Available online at www.sciencedirect.com

ScienceDirect

Journal of Systems Architecture 54 (2008) 507–518

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Using supplier locality in power-aware interconnects and caches in chip multiprocessors

Ehsan Atoofian, Amirali Baniasadi [*]

Department of Electrical and Computer Engineering, University of Victoria, Victoria, B.C., Canada V8W 3P6

Received 23 December 2006; received in revised form 31 August 2007; accepted 18 September 2007
Available online 12 October 2007

Abstract

Conventional snoopy-based chip multiprocessors take an aggressive approach broadcasting snoop requests to all nodes. In addition each node checks all received requests. This approach reduces the latency of cache to cache transfer misses at the expense of increasing power. In this paper we show that a large portion of interconnect/cache transactions are redundant as many snoop requests miss in the remote nodes.

We exploit this inefficiency and introduce power optimization techniques for chip multiprocessors. Our optimizations rely on the observation that in a snoopy-based shared memory system the data supplier can be predicted with high accuracy. Our optimizations reduce power by eliminating unnecessary activity at both the requester and the supplier end of snoop requests.

We reduce power as we (a) avoid broadcasting snoop requests to all processors and (b) avoid tag lookup for all nodes and for all requests arriving. In particular, we use supplier locality and introduce the following two optimizations.

First, and at the requester end, we introduce speculative selective request (SSR) to reduce power dissipation in the binary tree interconnect. In SSR, we send the request only 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.

Second, and at the supplier end, we propose speculative tag lookup (STL) to reduce power dissipation in data caches. We filter those accesses more likely to miss in the L_1 cache.

Using shared memory applications, we show that by limiting snoop requests to the speculated nodes we reduce interconnect power by 25% in a four-way multiprocessor. Moreover, we show that speculative tag lookup reduces power in tag arrays by 14.1% in a four-way multiprocessor. Both optimizations come with negligible performance loss and hardware overhead.

Keywords: Power-aware microarchitectures; Chip multiprocessors; Interconnect; Cache

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.

In order to achieve high performance, conventional snoopy-based CMPs take an aggressive and uniform approach, broadcasting all snoop requests to all nodes. This approach reduces the latency of cache to cache transfer misses as it assures that all nodes receive and process the request at the earliest possible. From the energy point of view, however, this aggressive approach translates to excessive and often unnecessary power dissipation. A large

[*] Corresponding author. Tel.: +1 250 721 8613; fax: +1 250 721 6052.
E-mail addresses: eatoofian@ece.uvic.ca (E. Atoofian), amirali@ece.uvic.ca (A. Baniasadi).

portion of the dissipated power is wasted since, as we show in this work, a high percentage of such requests are received and processed by nodes that do not have the missing data.

In this work, we use *supplier locality* to address this design inefficiency. Our study shows that the processor supplying a missing data is often predictable. 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 supplied by B again.

We take this locality into account and eliminate a large number of unnecessary snoop requests and cache lookups. We introduce two power optimization techniques for snoopy cache coherence protocols:

First, we introduce *speculative selective request* (SSR) at the requester end of snoop requests. In SSR, we avoid broadcasting requests to all the nodes. Instead we send the request only to the predicted supplier if there is a high confidence that it would provide the missing data. We reduce interconnect power since 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 the tag arrays of the non-supplying processors.

Second, we introduce *speculative tag lookup* (STL) to reduce power dissipation in the tag arrays at the supplier end of snoop requests. In a write-invalidate protocol, all processors access their local caches upon receiving snoop requests and respond to the requester processor when necessary. However, a significant fraction of these snoops miss in the caches and result in unnecessary power dissipation [13–15].

In SLT, a remote node receiving a snoop request avoids accessing its tag array if there is a high confidence that the tag lookup would be unsuccessful. As such, SLT reduces power dissipation in tag arrays.

In summary, we make the following contributions:

- We show that a simple and small predictor using node IDs can often predict data suppliers with high accuracy.
- We introduce SSR to reduce power dissipation in interconnects. SSR limits sending requests only to the predicted remote nodes. We show that SSR reduces interconnect power dissipation by 25% in a four-way chip multiprocessor system. This comes with negligible impact on performance.
- We introduce STL to reduce power dissipation in tag arrays by eliminating a considerable portion of unnecessary cache snoops. We show that STL reduces tag array power dissipation by 14.1% in a four-way chip multiprocessor system with negligible impact on performance.

The rest of the paper is organized as follows. In Section 2, we review related work. In Section 3, we review the related background. In Section 4, we discuss supplier locality. In Section 5, we discuss SSR. In Section 6, we review SLT. In Section 7, we discuss methodology and results. Finally, in Section 8 we offer concluding remarks.

2. Related work

Several speculative methods have been proposed to improve performance in directory-based cache coherence protocols [9,11,12,19]. To the best of our knowledge, this is the first time ID-based speculation is used to reduce power in snoopy cache coherence protocols.

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. For a local cache miss, a request is sent to both home directory and to the hint node. If the hint node has the copy, it sends it to the requester and this reduces the cache miss delay by one hop. Otherwise, the home directory provides data following the conventional method.

Mukherjee and Hill [9] used prediction in distributed shared memory systems to speculate coherent messages in advance. They used a general pattern-based predictor derived from the 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. Owner predictor [12] reduces latency of cache to cache transfer in cc-NUMA. Requests are sent directly to the speculated nodes, removing the directory from the critical path.

All the speculative methods mentioned above target performance and rely on cache block addresses. SSR is different as we use processor ID to improve power dissipation.

Saldanha and Lipasti [6] proposed serial snooping to reduce interconnects power. We introduce SSR as an alternative speculative approach and compare it to serial snooping in Section 7.

