SARC Coherence: Scaling Directory Cache Coherence in Performance and Power

THE SARC PROJECT SEEKS TO IMPROVE POWER SCALABILITY OF SHARED-MEMORY CHIP MULTIPROCESSORS (CMPS) BY MAKING DIRECTORY COHERENCE MORE EFFICIENT IN BOTH POWER AND PERFORMANCE. THE AUTHORS DESCRIBE HOW THEY ELIMINATE TWO MAJOR SOURCES OF INEFFICIENCY FOR DIRECTORY COHERENCE PROTOCOLS: INVALIDATION TRAFFIC ON WRITES AND DIRECTORY INDIRECTION FOR FINDING THE WRITER.

Stefanos Kaxiras
Uppsala University, Sweden

Georgios Keramidas
Industrial Systems Institute, Greece

• • • • • • To scale application performance in the multicore era, we must go beyond instruction-level parallelism (ILP) and instead rely on explicit parallelism. An important issue for advances in this direction is ease of parallel programming. To this end, the shared-memory programming model offers a good starting point. However, we cannot ignore the issue of power efficiency. Poor power efficiency—increasing power for diminishing performance gains—killed the development of wider ILP architectures. This danger is also visible in multicores—that is, when a parallel application experiences sublinear speedup and/or when its power consumption increases faster than the number of cores allocated to it.

Hill describes various notions of scalability.[1] For multicores, it is useful to think of the scalability of a parallel program in terms of power performance. In this article, we use the energy-delay product (EDP) to study the scalability of parallel programs. Three forces shape a parallel application's power efficiency (EDP):

- the application's performance scalability (speedup),
- the application's communication-to-computation growth rate, and
- the working set size and how it fits in the core caches.[2]

We focus on improving power efficiency for parallel applications in a shared-memory CMPs. The SARC project (http://www.sarc-ip.org) provides the framework. Our approach uses tear-off cache blocks (blocks that are not registered in the directory but self-invalidate on synchronization) to eliminate invalidation traffic, reduce false sharing, and upgrade traffic, and writer prediction to eliminate directory indirection and go to the writers directly. We thus achieve both power (in the network and caches) and performance (for reads and writes) benefits. We evaluate our approach using Gems and show significant improvements in EDP over a base MESI protocol—improvements that increase with core count.

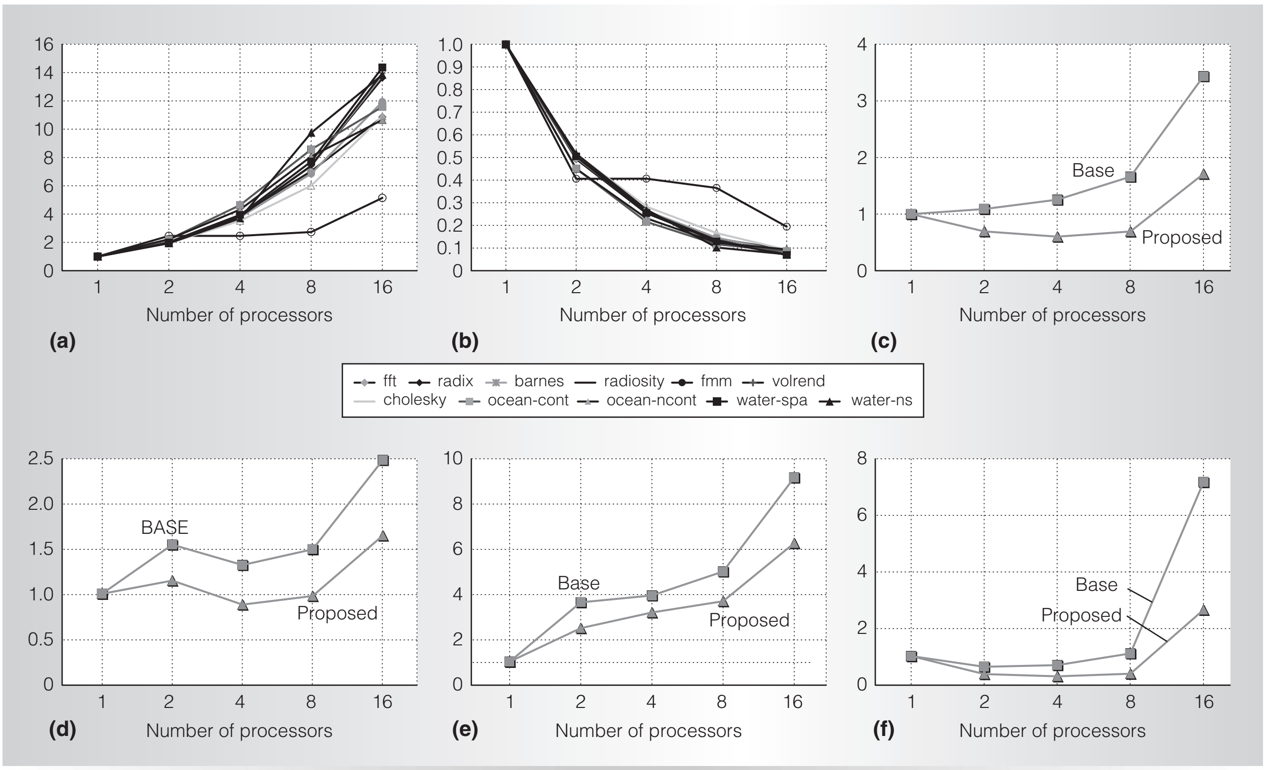


Figure 1. Speedup (a), normalized core energy-delay product (EDP) (b), normalized network and cache EDP averaged over all Splash-2 benchmarks (c) and normalized network & cache EDP for fft (d), radix (e), and ocean non-cont (f).

Energy-delay product scalability in shared memory CMPs

The SARC architecture assumes a heterogeneous multicore processor composed of general-purpose cores and application-specific accelerators.[3] General-purpose cores have a private cache hierarchy that we call level-one (L1) (although it might consist of more than one level) and a last-level cache before the external memory, called level-two (L2). General-purpose parallel programs, such as the Splash-2 benchmarks, run on the general-purpose cores. All on-chip communication is point-to-point messaging via a network on chip (NoC). A directory-based coherence protocol keeps distributed private (L1) caches coherent.[4] The directory is colocated with the shared on-chip L2 cache—that is, the directory tracks only the on-chip cache blocks.

