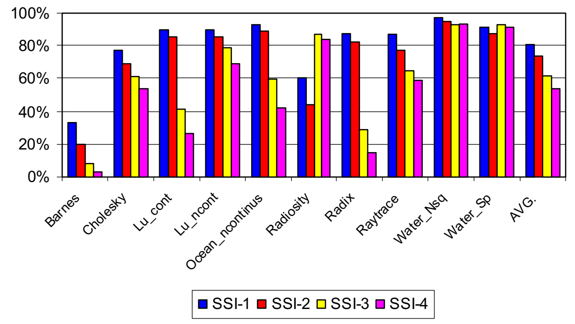
We define coverage as the percentage of all supplier nodes in cache to cache transfers that are accurately identified by predictors. Figure 6.a shows coverage for predictors with different sizes. We report for SSI-1, SSI-2, SSI-3 and SSI-4. We use thresholds values equal to zero, three, six, and 14, for SSI-1, SSI-2, SSI-3 and SSI-4 respectively. We picked theses thresholds after testing different alternatives. In general, coverage reduces as the counter size increases. On average, coverage varies from 54% to 80% for different counter sizes.

Figure 6.b shows accuracy for predictors with different counter sizes. Accuracy shows how often the speculated supplier turns out to be the correct one. In general, accuracy improves as counter size increases. On average, accuracy changes from 91% to 95% for different counter sizes.

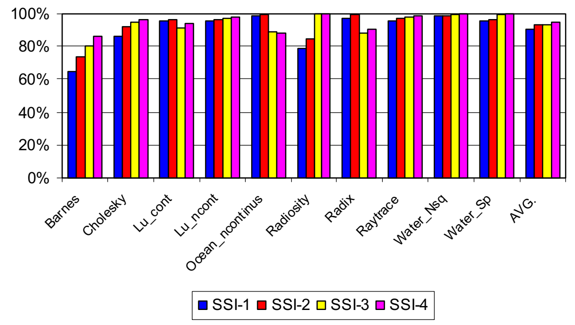
5.3. Execution Time

In this section, we report execution time for SSI and serial snooping [6] compared to the baseline cache coherence protocol. Table 3 reports execution time for different benchmarks. Positive numbers indicate an increase in execution time.

SSI has negligible impact on execution time. For most benchmarks, the impact is less than 0.5%. Note that for some benchmarks (*e.g., ocean_ncontinious*) SSI improves execution time. This could be explained by the following: a) for these benchmarks, often the missing data is provided by on-chip caches rather than the memory and b) SSI can speculate the supplier caches with high accuracy.



a)Coverage



b)Accuracy

Figure 6. Coverage and accuracy of predictors equipped with 1-, 2-, 3- and 4-bit counters.

The last column of Table 3 reports execution time for serial snooping. On average, serial snooping increases execution time by 9.6%. In some benchmarks, *e.g. raytrace*, serial snooping increases run time considerably. In these benchmarks, quite often the supplier is not close to the requestor, and serial snooping increases latency of cache to cache transfers.

5.4 Activity Reduction

In Figure 7, we report activity reduction in links, switches, and tag arrays for SSI and serial snoop compared to the baseline cache coherence scenario. Generally, activity reduction improves as counter size decreases. This is intuitive as bigger counters have lower coverage.

On average, as reported in Figure 7.a, SSI-1, SSI-2, SSI-3 and SSI-4 reduce link activity by 23%, 21%, 18%, and 18% respectively. Serial snooping reduces link activity by 31%. However, this comes with significant increase in run time.

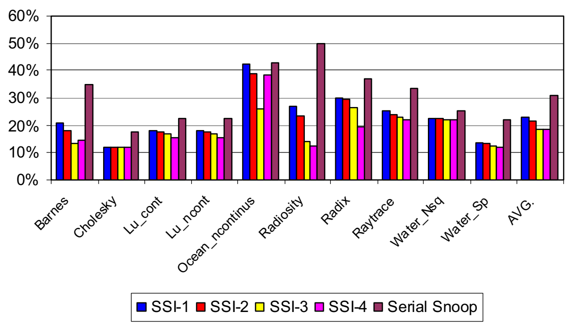
Table 3. Effect of predictors on execution time.

	SSI-1	SSI-22	SSI-3	SSI-4	Serial
Barnes	0.69%	1.62%	-0.27%	-0.20%	7.27%
Cholesky	0.01%	0.05%	-0.04%	-0.04%	18.57%
Lu_cont	-0.01%	-0.01%	0.00%	0.01%	3.89%
Lu_ncont	-0.01%	-0.01%	0.00%	0.01%	3.89%
Ocean_ ncont	-2.14%	-1.20%	-3.55%	-3.62%	5.78%
Radiosity	-0.03%	0.08%	-0.02%	0.00%	0.03%
Radix	0.00%	0.00%	0.00%	0.00%	0.00%
Raytrace	-0.15%	-0.61%	-0.75%	-0.58%	27.17%
Water_Nsq	-0.90%	-0.91%	-0.90%	-0.90%	11.00%
Water_Sp	0.06%	0.03%	0.01%	0.00%	2.30%
AVG.	-0.25%	-0.10%	-0.55%	-0.53%	7.09%