In Jetty [13], snoops from remote nodes are filtered to reduce the number of L_2 cache accesses in symmetric multiprocessors (SMPs). Each node has a filter on the bus side of the L_2 cache, checking the snoop requests sent from the 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 array lookups. Jetty uses two different filters, one holding addresses that do not exist in the L_2 cache, and the other holding addresses existing in the L_2 cache. Both structures use tables indexed by the cache line addresses. Ekman et al. [16] evaluated Jetty in the context of CMPs. Their study reported that power savings achieved by Jetty are often negated by the overhead associated with the filters. They concluded that Jetty, while serving well for SMPs, would not improve power in CMPs.

In RegionScout [14], a node determines in advance that a coarse grain region is not available in any other node. Consequently, instead of broadcasting, the request is sent directly to the memory, reducing both interconnect power and bandwidth. Similar to Jetty, RegionScout uses address of cache blocks to filter those snoop requests that will miss in all remote nodes.

STL is different from Jetty and RegionScout as instead of using the address of cache blocks, it uses the processor ID to speculate tag lookup outcome. As such, STL does not need the large tables indexed by address of cache blocks.

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 the 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-based interconnect structure

The address interconnect used in this paper is a binary tree similar to the Sun Fireplane interconnect [2]. While we focus on the binary tree interconnect in this work, it should be noted that our techniques could also be used for other alternative interconnects (*e.g.*, benes and fattree [18]). Fig. 1 shows the structure of the address interconnects. The 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. Note that from the processor viewpoint, the tree structure is similar to a bus [6].

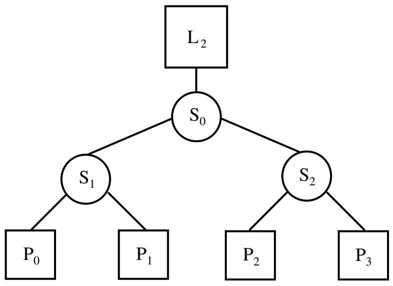


Fig. 1. Address interconnect structure.

Upon broadcasting a request, the request is first sent to the root switch. At 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.

We use separate data and address interconnects. The data interconnect is similar to the address interconnect and uses two level of switches. When the supplier is determined by the cache coherence protocol, the supplier sends the data to the requestor through the data interconnect.

We evaluate our techniques using the snoopy cache coherent protocol often used in mid-sized multiprocessor systems. It should be noted, however, that our techniques can be applied to larger multiprocessor systems as the binary tree used here could be exploited in a wide range of system sizes. For example a previous study shows how large Fireplane systems use multiple snooping coherence domains to maintain the low latency for local memory accesses provided by snoopy coherency and the bandwidth scalability of directory coherency [2]. Accordingly, each domain is a mid-size multiprocessor system that uses the snoopy cache coherence protocol. For inter-domain transactions, the directory protocol maintains cache coherence. Our techniques can be used in such systems as they can be applied within domains to reduce power dissipation for local transactions.

4. Supplier locality

Our study shows that there is a high chance that two consecutive cache misses in a local cache are supplied by the same remote node. We refer to this phenomenon as *supplier locality*. We study and exploit this locality at both the requester and the supplier end of the interconnect.

In Fig. 2, we report supplier locality at both ends for the Splash-2 benchmarks used in this study (see Section 7.1 for methodology). The first (left) bar for each application reports supplier locality at the requester end. At this end, 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 82%. The second (right) bar for each application reports supplier locality at the supplier end. Our study shows that if a snoop request from a remote node is missed in a local cache, chances are, next time the same remote node sends a snoop request, it will miss in the same local cache. We refer to this as

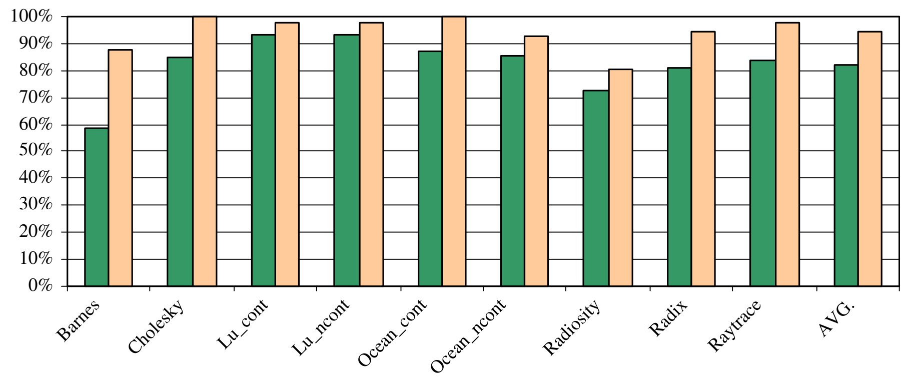


Fig. 2. Supplier locality at the supplier end (first bar for each application) and the requester end (second bar for each application).

supplier locality at the supplier end. Note that we measure this locality regardless of the address of cache block and based on the requester node ID. This locality is 94% on average and higher than 80% for all applications.

We use supplier locality at both ends and suggest two optimizations. At the requester end we suggest SSR to reduce the number of requests sent. SSR avoids sending a request to all nodes by sending requests to nodes more likely to provide the data. As such, only the path between the requestor and the supplier is accessed. This can reduce power compared to a conventional snoop-based system where all nodes are accessed uniformly and regularly.

At the supplier end we suggest STL. STL avoids tag lookup for all arriving requests by limiting lookups to those more likely to return the data. In STL L_1 caches are snooped only when there is high confidence that a miss will not occur. This can reduce power in tag arrays compared to a conventional snoop-based system if the prediction turns to be correct.

Power savings is possible if the supplier node is predicted accurately at both ends. In the event of a misprediction, for SSR, snoop requests have to be broadcasted to all nodes resulting in energy and latency penalty. For STL, if misprediction occurs, all tag arrays that have not been snooped should be snooped resulting in energy and latency penalty.

Misprediction 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 for both optimizations

5. SSR

In this section, we discuss motivation and implementation details for SSR.

5.1. Motivation

In Fig. 3, 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 (Fig. 3a). $P_1, P_2, \ldots, P_{n-1}$ 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.