EDP scalability depends on the behavior of core, cache, and network power. Core power, at first approximation, is fairly stable per core and increases about linearly with the number of cores. This leads to poor power efficiency if we use more cores for relatively less speedup. Figure 1 shows the speedups achieved in our simulations of a 16-core CMP for Splash-2 benchmarks (we discuss the evaluation setup later in the article). Because core EDP can be derived from the performance scalability and average core power, we can concentrate instead on the more interesting problems of the cache and network EDP scalability:

- Network power is an interplay between the capacity-miss traffic at low core counts and the increased coherence-communication traffic at higher core counts. Furthermore, network power is also affected by the distances the messages travel. As we use larger networks, the energy spent per message increases.
- Cache power is also directly related to coherence. A significant source of inefficiency is the directory itself—specifically, the directory's central role in all coherence operations necessitates costly

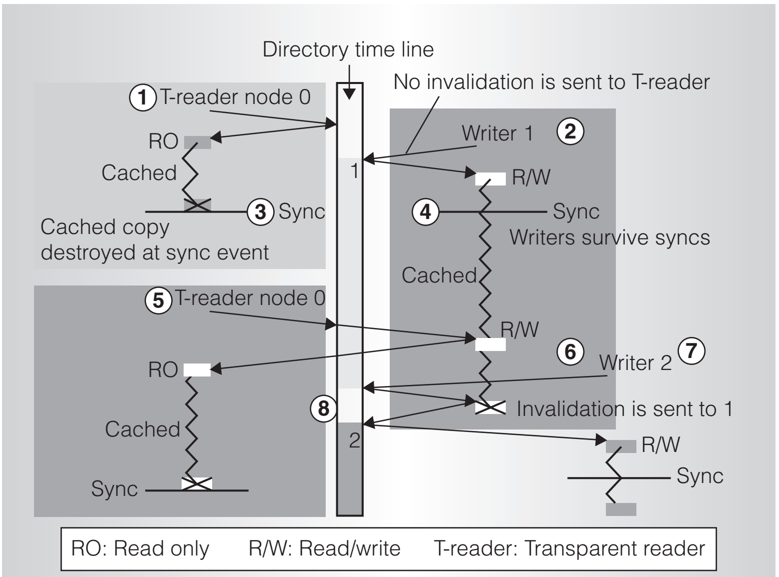

Directory time line
No invalidation is sent to T-reader
1 T-reader node 0
RO
Cached
3 Sync
Cached copy destroyed at sync event
Writer 1 2
R/W
4 Sync
Writers survive syncs
Cached
5 T-reader node 0
R/W
RO
6 Writer 2 7
8 Invalidation is sent to 1
Cached
R/W
Sync
Sync
1
2
RO: Read only R/W: Read/write T-reader: Transparent reader

Figure 2. Basic operations with transparent reads and tear-off copies (tear-off, read-only [TRO] protocol).

(in power and latency) indirections through it.

- Directory coherence, due to the large number of messages it sends and the large number of cache (directory) accesses it causes, eventually does not scale well in terms of power and performance (EDP). Indeed, as Figure 1 shows, cache-and-network EDP scalability suffers significantly as we run the Splash-2 benchmarks on an increasing number of cores.

To address these problems, we combine the use of tear-off cache blocks and writer prediction. This combination is important because tear-off blocks make going to the directory optional. This greatly simplifies writer prediction, which can be built on top of a tear-off protocol practically for free: guess a writer, go to it directly; if it is indeed the writer, simply create a tear-off copy. The combined protocol is minimal in its messaging and requires minimal hardware support over a standard MESI protocol.

Furthermore, we use frugal (very power-efficient) predictors for the writer prediction. We use instruction-based prediction with a small PC-indexed predictor (near the L1) and fall back to simple address-based prediction when needed. Our predictors provide the necessary accuracy and improve EDP by as much as 82.62 percent in 16 cores. Furthermore, the EDP improvement in SARC coherence is not a trade-off between energy and delay, but rather an improvement in both.

Transparent reads and tear-off copies

Transparent reads are reads that do not register in the directory and create a tear-off, read-only (TRO) copy.[5] The TRO copy is thrown away (self-invalidates) at the first synchronization event experienced by the core that issued the transparent read. Figure 2 shows an example. In the figure, the node on the left transparently reads a clean block (1) by going to the L2/directory. A transparent read leaves no trace in the directory but gets a clean cache block in read-only mode (RO); its state is designated *tear-off* or TRO. A writer (on the right of Figure 2) goes to the directory (2) and obtains the block with read/write permissions (R/W). The writer cannot invalidate the reader because the latter did not register in the directory. However, the transparent reader is bound to throw away the TRO block at the first synchronization point (3). The writer, on the other hand, registered in the directory, does not throw away its copy (4). The self-invalidated reader sends a new transparent read (5), which, after reaching the directory, proceeds to fetch the latest value from the writer (6). A second writer (7), invalidates the previous one (8) and registers its own ID in the directory.

With transparent reads, coherence in the traditional sense—that is, sending invalidations upon a write, is maintained only among writers. We call this *writer coherence*. Writers must still register in the directory so the latest value of the cache block can be safely tracked. A new write simply invalidates the previous writer of the block (typically, there are no readers registered in the directory). A significant difference between our work and prior work[5] is that the writer does not downgrade with a transparent read. This significantly reduces upgrade traffic. Normal reads still downgrade a writer from read/write permissions to read-only. The downgraded copy, however, is not a

TRO and is registered in the directory, which means that it will be invalidated by the next writer. Lastly, because the invalidation protocol is always active underneath the transparent read and tear-off mechanisms, we can freely mix invalidation and tear-off copies, simultaneously, in the same directory entry.