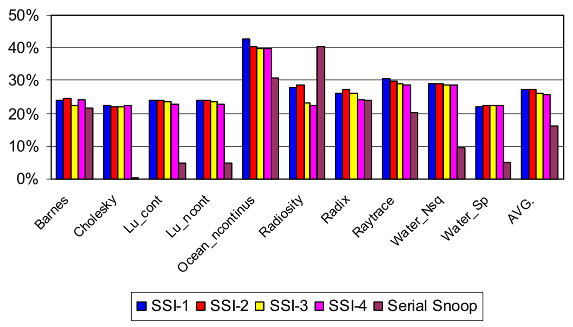
On average, as reported in Figure 7.b, SSI-1, SSI-2, SSI-3 and SSI-4 reduce interconnect switch activity by 27%, 27%, 26% and 25% respectively. Serial snooping reduces switch activity by 16%. For half of the benchmarks, SSI reduces switch activity twice that achieved by serial snooping. This is due to the fact that in serial snooping, whenever a cache lookup fails, the request is forwarded to the next node. Consequently, the closest switch to the processor is accessed at least twice. For example, in Figure 3, if P_1 receives a snoop request from P_0, and the requested address misses in P_1, the request is forwarded to P_2 through S_1 , and S_1 is accessed twice: once, when P_0 sends snoop request to P_1, and once when P_1 forwards snoop request to P_2. This is not the case under SSI. The two methods improve tag array power competitively (see Figure 7.c). SSI-1, SSI-2, SSI-3 and SSI-4 improve tag array accesses by 16%, 15%, 10% and 9% respectively. Serial snooping improves tag array power by 14%.

Note that tag array activity reduction is zero for *cholesky*. Our study shows that cache to cache transfers occur rarely for *cholesky*. As such, despite high accuracy, SSI does not improve tag array activity. However, *cholesky* shows activity reduction in links and switches as the result of the cache coherence step reduction explained in section 3.2.

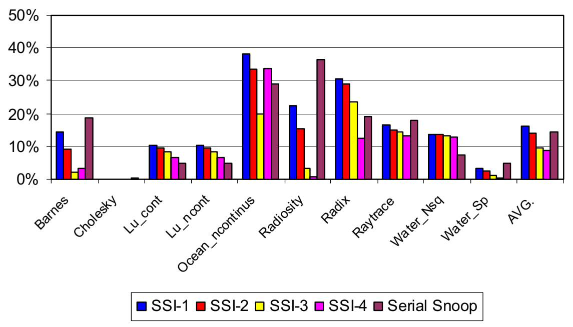
Overall, SSI-1 provides substantial power savings with negligible performance degradation, and minimal hardware overhead.



a) Activity reduction of links



b) Activity reduction of switches



c) Activity reduction of tag arrays

Figure 7. Activity reduction in links, switches, and tag arrays.

6. Related Work

Saldanha and Lipasti [6] proposed serial snooping to reduce interconnects power. We introduce SSI as an alternative speculative approach and compare our results to their method.

Bjorkman et al. [19] proposed hints to reduce cache miss penalty. For each block in memory, they use one hint to identity the potential holder of the copy. When a cache miss can not be serviced in the local node, a request is sent to both home directory and to the hint node. If hint node has the copy, it sends it to the requestor and this reduces the cache miss delay by one hop. Otherwise, home directory provides data following convention method. While their method improves performance, sending two requests for each cache miss pollutes interconnect network and increases power dissipation in inter-node communication. In addition, they use hints for each memory block which increases hardware complexity dramatically. SSI, however, uses one SPL per node to reduce power dissipation in interconnects.

Mukherjee and Hill [9] used prediction in distributed shared memory systems to speculate coherent messages in advance. Their work is based on the observation that memory blocks have a small number of repetitive sharing patterns. They used a general pattern-based predictor derived from two-level PAp branch predictor [10]. Memory Sharing Predictor (MSPs) [11] is a special type of general pattern-based predictor. MSP only predicts remote memory accesses and not the subsequent coherent messages. As such, it reduces predictor cost and improves accuracy.

While both works discussed above use speculation in directory-based cache coherence, we apply speculation in snoopy cache coherence to reduce power of interconnect.

Owner predictor [12] reduces latency of cache to cache transfer in cc-NUMA. By using a two level predictor, 3-hop misses are converted to 2-hop misses. The first level of predictor determines those misses that are satisfied by cache to cache transfers. The second level determines the list of nodes that have a valid copy of memory line. Requests are sent directly to the speculated nodes, removing the directory from the critical path. Our work is different as we focus on snoop-based protocols for chip multiprocessors and exploit much simpler predictors.

In Jetty [13] snoops from remote nodes are filtered to reduce the number of L_2 cache accesses in SMPs. Each node has a filter on the bus side of the L_2 cache, checking the snoop requests sent from remote nodes. The filter identifies situations where the L_2 cache does not include the requested data and eliminates the associated extra L_2 tag arrays lookups. In RegionScout [14], a node determines in advance that a coarse grain region is not available in none of the other nodes. As such, the request is sent directly to the memory, reducing both interconnect power and bandwidth. SSI can be used on top of Jetty and RegionScout possibly increasing power savings.