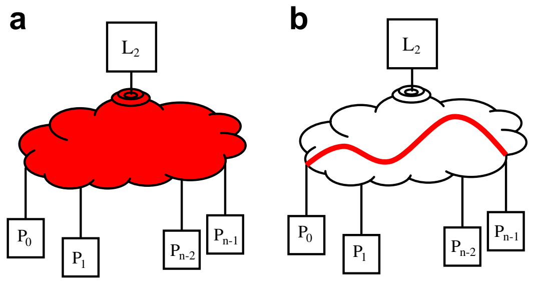


Fig. 3. (a) Conventional snooping and (b) SSR snooping.

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].

5.2. Implementation

In SSR, we modify the baseline cache coherence protocol and reduce the number of steps involved. Instead of sending the 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 Fig. 1, and 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

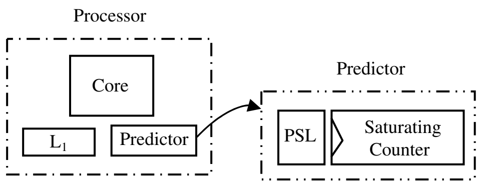


Fig. 4. A processor using a node predictor: each predictor includes a speculative supplier (PSL) and a saturating counter.

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.

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

Fig. 4 depicts a typical SSR-enhanced 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 the predicted supplier or PSL (PSL has two bits in our example of four processors) and a q-bit saturating counter. PSL 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 predecided threshold. Note that the area and energy overhead associated with the predictor is negligible as the predictor only includes a q-bit counter and a $\log_2 N$ bit register. We refer to an SSR system using a q-bit counter as SSR-q (*e.g.*, SSR-2 uses a single two-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 ID. 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 are accessed. This reduces power in both interconnect and tag arrays.

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 SSR does not impose any changes on the underlying cache coherence protocol. In a MESI protocol [18], 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 the 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, SSR would not change any state transition in the MESI protocol. Moreover, SSR does not impose any limitation on software, and is completely transparent to operating system.

Note that we use sequential consistency as the memory consistency model for our simulations. As mentioned earlier, SSR is used for read operations, and is not applied for write operations. SSR may reduce read latency if it speculates the supplier node correctly. However, this does not violate sequential consistency as we take a conservative approach and assume (similar to earlier research [6]) that the interconnect contains at most one message at any point in time [5]. An alternative and less conservative policy would allow multiple messages so long they have different addresses. This would require a central unit that monitors address of messages [18]. Prior to sending a message, each node would have to submit the message address to the central unit. The central unit has to compare the address to the already existing addresses in the interconnect. The node is permitted to send its message if its address does not match

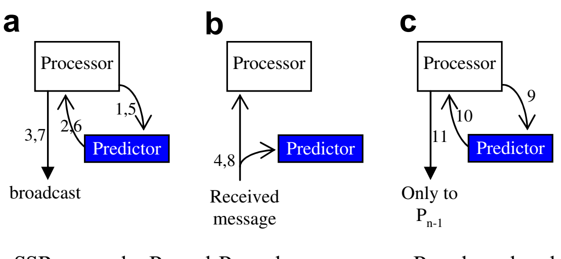


Fig. 5. SSR example: P_0 and P_{n-1} share an array. P_{n-1} has already read the elements and has them in its L_1 cache. P_0 starts reading the (missing) array elements. (a.1) P_0 asks the predictor for the likely supplier. (a.2) The predictor cannot make a prediction as there is no previous record. (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 the supplier. (c.11) P_0 sends the request only to P_{n-1} (instead of broadcasting). P_{n-1} provides the array element.

[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. To keep our predictors simple, we do not use SSR for write misses.

any of the existing addresses. Exploiting a central checker restricts system scalability and results in additional complexity. Therefore, we do not use such a policy in our simulations.

To provide better understanding, in Fig. 5, we show the actions taken under SSR for the example discussed earlier in Section 5.1. For simplicity, we assume SSR-1 with a prediction threshold equal to 0.

6. STL

In Fig. 6, we present the typical processor used as our STL-enhanced CMP configuration. Each processor includes a core, a private L_1 cache, and a predictor. The predictor has $(N-1)$ entries where N is the number of nodes in the CMP. The ith entry is used to speculate if a snoop request for a missing data read[2] from processor i hits or misses in the L_1 cache.

Each predictor entry has two fields: one is a saturating counter (SC) and the other is an *availability bit* (A). Upon an accurate prediction, the counter is incremented. For mispredictions, the counter is reset to zero. The predictor is trusted only if the value of the associated saturating counter is more than a pre-decided threshold. A_i is set if the most recent snoop request from processor i has resulted in an L_1 cache hit in the processor receiving the request, otherwise A_i is reset.

Note that the area and energy overhead associated with the predictor is negligible as the predictor only includes $(N-1)$ counters and $(N-1)$ bits. We refer to an STL system using q-bit saturating counters as STL-q (*e.g.*, STL-2 uses two-bit saturating counters).

Initially, none of the predictor entries in node m include any information. Therefore, no prediction is made when node m receives a snoop request from any of the processors. Under such circumstances, the processor follows the conventional approach and probes the tag array with the requested address. The availability bit (A_i) is set to one if the requested cache block hits in node m's L_1. Otherwise, it is reset. Upon the next snoop request from any processor, if the associated saturating counter is more than the threshold, the predictor is trusted, and the tag array is not accessed if A_i is zero.

For accurate predictions and if any of the $(N-1)$ processors provides the requested data, no additional step is needed. Consequently, power is reduced as only tag arrays more likely to have the requested data are accessed.

Predictors are not always accurate. Misprediction occurs under two scenarios: (1) a requested address missing in the L_1 cache, is speculated to be available and (2) a requested address is available in the L_1 cache, but is speculated to be missing.

[2] We avoid using STL during data write as it could result in violating the cache coherence protocol in the event of mispredicting a sharer node.