Correctness with tear-off cache copies requires that programs be written as if for a weak-ordered memory system, which means that tear-off copies require correctly synchronized programs.[5,6] As long as synchronization in the program is clearly identified and exposed to the hardware (to purge the TRO copies), programs run correctly. TRO copies are incompatible with data races such as those used for flag synchronization (flag synchronization based on ordinary reads and writes) because reads might completely miss all writes, in the absence of any synchronization among them. Banning such code practices, however, might ultimately lead to safer code. Where needed, replacing flag synchronization with semaphore synchronization corrects the problem. In our case, the Splash-2 benchmarks run unmodified, without glitch, on our protocol by simply making the synchronization primitives (atomic instructions and lock releases) visible to the hardware for the purpose of self-invalidating the TRO copies.

Transparent reads and tear-off copies have both performance and power implications. In terms of performance, writes become faster because reader invalidation (and its acknowledgment) is no longer required; in addition, upgrade traffic is reduced. When transparent reads and tear-off copies are coupled with a weakly ordered consistency model, the improvement of write performance does not contribute significantly to the overall performance. In large part, weak consistency models hide write latency,[6] so its reduction is unimportant in this context. Secondary performance benefits come from removing the invalidation and upgrade traffic from the network, reducing congestion and average message latency. On the other hand, significant power benefits come from removing much of the invalidation traffic as well as upgrade traffic.

A possible pitfall when using transparent reads is to needlessly discard at synchronization copies that do not change (read-mostly data). This could negatively impact both performance and power by constantly refetching such copies after synchronization events. To guard against this, we can use the directory to classify whether a cache block is read-mostly or frequently written. TRO copies are appropriate for frequently written lines (because self-invalidation would replace the frequent invalidations), whereas invalidation copies are most appropriate for read-mostly lines because they are rarely invalidated and can stay live in the caches for as long as needed. In our case, the tell-tale sign for read-mostly TRO copies is that the directory sees the same nodes repeating their transparent reads for the same line without an intervening write. Lebeck and Wood proposed a similar but distributed adaptation for tear-off copies based on block versioning, in which the nodes themselves decide whether to request TRO copies.[5] We implemented our adaptation mechanism in the directory and, although it adds marginal benefit, it also increases the directory's complexity and cost. For this reason, in the rest of this article, we omit directory adaptation in the interest of clarity.

Writer prediction: Avoiding directory indirection

Although transparent reads and tear-off copies reduce the invalidation traffic, another significant source of inefficiency remains in directory protocols: the indirection through the directory. The directory (whether centralized or distributed) is a fixed point of reference to locate and obtain the latest version of the data from the writer or invalidate its sharers. It is difficult and rather complex to avoid an indirection via the directory,[7] which leads to the classic three- or four-hop invalidation protocols, shown in Figures 3a and 3b.

Our main contribution is in exploiting the properties of transparent reads and tear-off copies, and, with a simple and minimal protocol, go directly to the writers (by predicting their identity), skipping the directory when possible. In essence, we let the writers assume the role of the directory (the central role of coherence) but revert back to

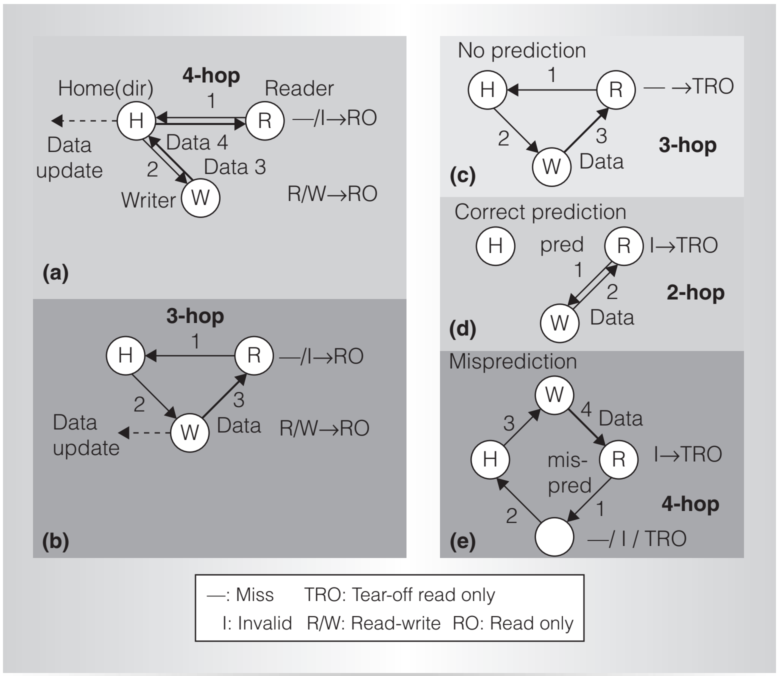


Figure 3. Writer prediction aims to avoid the indirection via the directory (when possible) and go directly to the writer. Directory indirection protocols are either three-hop (a) or four-hop (b), depending on whether the data returns to the directory. Writer prediction protocols: no prediction (c); correct prediction (d); and misprediction (e).

directory indirection (the safety net) when we cannot locate the writers. We can freely intermix the TRO and the TRO enhanced with writer-prediction with the base invalidation protocol at any time and for any cache block.

Reads

In directory protocols, a read goes to the directory to find the location of the latest version of the data. The last writer then forwards the correct data using a three-hop protocol (Figure 3b) or sends it back via the directory with a four-hop protocol (Figure 3a). TRO copies, however, let the reads avoid going to the directory altogether. Rather, reads try to obtain the data directly from the writer if they can locate it.

We use prediction to locate the current writer. Based on the history, a reader sends a direct request to a predicted writer (Figure 3d). If this node has write permissions (read/write) for the requested cache block, the prediction is correct: the node is the (one and only) writer of the block and has the latest copy of the data. If the node is in any other state (including knowing nothing about the block), it bumps the request to the directory (Figure 3e). From there, the request is routed to the correct writer and back to the reader (as in a normal three-hop protocol). The penalty for a misprediction is one extra message (indirection via the wrong node).

Avoiding directory indirection for reads is important in two ways. First, reads, in contrast to writes, are performance-critical, meaning a reduction of their latency directly reflects on overall performance. Second, directory indirection accounts for a significant part of the read traffic. Eliminating it immediately impacts network power and performance. Writer prediction yields a two-hop transaction when correct, or a four-hop transaction on a misprediction. A simple calculation reveals that any prediction accuracy over 50 percent yields both performance and power benefits.