Ekman et al. [16] evaluated Jetty and serial snooping in chip multiprocessors. They concluded that Jetty can not improve power of caches in CMPs as the power loss due to filters in Jetty outweighs cache power savings. Also, they demonstrated that serial snooping has little benefit in CMPs. Their study, however, did not investigate power savings in interconnects.

7. Conclusion

We proposed a prediction-based cache coherence protocol to speculate the supplier processor. By using a low overhead predictor, requests are sent directly to the speculated supplier. We save power as we avoid broadcasting whenever there is high confidence in the prediction outcome. We showed that simple predictors can effectively identify a considerable share of suppliers with high accuracy. Our method results in considerable activity reduction while improving performance slightly. We showed that SSI reduces activity of links, switches, and tag arrays by 23%, 27%, 16% respectively.

Acknowledgement

This work was supported by the Natural Sciences and Engineering Research Council of Canada, Discovery Grants Program and Canada Foundation for Innovation, New Opportunities Fund.

Refrences

[1]Loghi, M., Poncino, M. and Benini, L., Cycle- Accurate Power Analysis for Multiprocessor System-on-a-Chip, In *Proceeding of the 2004 ACM Great Lakes Symposium on VLSI , pp. 401-406*, 2004.

[2] Alan E. Charlesworth. The Sun Fireplane System Interconnect, In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, 2001.

[3] M. E. Acacio, J. Gonzalez, J. M. Garcia, and J. Duato, The Use of Prediction for Accelerating Upgrade Misses in CCNUMA Multiprocessors, In *Proceedings of PACT-11*, 2002.

[4] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *International Symposium on Computer Architecture*, June 1995.

[5] Robert C. Steinke, Gary J. Nutt, A unified theory of shared memory consistency, Journal of the ACM (JACM), v.51, n.5, p.800-849, September 2004.

[6] C. Saldanha and M. H. Lipasti, Power Efficient Cache Coherence, High Performance Memory Systems, edited by H. Hadimiouglu, D. Kaeli, J. Kuskin, A. Nanda, and J. Torrellas, Springer-Verlag, 2003.

[7] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, K. Strauss, S. Sarangi, P. Sack, and P. Montesinos. SESC Simulator, Jan 2005. http://sesc.sourceforge.net.

[8] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood, Multicast Snooping: A New Coherence Method using a Multicast Address Network, SIGARCH Comput. Architure News, pp. 294–304, 1999.

[9] Shubhendu S. Mukherjee and Mark D. Hill, Using prediction to accelerate coherence protocols, In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998.

[10] Tse-Yuh Yeh and Yale Patt, Alternative implementations of two-level adaptive branch prediction, In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

[11] A.-C. Lai and B. Falsafi, Memory sharing predictor: the key to a speculative coherent DSM, In *Proceedings of the 26th annual international symposium on Computer architecture*, pp. 172–183, 1999.

[12] M. E. Acacio, J. González, J. M. García, and J. Duato, Owner Prediction for Accelerating Cache-to-Cache Transfers in a cc-NUMA Architecture, In *Proceedings of SC2002*, Nov. 2002.

[13] A. Moshovos, B. Falsafi and A. Choudhary, JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers, In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, January 2001.

[14] J. Cantin, A. Moshovos, M. Lipasti, J. Smith, and B. Falsafi, Coarse-Grain Coherence Tracking: RegionScout and Region Coherence Arrays, IEEE Micro, v.26, n.1, pp. 70-79, Jan-Feb 2006.

[15] J. Huh, J. Chang, D. Burger, and G. S. Sohi, Coherence Decoupling: Making Use of Incoherence, In *Proceedings of ASPLOS-XI*, pp. 97-106, 2004.

[16] M. Ekman, F. Dahlgren, and P. Stenström: Evaluation of Snoop-Energy Reduction Techniques for Chip-Multiprocessors. In Proceedings of the First Workshop on Duplicating, Deconstructing, and Debunking (WDDD-1), May 2002.

[17] Milo M. K. Martin, Pacia J. Harper, Daniel J. Sorin, Mark D. Hill, and David A. Wood, Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared-Memory Multiprocessors, In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 206-217, 2003.

[18] K. M. Lepak and M. H. Lipasti, Temporally Silent Stores, In *Proceedings of ASPLOS-X*, pages 30–41, 2002.

[19] M. Bjorkman, F. Dahlgren, and P. Stenstrom, Using Hints to Reduce the Read Miss Penalty for Flat COMA Protocols, In *Proceedings of the 28th Annual Hawaii International Conference of System Sciences*, pages 242-251, January 1995.

[20] A. Landin, E. Hagersten, and S. Haridi, Race-free interconnection networks and multiprocessor consistency, In Proc. of the 18th Intl. Symp. on Comp. Architecture, 1991.

[21] D. E. Culler, J. Singh, A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann Publishers, San Francisco, Calif., 1998.