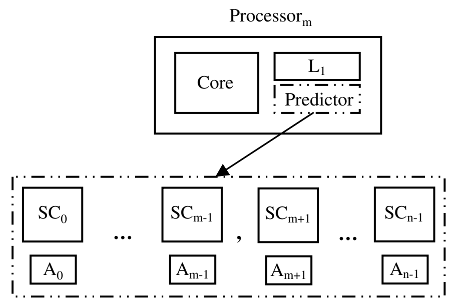


Fig. 6. Processor m using $(N-1)$ entries: each predictor entry includes a saturating counter and an availability bit.

Under the first scenario, and similar to the conventional approach, STL results in unnecessary tag array access but would not violate cache coherence protocol.

Under the second scenario, there are two possibilities: (1) if at least one of the sharers replies to the requester node, power is saved and the state of cache block is changed to shared. (2) If none of the nodes reply to the requester with the requested data, all those nodes that have not accessed their tag arrays due to prediction, should probe their tag arrays. This is necessary as a node, while having the requested block, may have speculatively skipped tag access. This will result in extra power dissipation and latency. Power is increased since interconnect should be accesses twice as the requester node should ask all the nodes that have skipped tag access for snooping. This is possible using a single bit of the bus, as the requester does not need to send the address of the cache block for the second time. As such, this power penalty is negligible. Latency is increased as snoop requests traverse the interconnect for the second time. In addition, some tag arrays are snooped, and finally, the result of snooping is sent back to the requester. We take this latency overhead into account in our simulations.

It is important to note that STL does not impose any changes to the underlying cache coherence protocol. The requested address may be supplied by a cache-to-cache transfer or by the lower memory hierarchy (L_2 cache or memory).

If the requested cache block is available in at least one of the local caches, two cases may happen: (1) at least, one of the sharer processors predicts correctly. (2) All the sharer nodes mispredict. In the former case, the supplier sends data to the requester and the state of cache block in both requester and supplier is changed to the shared state. Note that if more than one node holds the block, the state of cache block is already in shared state in all of the sharer nodes. In the latter case, the requester will not receive the requested data and asks all of the nodes that have skipped snooping to probe their tag arrays. Similar to the baseline CMP, one of the sharer nodes supplies the data and the state of cache block will be shared.

If the lower memory hierarchy turns out to be the supplier, the requester will broadcast the snoop request twice, and none of the remote nodes reply to the requester. Similar to the baseline approach, the requester forwards the requested address to the lower memory hierarchy.

7. Evaluation

In Section 7.1, we present the methodology. In Sections 7.2 and 7.3, we report results for SSR and STL, respectively.

7.1. Methodology

We use SPLASH-2 [4] benchmarks (details reported in Table 1) to evaluate our scheme. Note that some of the benchmarks could not run under SESC as system calls are not completely implemented in this simulator. The cache-to-cache column in Table 1 shows how often L_1 misses are provided by cache to cache transfers. For our simulations, we use Sesc's execution driven mode [7], modeling the out of order processors and the memory subsystem presented in Table 2. We used Orion [21] to estimate interconnects power. Orion uses architectural parameters of interconnects, e.g. length of wire, to estimate dynamic power dissipation. We assume a 1 mm link length and a 180 nm feature size. We use Orion to estimate power in datapath of the switches. We used CACTI [8] to measure tag arrays' dynamic power dissipation. We use the 180 nm technology and one read-write port. CACTI takes as input cache parameters, e.g. cache capacity, cache block size, and the number of read/write ports. It uses analytical models to compute cache latency and energy consumption for different configurations and picks the configuration that has the best latency and energy consumption. Both Orion and CACTI measure dynamic power and do not consider leakage.

We used MESI protocol to maintain cache coherence in L_1 caches [18]. In a typical CMP, both the processor and the bus access the L_1 cache. With only one tag array, if both processor and bus need to access tag array simultaneously, one side has to stall which will degrade the overall performance. To address this problem, and similar to previous work, we use two tag arrays, one for the processor side and one for the bus side [18].

7.2. SSR: Results

In Section 7.2.1, we report SSR accuracy and coverage. In Sections 7.2.2 and 7.2.3, we report how SSR impacts performance and interconnect power. We compare SSR to the conventional baseline cache coherence where snoop requests are broadcasted to all nodes upon any local cache miss. To make better evaluation possible, we also compare SSR 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 SSR 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 2
System parameters

Processor	Interconnect	Memory system
Frequency: 1 GHz	Bus clock cycle: 7 ns	Cache block size: 64 B
Branch predictor: 16k entry	Switch latency: 1 cycle	Split I-L1, D-L1: 32 KB, 4-way
Bimodal and gshare	Link latency: 1 cycle	L1 latency: 2
Branch penalty: 17	Interconnect width: 64 B	L2: 512 KB/8-way
Fetch/issue/commit: 6/4/4		L2 latency: 11
RAS: 32 entries		Memory latency: 70 ns
BTB: 2k entries, 2-way		

7.2.1. Coverage and accuracy

In Fig. 7, we report coverage and accuracy for SSR. Note that in a four-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. Fig. 7a shows coverage for predictors with different sizes. We report for SSR-1, SSR-2, SSR-3 and SSR-4. We use thresholds values equal to zero, two, six, and 14, for SSR-1, SSR-2, SSR-3 and SSR-4, respectively. We picked these thresholds after testing different alternatives. In general, coverage reduces as the counter size increases. On average, coverage varies from 36% to 67% for different counter sizes.