Writes/upgrades

Under an invalidation protocol, a (new) writer sends its request to the directory, which either forwards the request to the previous writer if one exists (Figure 4a), or invalidates multiple readers (Figure 4b). In our case, the new writer predicts the previous writer and sends a direct request to it. Figures 4c, 4d, and 4e show the sequence of hops for the base three-hop invalidation as well as for correct prediction and misprediction.

- *No prediction* (Figure 4c). In the base three-hop invalidation, data from the old writer directly transfers to the new writer along with read/write permissions. The previous writer returns its acknowledgement to the directory (3b). This is a three-hop-latency protocol with an additional overlapping hop (3a and 3b overlap), which we designate as a (3+1)-hop protocol.
- *Correct prediction* (Figure 4d). A direct request from the new writer arrives at the predicted node. If the predicted node has read/write privileges, it is the block's writer. It returns the data to the requester, passing along its read/write privileges. It also informs the directory

that it has relinquished its rights to the new writer and has self-invalidated. This is a (2+1)-hop protocol because it overlaps the last two messages. The previous writer's acknowledgment to the directory (message 2b) carries the new writer's identity and plays the same role as acknowledgment (message 3b) in the no-prediction case.

- *Misprediction* (Figure 4e). The incorrectly routed request bumps-off the wrong node, which is not the block's writer, and is rerouted to the directory. The penalty in this case is an extra request message indirection via the wrong node, resulting in a (4+1)-hop protocol.

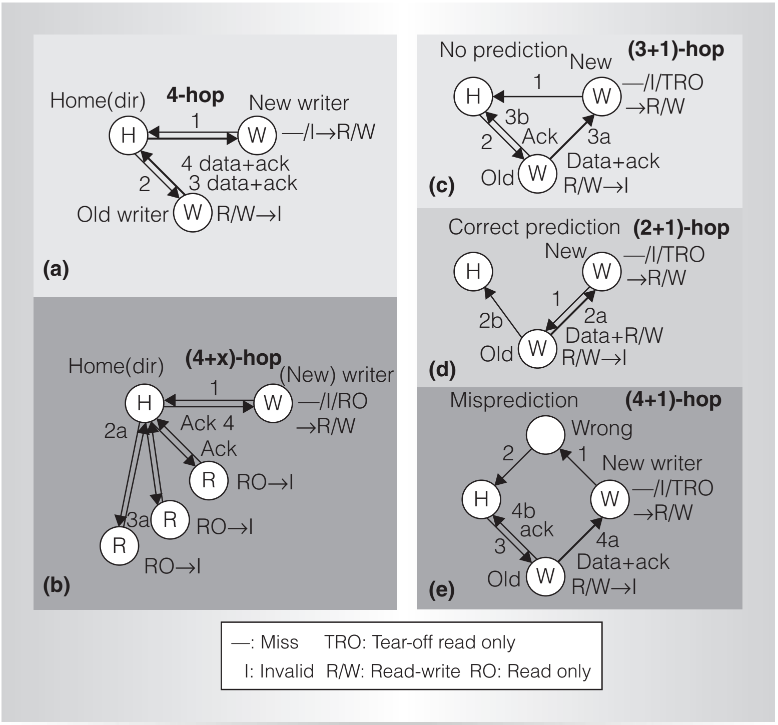


Figure 4. Writer prediction on writes and upgrades: Four-hop invalidation of a single writer (a), four-hop invalidation of multiple readers when no writer exists (b), base three-hop invalidation when there is no prediction (c), correct prediction (d), and misprediction (e).

The main idea here is that the previous writer assumes the directory role, passing out its read/write privileges. Similarly to the writer prediction for reads, when the prediction is correct, the predicted node indeed has write permissions to the cache block and returns data and read/write privileges to the requestor. The previous writer also notifies the directory of the new writer's identity and invalidates its local copy. This notification from the previous writer to the directory is the basis for resolving possible races among (new) writers, some of which might use prediction while others might go directly to the directory. We carefully checked the writer prediction's correctness for writes and upgrades and have emphasized resolving writer races safely.[8]

Correct writer prediction reduces the message count from four messages to three because it coalesces the two directory-indirection messages into one direct message to the writer. In addition to the power benefit of eliminating a message, there is a performance benefit because the critical latency further drops from three hops to two hops—hence, a (2+1)-hop protocol. However, a misprediction results in a four-hop, five-message protocol with negative impact on both power and performance. A simple calculation again reveals that a prediction accuracy of more than 50 percent starts to yield benefits.

Although simple predictor schemes can achieve an accuracy of 50 percent or greater, writer prediction is futile in certain situations. Writer prediction in migratory sharing or unstructured sharing cannot, in any useful sense, be performed using history information. In this case, we could adapt the directory to prevent writer prediction for migratory data.[9,10] We do not use migratory sharing classification in our evaluation to avoid the increased cost and complexity and keep our approach simple. In this respect, our results without it are conservative. This classification is not the same as the migratory optimization—that is, collapsing a read-miss and a write-miss into one[9-11]—but simply prevents writer prediction in hopeless cases. The migratory optimization is orthogonal to our approach and can be easily incorporated in the same instruction-based predictor for additional performance and power benefits.

Implementation

Implementing transparent reads requires minimal changes in the cores, protocol engines, and caches.[5] Protocol logic changes

Table 1. Simulator configuration.

Parameter	Chip configuration
Processor	16 cores
Cores	3 GHz in-order, single-issue, blocking model (Simics)
Block size	64 bytes
Data and instruction L1 caches	256 Kbytes, 8-way, 3-cycle latency, Pseudo-least recently used (LRU) (32 Kbytes, 64 Kbytes, 256 Kbytes, 512 Kbytes, 1 Mbyte for sensitivity studies[8,15])
Shared L2 cache	8-Mbyte NUCA, 16-way, 4 banks, 15-cycle latency for tag accesses and 30-cycle latency for data accesses, P-LRU, each L2 bank is connected to the four edge routers of the 2D mesh network
Memory	4 Gbytes, 256-cycle latency
Interconnection network	2D mesh topology, 2-cycle link latency, 16 bytes flit size

Table 2. Benchmarks used in our evaluation.

Benchmark	Input
fft	64 K complex doubles
radix	2 M keys
barnes	8 K bodies, 4 time steps
radiosity	Room
fmm	8 K particles
volrend	Head
cholesky	tk29.0
ocean non-contig.	258 × 258 grid
ocean contig.	258 × 258 grid
water-SP	512 molecules, 3 time steps
water-Nsq	512 molecules, 3 time steps

and modifications to MESI Dir$_i$NB[12] or DASH-like directory protocols are simple.[4] Discarding the TRO copies upon synchronization requires some support in the caches that is similar to but simpler than hierarchical decay counters, which minimally impact power consumption.[13]

Prior work has shown that instruction-based prediction can efficiently capture the access behavior of programs and relate it to a small set of PCs.[11] This efficiency is due to the fact that, unlike address prediction, which is based on data behavior, the behavior of the code is constant over time and can be learned quickly.[11,14] In contrast to other work that aims to predict the destination set of readers,[14] we predict the writer.

The predictor is a small structure indexed and tagged by the PC of instructions causing misses in the L1. By using just 64 entries (8-way set associative with least-recently used replacement), we can capture 99 percent of the benefit of a predictor of any larger size. Each entry holds the predicted writer and a 2-bit saturating confidence counter. A side buffer holds the miss address and the PC for updating the predictor when a miss is satisfied. We implement this side structure with minimal cost as part of the L1 MSHRs.

Although the instruction-based predictor works well for instructions causing read and write misses, its performance for stores that cause upgrade misses lags behind. The reason is that a read miss precedes each upgrade miss. Because the predictor is updated when misses are satisfied, writer information in this case is related to the load that caused the read miss. By the time the store that caused the upgrade miss is updated, the writer information is not available. To solve this problem, we rely on a simple address-based prediction, which is carried by the TRO cache blocks. Each TRO copy carries the ID of the last known writer—that is, the node from which it got the data. The overhead for a 16-core CMP is just 4 bits per cache line, which is negligible. The performance of this simple scheme is enough to give stores a prediction accuracy comparable to—or better than—the loads.

Evaluation

We implemented our protocols using Gems on top of Vitutech Simics (http://www.virtutech.com). We configured Simics to simulate a 16-core SPARC running Solaris 10, and configured Gems/Ruby to model a CMP with a mesh interconnect. Table 1 shows the simulator parameters and Table 2 shows the Splash-2 benchmarks with their inputs. As Figure 1 shows, all of the benchmarks show relatively good speedups up to 16 cores, except for radiosity, due to its small input (larger inputs for radiosity lead to long simulation times). We use full-run simulations (from start to completion of the parallel part of the application) to correctly compute EDP. Relying on a small part of the execution (for example, a

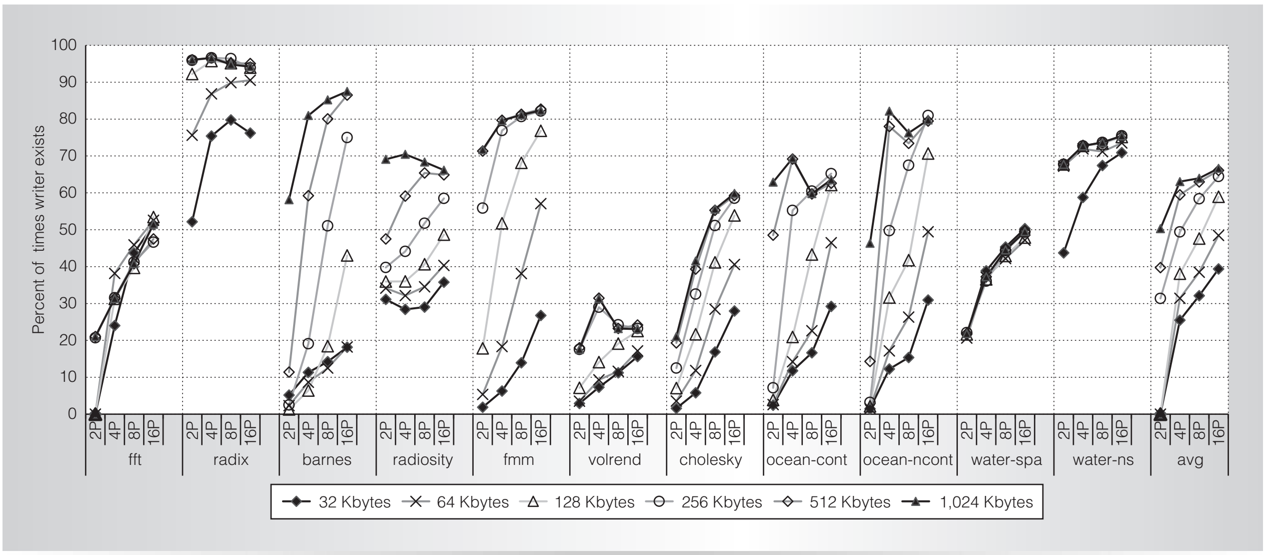


Figure 5. Potential benefit from writer prediction. The benefit largely depends on how much cache space is available.

fixed number of instructions) cannot yield reliable EDP results because instruction per cycle (IPC) is not the correct metric for CMPs. We concentrate on cache and network EDP because they are critical for scaling the coherence protocol. Individual core power (which we can compute using partial Gems/Opal runs) is roughly constant per benchmark, regardless of the core count used to run the benchmark. Thus, core EDP largely follows the speedup curves, and so is not interesting in our study.