Fig. 7b shows accuracy for predictors with different counter sizes. Accuracy shows how often the predicted supplier turns out to be the correct one. In general, accuracy

Table 1
SPLASH-2 benchmarks and input parameters

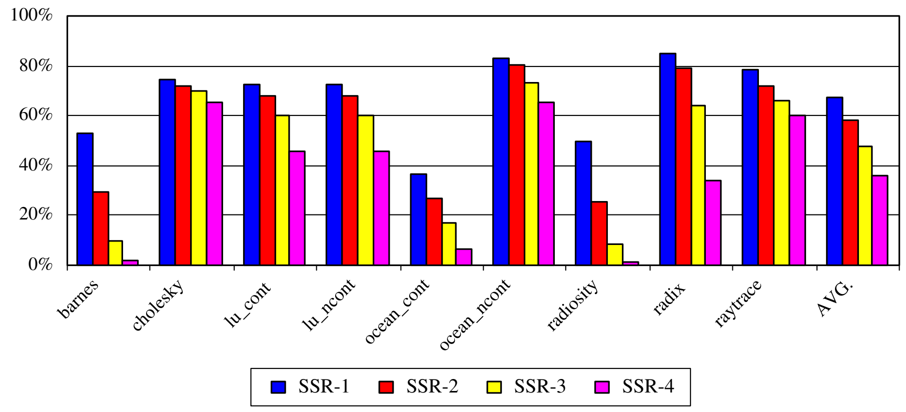
Benchmarks	Input parameters	Cache-to-cache (%)	Benchmarks	Input parameters	Cache-to-cache (%)
Barnes	16k particles	46.1	Ocean (non-contiguous)	258 × 258 grid	64.6
Cholesky	tk29.O	0.3	Radiosity	-batch-room	64.1
Lu (contiguous, non-contiguous)	512 × 512 matrix, $B = 16$	22.1	Radix	8M keys	56.6
Ocean (contiguous)	258 × 258 grid	1.4	Raytrace	Balls4.env	35.7

improves as counter size increases. On average, accuracy changes from 89% to 93% for different counter sizes.

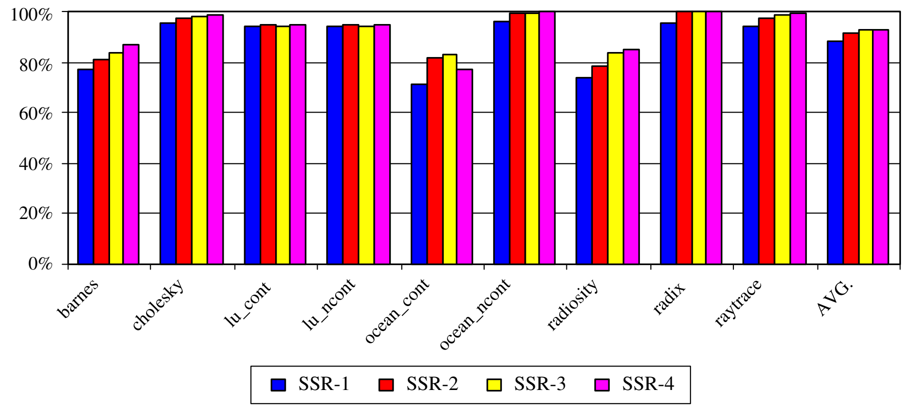
7.2.2. Performance

In this Section, we report performance for SSR and serial snooping [6] compared to the baseline cache coherence protocol. Fig. 8 reports performance for different benchmarks. Numbers less than 1 indicate slowdown.

SSR has negligible impact on performance. For most benchmarks, the impact is less than 0.5%. Note that for some benchmarks (*e.g.*, *ocean_ncontinious*) SSR improves performance. This could be explained by the following: (a) for these benchmarks, often the missing data is pro-



(a) Coverage



(b) Accuracy

Fig. 7. SSR: coverage and accuracy for predictors equipped with 1-, 2-, 3- and 4-bit counters.

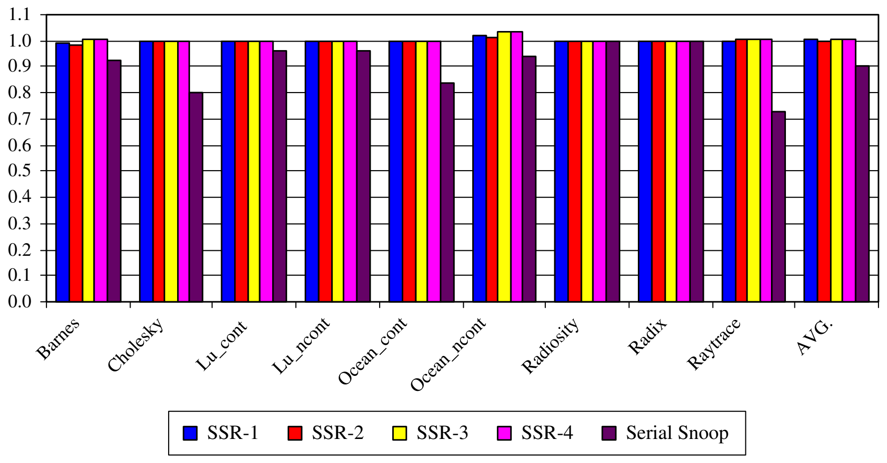
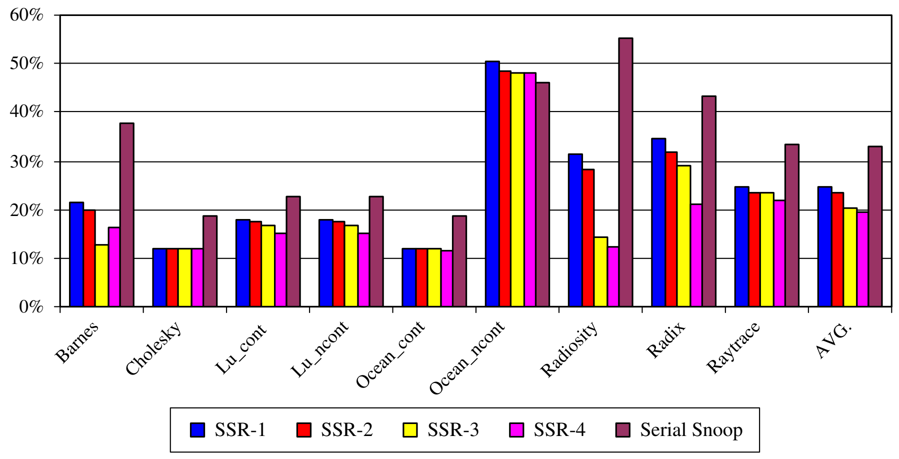