We rely on Orion 2.0 to compute network energy and Cacti 6.5 to compute cache energy. We used a 45-nanometer process technology for both Orion and Cacti. We derived all process-specific parameters used by Cacti 6.5 from the ITRS roadmap, and modeled the caches in Cacti using the parameters in Table 1. We also modeled the instruction-based predictor in Cacti as an 8-way 64-entry cache (2-byte tags and 1-byte prediction data). Our Cacti simulations reveal that the instruction-based predictor energy per access is 10.06 percent of the energy per accesses consumed by a 256-Kbyte L1 cache (12.12 percent of a 32-Kbyte cache, 1.63 percent of the 8-Mbyte L2 cache, and 8.2 percent of a router). The extra power introduced by our instruction-based predictor is more than sufficiently compensated by reducing the data and control messages as well as by substituting L2 bank accesses with L1 accesses. Finally, we assume that the network link width for L1-to-router communication and for router-to-router communication is 16 bytes (the same as the flit size), while the network link width for L2-directory-to-router communication is 2 bytes.

Oracle writer prediction

To determine writer prediction's potential, we explore an oracle predictor with 100 percent accuracy. Its coverage gives the number of times a writer actually exists in a local cache and can supply the data with a direct transfer to a requestor. For the misses not covered by the oracle predictor, no writer exists in the system at the time of the miss; data must be fetched by going to the directory (L2). The oracle predictor simply peeks at the directory to see who the writer is (the node holding the specific cache line in exclusive/modified state) whenever a prediction opportunity exists.

Figure 5 shows the potential benefit across several cores (from 2 to 16) and for a wide range of L1 cache sizes (from 32 Kbytes to 1 Mbyte) for the Splash-2 benchmarks. The potential benefit is significant in many of the benchmarks, but varies significantly within a benchmark in relation to core count and cache size. The potential benefit largely depends on the total amount of cache space available. Increasing either the core count or the cache size increases the total amount of L1 cache space available to the application. With more cache space, writers remain longer in the L1 caches and can satisfy direct requests.

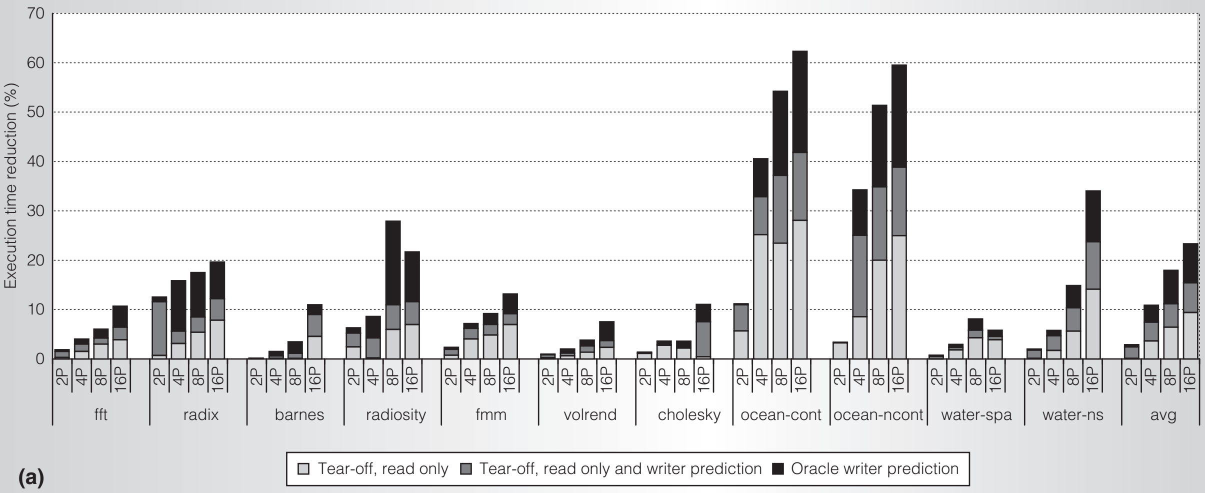
Results

Figure 6 presents the overall results for execution time, energy, and EDP (network and cache) for 11 of the Splash-2 benchmarks for 256-Kbyte L1 caches and realistic writer prediction (comparable results, accounting for the reduced prediction potential, for 32-Kbyte L1 caches are available elsewhere[15]). Each graph shows the percentage reduction (improvement) in execution time, energy, and EDP normalized to the base MESI protocol. For each benchmark we show results for 2, 4, 8, and 16 cores. In each case, we present the results for the TRO protocol, for the TRO with writer prediction (TRO + WP), and for the TRO with oracle prediction (Oracle WP). As the graphs show, the TRO protocol alone achieves a modest reduction in execution time over the base protocol—from 0.3 percent (2 cores) to 9.48 percent (16 cores) on (arithmetic) average over all benchmarks—while writer prediction yields a reduction from 2.57 percent (2 cores) to 15.37 percent (16 cores). Both the TRO and the TRO + WP yield significant reduction in network and cache energy, ranging from 10.58 percent (2 cores) to 35.88 percent (16 cores) for TRO and 27.66 percent to 52.8 percent for TRO + WP (Figure 6b). This is the result of eliminating a portion of the overall network traffic and reducing L1 misses and L2 accesses. The reduction in cache and network energy combined with execution time reduction gives the EDP reduction shown in Figure 6c. Overall, the reduction in EDP is 8.66 percent (2 cores) to 39.77 percent (16 cores) for TRO and 17.38 percent to 50.78 percent for TRO + WP. These results contribute to our initial argument for better EDP scaling.