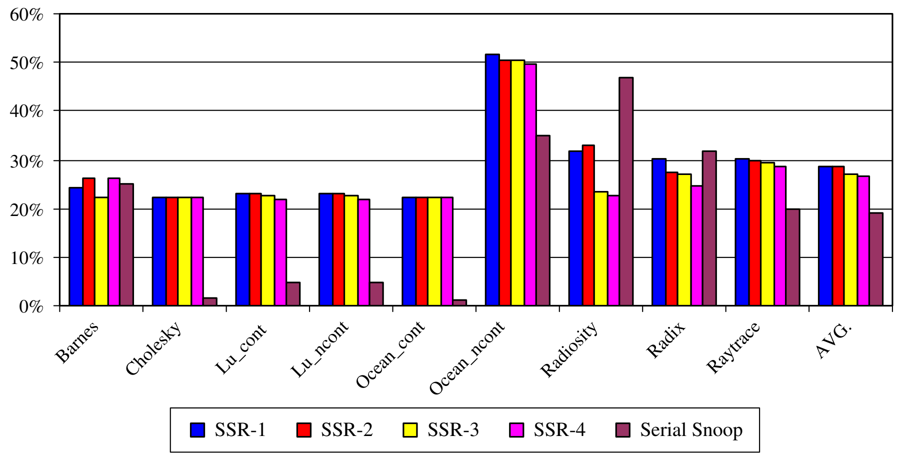
Fig. 8. SSR: performance for different SSR-enhanced processors and serial snooping compared with baseline scheme. Numbers below 1 show slowdown.

vided by on-chip caches rather than the memory and (b) SSR can speculate the supplier caches with high accuracy.

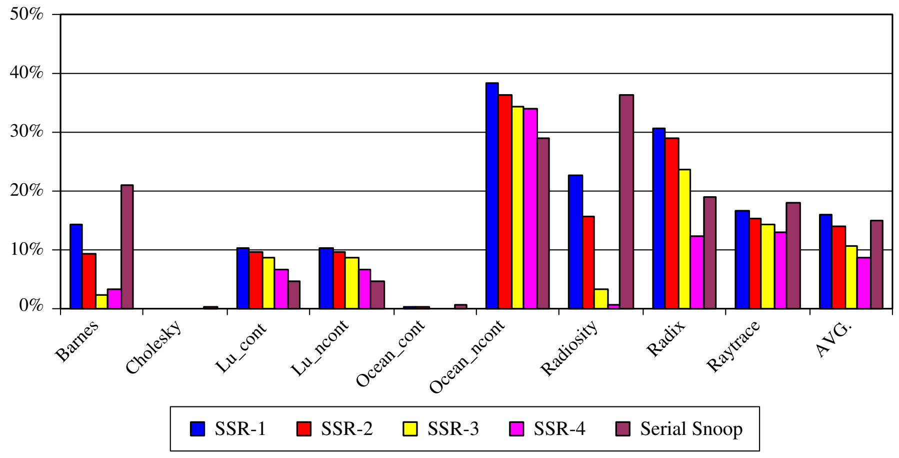
Serial snooping degrades average performance by 9%. In some benchmarks, *e.g.*, *raytrace*, serial snooping degrades performance considerably. In these benchmarks, quite often the supplier is not close to the requestor, and serial snooping increases cache to cache transfer latency.



(a) SSR: Link power reduction.



(b) SSR: switch power reduction.



(c) SSR: tag array power reduction

Fig. 9. Power reduction in links, switches, and tag arrays.

7.2.3. Power reduction

In Fig. 9, we report power reduction in links, switches, and tag arrays for SSR and serial snoop compared to the baseline cache coherence scenario. Generally, power reduction improves as counter size decreases. This is intuitive as bigger counters have lower coverage.

On average, as reported in Fig. 9a, SSR-1, SSR-2, SSR-3 and SSR-4 reduce link power by 25%, 24%, 21%, and 19%, respectively. Serial snooping reduces link power by 33%. However, this comes with significant increase in run time.

On average, as reported in Fig. 9b, SSR-1, SSR-2, SSR-3 and SSR-4 reduce interconnect switch power by 29%, 29%, 27% and 27%, respectively. Serial snooping reduces switch power by 19%. In four benchmarks, SSR reduces switch power 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 Fig. 1, 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 SSR.

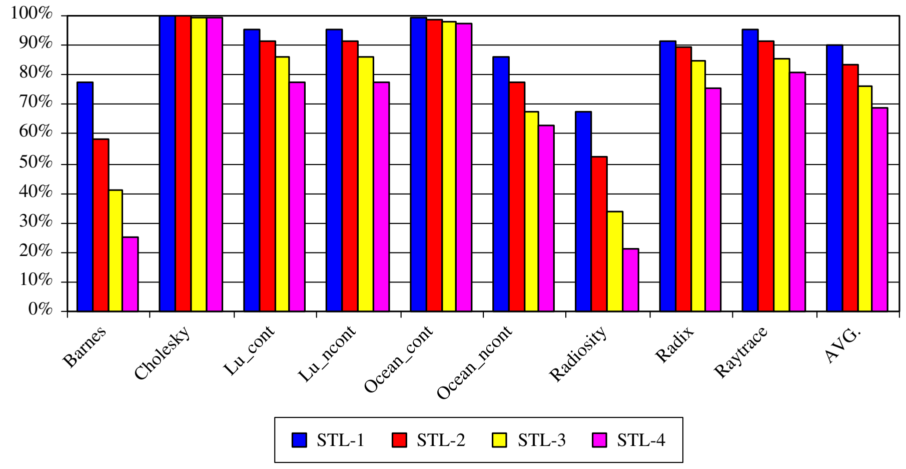
The two methods improve tag array power competitively (see Fig. 9c). SSR-1, SSR-2, SSR-3 and SSR-4 improve tag array accesses by 16%, 14%, 11% and 9%, respectively. Serial snooping improves tag array power by 15%.