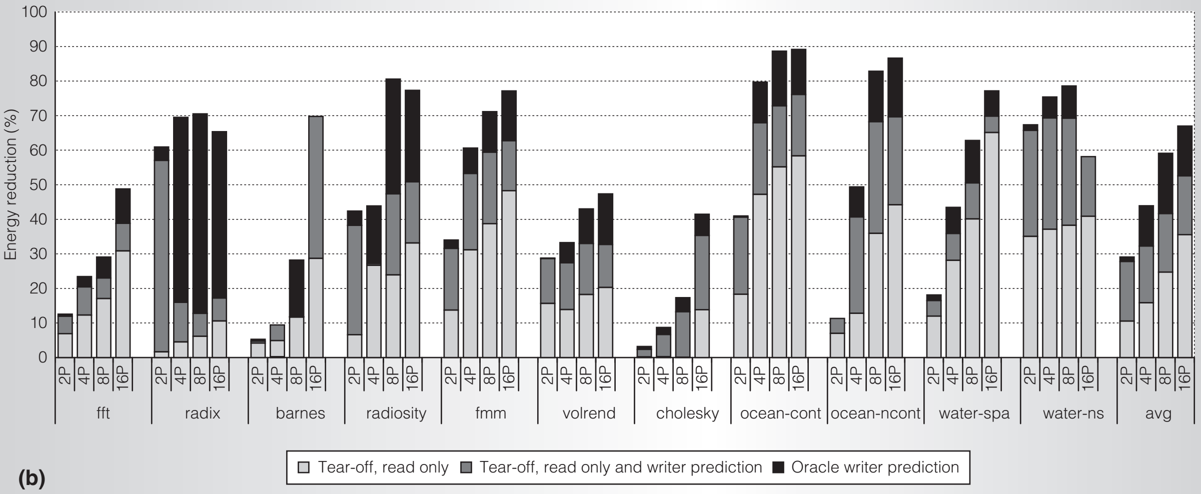
To gain a better understanding of these results, especially with respect to individual benchmarks' behavior, we also present the reduction experienced in the network traffic in Figure 7. Because of space limitations, we only present the results for eight and 16 cores. We place traffic in three categories: invalidation traffic (invalidations and acknowledgments), data messages, and control messages (requests). According to our previous analysis, the TRO protocol almost eliminates the invalidation traffic. This is indeed the case for most benchmarks. Only cholesky sees just a modest reduction in invalidation traffic. In this case, although the actual invalidation messages are sufficiently reduced (up to 75 percent in 16 nodes), those that remain correspond to a relatively large number of acknowledgment messages.

The basic protocol transactions of TRO + WP remain invariant with respect to data traffic compared to the base protocol. TRO copies, however, can be self-invalidated and refetched even in the absence of intervening writes. This results in additional data traffic nonextant in the base protocol. Recall that we do not use any directory classification (that is, for frequently written versus read-mostly lines) to avoid this. Increased data traffic due to this phenomenon appears in barnes, radiosity, and cholesky. Even when protocol classification is present, barnes cannot benefit from tear-off copies.[5] In many benchmarks (fft, fmm, volrend, ocean-cont, ocean-ncont, water-spa, and water-ns) we see fewer data messages. In the base protocol, data are put back to the directory when a writer is downgraded because there is no "owner" state. In contrast, because we do not downgrade the writers on TRO reads, we achieve the same benefit as with having an owner state in addition to a significant reduction in L1 upgrade misses—32.06 percent on average in 16 cores.

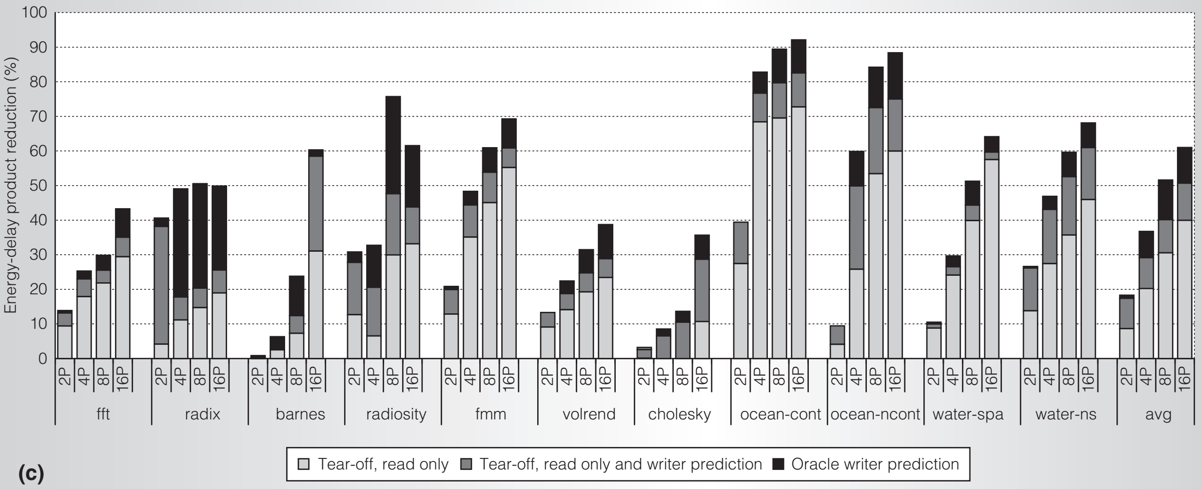
Writer prediction also has a significant effect on the control (request) messages. The reduction of requests depends on both the correct predictions and the mispredictions (which introduce additional control messages). In general, the benchmarks with good accuracy also show a significant decrease in control messages. Prediction accuracy ranges from 91.43 percent in two cores to 73.97 percent in 16 cores, averaged over all benchmarks. Detailed accuracy and coverage results are available elsewhere.[15] The combination of the change in the three message categories (invalidation traffic, data, and control messages) weighted by their relative frequency in the program actually



(a)



(b)



(c)

Figure 6. Execution time reduction (a), cache and network energy (b), and EDP reduction (c) normalized to the base MESI protocol for 256-Kbyte L1 caches.