Note that tag array power reduction is zero for *Cholesky* and *Ocean_cont*. Our study shows that cache to cache transfers occur rarely in these benchmarks (see Table 1). As such, despite high accuracy, SSR does not improve tag array power. However, *Cholesky* and *Ocean_cont* show power reduction in links and switches as the result of the cache coherence step reduction explained in Section 5.2.

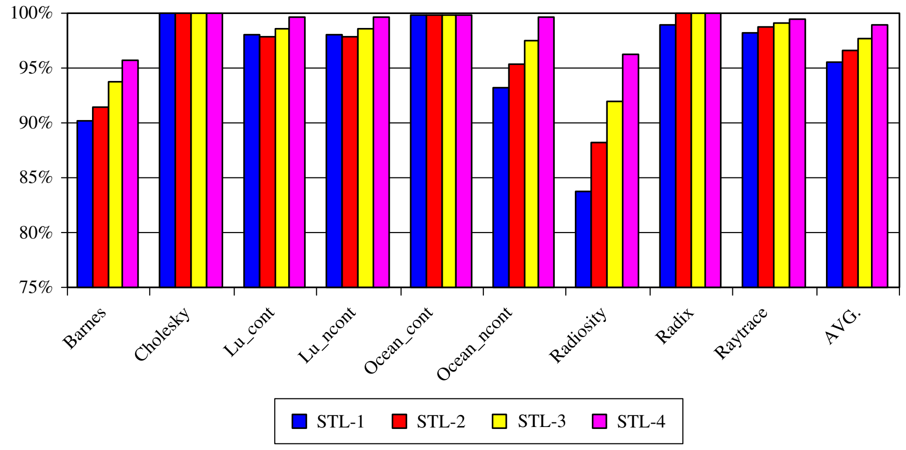
Overall, SSR-1 provides substantial power savings with negligible performance degradation, and minimal hardware overhead.

7.3. STL: Results

In Section 7.3.1, we report STL accuracy and coverage. In Section 7.3.2, we report how STL impacts performance. We report power saving in tag arrays in Section 7.3.3.



(a) STL: Coverage



(b) STL: Accuracy

Fig. 10. Predictor coverage and accuracy for STL-1, STL-2, STL-3, and STL-4.

7.3.1. Coverage and accuracy

In Fig. 10, we report coverage and accuracy for STL-1, STL-2, STL-3, and STL-4. We use thresholds values equal to zero, two, six, and 14, for the four configurations, respectively. Note that in a four-way multiprocessor there are four predictors. We report average data for the four predictors.

We define coverage as the percentage of missed cache snoops that are accurately identified by predictors. Fig. 10a shows that coverage reduces as the counter size increases. On average, coverage varies from 69% to 90% for different counter sizes.

Fig. 10b shows accuracy for predictors with different counter sizes. Accuracy shows how often the predictors speculate correctly. In general, accuracy improves as counter size increases. On average, accuracy changes from 96% to 99% for different counter sizes.

7.3.2. Performance

In Fig. 11, we report performance for STL compared to the baseline cache snoop scheme. Numbers less than one show performance slowdown.

STL has negligible impact on performance. On average, performance changes from 0.33% to 0.36% for different counter sizes.

7.3.3. Power reduction

In Fig. 12, we report power reduction in tag arrays for STL-1, STL-2, STL-3, and STL-4 compared to the baseline cache snoop scenario.

On average, STL-1, STL-2, STL-3, and STL-4 reduce power in tag arrays by 14.1%, 12.6%, 11.5%, and 11%, respectively.

Generally, power reductions depend on how often the missing data is provided by cache to cache transfers. In *Cholesky* and *Ocean_cont*, L_2 and memory often provide the missing data (see Table 1). For these benchmarks, power saving is negligible.

In *Radiosity*, power savings decrease rapidly as the size of counters increases. This is consistent with the rapid coverage reduction observed for this benchmark when counter size varies (Fig. 10a).

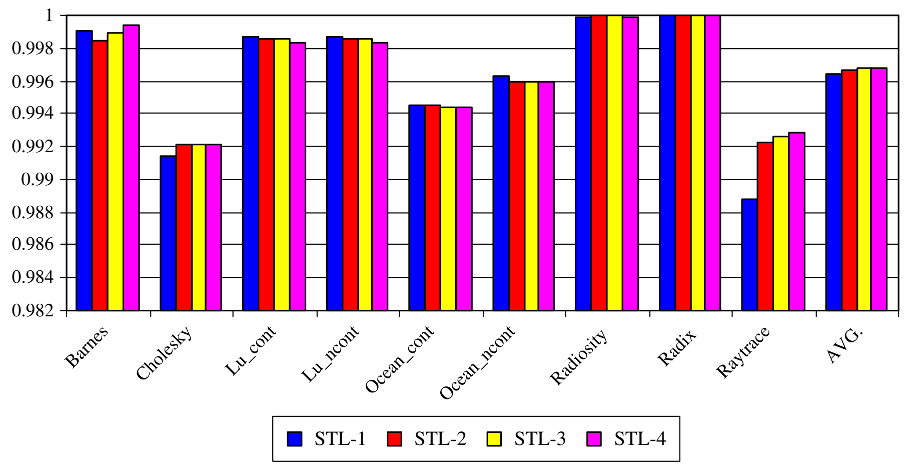


Fig. 11. STL: relative performance for STL-1, STL-2, STL-3, and STL-4.