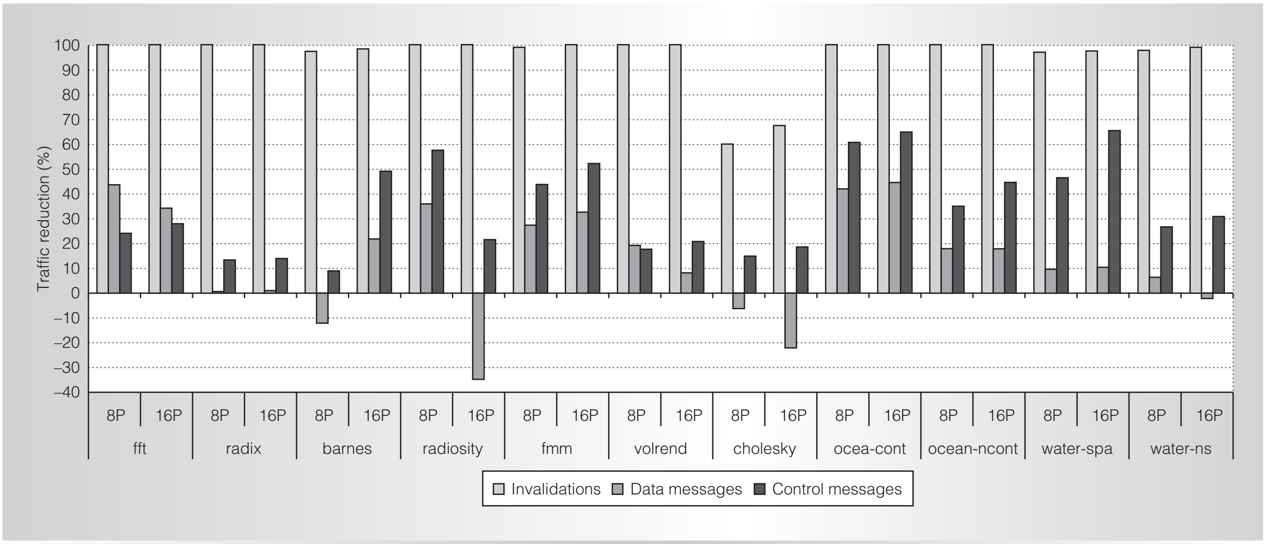


Figure 7. Relative traffic reduction (TRO + WP versus the base MESI protocol) for 256-Kbyte L1 caches.

represents the reduction in L2/directory accesses, which ranges from 34.08 percent for two nodes to 55.91 percent for 16 nodes (averaged over all benchmarks) and reaching as high as 78.12 percent for ocean_cont in 16 nodes.[15] L2 cache accesses have an important contribution in the total energy. Their reduction combined with reduced network traffic constitute the bulk of our energy savings.

By simply relying on the independence of tear-off blocks from costly (and many times complex) directory updates, our writer prediction approach avoids many of the problems and much of the complexity of earlier attempts to graft prediction onto cache coherence. It thus opens the way to efficiently incorporate further optimizations without inordinately complicating the coherence protocols. Inspiration for many optimizations can be drawn from the vast body of prior work on cache coherence, spanning hardware caches to software virtual memory systems. Our future work concentrates on incorporating such optimizations to further reduce power consumption and improve network performance. MICRO

References

1. M. Hill, ''What is Scalability?'' *Computer Architecture News,* vol. 18, no. 4, 1990, pp. 18-21.
2. J.P. Singh and D. Culler, *Parallel Computer Architecture: A Hardware/Software Approach,* Morgan-Kaufmann Publishers, 1998.
3. A. Ramirez et al., ''The SARC Architecture,'' *IEEE Micro,* vol. 30, no. 5, 2010, pp. 16-29.
4. D. Lenoski et al., ''The Stanford DASH Multiprocessor,'' *Computer,* vol. 25, no. 3, 1992, pp. 63-79.
5. A.R. Lebeck and D. Wood, ''Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors,'' *Proc. Int'l Symp. Computer Architecture* (ISCA-22), IEEE Press, 1995, pp. 48-59.
6. K. Gharachorloo et al., ''Programming for Different Memory Consistency Models,'' *J. Parallel and Distributed Computing,* vol. 15, no. 4, 1992, pp. 399-407.
7. M.E. Acacio et al., ''Owner Prediction for Accelerating Cache-to-Cache Transfer Misses in a cc-NUMA Architecture,'' *Proc. Int'l Conf. Supercomputing* (ICS-02), IEEE Press, 2002, pp. 1-12.
8. S. Kaxiras, G. Keramidas, and I. Oikonomou, ''Power-Efficient Scaling of CMP Directory Coherence,'' *Proc. Workshop Programmability Issues for Multi-Core Computers,* 2009; available at http://multiprog.ac.upc.edu/multiprog09/resources/proceedings2009.pdf
9. A.L. Cox and R.J. Fowler, ''Adaptive Cache Coherency for Detecting Migratory Shared Data,'' *Proc. Int'l Symp. Computer Architecture* (ISCA-20), IEEE Press, 1993, pp. 98-108.

10. P. Stenström, M. Brorsson, and L. Sandberg, ''An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing,'' *Proc. Int'l Symp. Computer Architecture* (ISCA-20), IEEE Press, 1993, pp. 109-118.
11. S. Kaxiras and J.R. Goodman, ''Improving CC-NUMA Performance Using Instruction-based Prediction,'' *Proc. Int'l Symp. High-Performance Computer Architecture* (HPCA-5), IEEE Press, 1999, pp. 161-172.
12. A. Agarwal, M. Horowitz, and J. Hennessy, ''An Evaluation of Directory Schemes for Cache Coherence,'' *Proc. Int'l Symp. Computer Architecture* (ISCA-15), IEEE Press, 1988, pp. 280-298.
13. S. Kaxiras, Z. Hu, and M. Martonosi, ''Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power,'' *Proc. Int'l Symp. Computer Architecture* (ISCA-28), IEEE Press, 2001, pp. 240-251.
14. S. Kaxiras and C. Young, ''Coherence Communication Prediction in Shared-Memory Multiprocessors,'' *Proc. Int'l Symp. High-Performance Computer Architecture* (HPCA-6), IEEE Press, 2000, pp. 156-167.
15. S. Kaxiras and G. Keramidas, ''Power-Scalable Coherence, HiPEAC tech. report,'' http://www.hipeac.net/system/files/Power_Scalable_Coherence_TR.pdf, 2010.

Stefanos Kaxiras is a professor in the Information Technology Department at Uppsala University, Sweden. His research interests include power-efficient processor/memory design and modeling of power consumption and performance in multicore architectures. Kaxiras has a PhD in computer science from the University of Wisconsin.

Georgios Keramidas is a postdoctoral fellow at the Industrial Systems Institute, Greece, and a visiting assistant professor at the University of Patras, Greece. His research interests include computer architecture, in particular, the design of the memory subsystem of single core and multicore systems. Keramidas has a PhD in electrical and computer engineering from the University of Patras.

Direct questions and comments about this article to Georgios Keramidas, ISI, Patras Science Park S.A., Stadiou str., Platani, Patras GR-26504, Greece; keramidas@ece.upatras.gr.