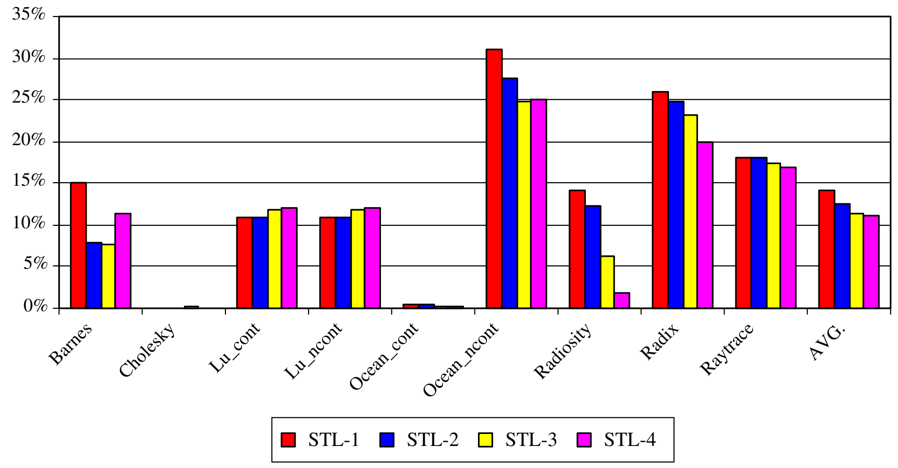


Fig. 12. STL: tag array power reduction.

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

8. Conclusion

In this work, we used supplier locality to reduce power dissipation induced by snoopy cache coherence protocol in interconnects and tag arrays. In contrast to previous studies [1,3,12–14,17] that used cache block address for prediction, our scheme comes with low hardware overhead for prediction. We used our findings and proposed two power aware techniques for CMPs.

First, we introduced SSR as a prediction-based cache coherence protocol to speculate the supplier processor at the requester end. By using a low overhead predictor, requests are sent directly to the predicted 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 power reduction. SSR reduces power of links, switches, and tag arrays by 25%, 29%, 16%, respectively.

Second, we used supplier locality at the supplier end and introduced STL to skip unnecessary snoop induced tag lookups. STL uses small yet accurate predictors. These predictors are located on the interconnect side of the L1 cache. On average, STL reduces tag array power by 14.1% in a four-way CMP.

References

[1] J. Nilsson, A. Landin, P. Stenström, Coherence predictor cache: a resource efficient coherence message prediction infrastructure, in: Proceedings of the Sixth IEEE International Symposium on Parallel and Distributed Processing Symposium, 2003.
[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, J. Duato, The use of prediction for accelerating upgrade misses in CCNUMA multiprocessors, in: Proceedings of the PACT-11, 2002.
[4] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, 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 51 (5) (2004) 800–849.
[6] C. Saldanha, M.H. Lipasti, Power Efficient Cache Coherence, High Performance Memory Systems, Springer-Verlag, 2003.
[7] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, K. Strauss, S. Sarangi, P. Sack, P. Montesinos. SESC Simulator, January 2005. http://sesc.sourceforge.net.
[8] P. Shivakumar, N. Jouppi, CACTI 3.0: an integrated cache timing, power and area model, Technical Report 2001/2, Compaq Computer Corporation, August 2001.
[9] Shubhendu S. Mukherjee, Mark D. Hill, Using prediction to accelerate coherence protocols, in: Proceedings of the ISCA-25, June 1998.
[10] Tse-Yuh Yeh, Yale Patt, Alternative implementations of two-level adaptive branch prediction, in: Proceedings of the ISCA-19, May 1992.
[11] A.-C. Lai, B. Falsafi, Memory sharing predictor: the key to a speculative coherent DSM, in: Proceedings of the ISCA- 26, 1999, pp. 172–183.
[12] M.E. Acacio, J. González, J.M. García, J. Duato, Owner prediction for accelerating cache-to-cache transfers in a cc-NUMA architecture, in: Proceedings of the SC2002, November 2002.
[13] A. Moshovos, B. Falsafi, A. Choudhary, JETTY: filtering snoops for reduced energy consumption in SMP servers, in: Proceedings of the HPCA-7, January 2001.
[14] J. Cantin, A. Moshovos, M. Lipasti, J. Smith, B. Falsafi, Coarse-grain coherence tracking: regionscout and region coherence arrays, IEEE Micro 26 (1) (2006) 70–79.
[15] M. Ekman, F. Dahlgren, P. Stenström, TLB and snoop energy-reduction using virtual caches for low-power chip-multiprocessors, in: Proceedings of the ACM International Symposium on Low Power Electronics and Design, 2002.
[16] M. Ekman, F. Dahlgren, P. Stenström, Evaluation of snoop-energy reduction techniques for chip-multiprocessors, in: Proceedings of the First Workshop on Duplicating, Deconstructing, and Debunking, May 2002.
[17] Milo M.K. Martin, Pacia J. Harper, Daniel J. Sorin, Mark D. Hill, David A. Wood, Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors, in: Proceedings of the ISCA-30, 2003, pp. 206–217.
[18] D.E. Culler, J. Singh, A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufman Publishers, San Francisco, CA, 1999.
[19] M. Bjorkman, F. Dahlgren, 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, January 1995.
[20] A. Landin, E. Hagersten, S. Haridi, Race-free interconnection networks and multiprocessor consistency, in: Proceedings of the 18th International Symposium on Comp. Architecture, 1991.
[21] H.S. Wang, X.P. Zhu, L.S. Peh, S. Malik, Orion: a power-performance simulator for interconnection networks, in: International Symposium on Microarchitecture, 2002.

Ehsan Atoofian is a Ph.D. candidate in electrical and computer engineering at the University of Victoria. His research interests include computer architecture with emphasis on multiprocessors, high speed memory systems, and embedded systems. He received B.S. and M.S. degrees in computer engineering from the University of Tehran, Tehran, Iran, in 2000 and 2003, respectively.

Amirali Baniasadi received his B.S. degree in electronics and electrical engineering from Tehran University, Tehran, Iran, in 1993. He received his M.S. degree in digital electronics from Sharif University of Technology, Tehran, Iran in 1995. He received his Ph.D. degree in computer engineering from Northwestern University, Evanston, IL, USA in 2002. He is currently an assistant professor at the ECE department of University of Victoria, Victoria, BC. His major research interest is finding new ways to design modern processors. In particular, he has been working on developing microarchitectural techniques to reduce power dissipation in modern processors.