JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers

Andreas Moshovos, Gokhan Memik[†], Babak Falsafi[‡], Alok Choudhary[†]

Electrical and Computer Engineering
University of Toronto
moshovos@eecg.toronto.edu

[†]Electrical and Computer Engineering
Northwestern University
{memik, choudhar}@ece.northwestern.edu

[‡]Electrical and Computer Engineering
Carnegie Mellon University
babak@ece.cmu.edu

Abstract

We propose methods for reducing the energy consumed by snoop requests in snoopy bus-based symmetric multiprocessor (SMP) systems. Observing that a large fraction of snoops do not find copies in many of the other caches, we introduce JETTY, a small, cache-like structure. A JETTY is introduced in-between the bus and the L2 backside of each processor. There it filters the vast majority of snoops that would not find a locally cached copy. Energy is reduced as accesses to the much more energy demanding L2 tag arrays are decreased. No changes in the existing coherence protocol are required and no performance loss is experienced. We evaluate our method on a 4-way SMP server using a set of shared-memory applications. We demonstrate that a very small JETTY filters 74% (average) of all snoop-induced tag accesses that would miss. This results in an average energy reduction of 29% (range: 12% to 40%) measured as a fraction of the energy required by all L2 accesses (both tag and data arrays).

1 Introduction

The ever-increasing levels of on-chip integration have enabled phenomenal improvements in computer system performance. Unfortunately, the increase in performance has been accompanied by an increase in power dissipation. High power dissipation has historically been a concern of mobile system designers, because it reduces battery life, diminishing the utility of mobile platforms. High power dissipation is now becoming a concern of server designers, because it requires more expensive packaging and cooling technology, increases cost, and decreases product reliability [21,24]. The key to continued proliferation of servers (e.g., for use in networking, telecommunication, and enterprise computing) is server cost-effectiveness and reliability. Accordingly, techniques to reduce power dissipation in servers are becoming increasingly important.

In state-of-the-art processors, a significant fraction of the power is dissipated in caches. The primary component of power dissipation in today's CMOS circuits is the switching energy due to charging and discharging of load capacitances whenever the circuit transistors switch. CMOS memory structures — such as caches — exhibit high capacitive loads on their bit lines, dissipating correspondingly large amounts of switching power [11]. Conventional circuit-level power reduction techniques — such as voltage scaling and clock gating — have helped in maintaining low power dissipation levels across chip generations. However, computer system designers are also beginning to resort to (micro-)architectural solutions to reduce power dissipation. Previous research has primarily focused on reducing bit line power dissipation in uniprocessor memory hierarchies. Most of these techniques exploit locality to service a large fraction of accesses using very small caches [3,13] or by dynamically reducing the cache associativity [1].

In symmetric multiprocessor (SMP) servers — the most popular small- to medium-scale commercial server platforms — lower cache hierarchy levels (such as L2) dissipate considerable amounts of power. Unlike uniprocessors, in these systems *coherence snoops* account for a substantial if not dominant number of cache accesses and amount of power dissipation in the lower cache hierarchy levels. In SMPs, processor memory reads/writes may generate either a cache fill request from memory or a write permission request for an already cached block copy. In a typical write-invalidate protocol, all bus-side cache controllers "snoop" the bus upon a request, substantially increasing the access frequency to lower-level caches as compared to uniprocessors. For example, we have found that for a set of commonly used benchmarks, snoops double or quadruple L2 accesses on 4-way or 8-way SMPs respectively.

There are optimizations that reduce energy consumption in L2 such as using a dedicated tag array for snoops or serial tag and data array accesses (e.g., as in Alpha 21264 [4] and Intel Xeon [2]). While effective, these optimizations only reduce energy consumption in the data array and not in the tag array. In SMPs, however, tag array energy consumption is also high because SMPs incorporate large L2 caches with high-associativity (to reduce bus traffic). In such caches, tag lookups involve reading multiple cache block tags (while data accesses involve only a single data block) and account for a significant fraction of the overall energy consumed.

While frequent, coherence snoops typically "miss" (i.e., fail to find the block of interest) in the tag array, thereby wasting the energy consumed. For example, using the analytical model of Kamble and Ghose [11] (in Section 2.1) we estimate that snoop-miss-induced tag accesses account for about 33% of all energy consumed by L2 caches for a 4-way SMP with Intel Xeon-like caches and assuming typical L2 hit and snoop rates for the applications we studied.

This paper proposes JETTY, a family of energy-efficient

[1] This work was performed while A. Moshovos was at the ECE Dept., of Northwestern University and while B. Falsafi was at the School of ECE, of Purdue University.

Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01)
0-7695-1019-1/01 $10.00 © 2001 IEEE

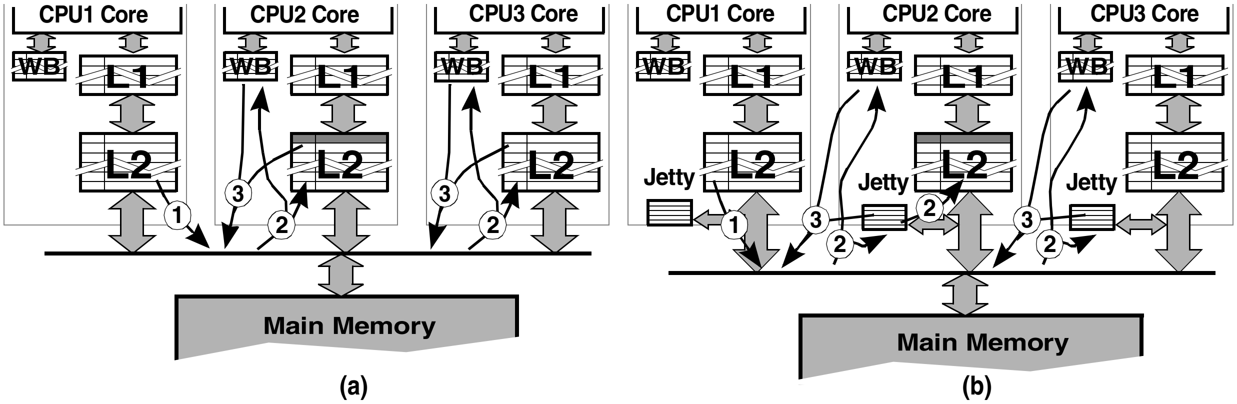


Figure 1: *(a) Conventional snoopy-coherence, bus-based SMP system: All L2 tag-arrays consume energy for snoops. (b) JETTY enhanced system: the local JETTY filters snoops that would miss. Only if a hit is likely, the L2 tag-array is probed consuming energy (and so is the L1 tag-array if necessary). Otherwise, energy consumption is limited to that required by the local JETTY and the writeback buffer array.*

structures that can filter snoop traffic and reduce energy consumption in all lower-level caches in SMPs. A JETTY resides between the processor/memory-bus interface at each SMP processor node. A bus snoop request first probes the local JETTY. JETTY either responds and *guarantees* that no copies exist — eliminating cache tag accesses and saving energy — or responds that copies *may* exit requiring a subsequent snoop to the cache hierarchy.

We propose a number of possible JETTY variants exploiting key memory access and sharing patterns. One set of JETTY designs capitalizes on temporal and spatial locality of sharing and the resulting induced bus snoops. These designs capture recently missed snoops — i.e., recording a *subset* of blocks that are not cached — in a small, associative structure. Another set exploits regularity in memory access patterns and block addresses — encoding a *superset* of block addresses that are cached — in a compact random-access structure.

On the average, JETTY reduces energy provided there is a sufficiently large fraction of snoops that miss in caches. As we show in Section 4, for the parallel programs we studied JETTY results in significant energy savings. It is likely that the savings will be larger when an SMP is used mostly as a throughput-engine (i.e., running several independent programs) rather than as a parallel-engine.

While L2 power already represents a sizeable fraction of overall power (see Section 2.1), a study of potential optimizations such as those we describe is further justified for the following reasons: (1) As L2 size and associativity increase the power required for their operation also increases. This is especially a concern for single-chip multiprocessor systems and for processors that integrate large on-chip caches. (2) As other power-optimization techniques are perfected, tag-related optimizations like JETTY will increase in importance. (3) Finally, it is likely, that similarly to performance optimizations, a plethora of power-optimizations will be needed at multiple levels (software and hardware) and structures to attain a desired overall power reduction.

The rest of this paper is organized as follows. In Section 2, we present the rationale for our approach, together with an analysis of the relative importance of snoop-miss energy consumption. In Section 3, we discuss JETTY's operation and present several alternative organizations. We present experimental results in support of our method's utility in Section 4. In Section 5, we comment on related work. Finally, in Section 6 we summarize our findings and offer concluding remarks.

2 An Opportunity to Reduce Snoop-Induced Energy Consumption

To motivate our snoop-filtering approach, we briefly describe how a conventional SMP handles snoop requests. In Section 2.1, we argue that snoop-misses constitute a sizeable fraction of overall power dissipation. Finally, in Section 2.2, we discuss complexity and latency issues.

Figure 1 illustrates a typical SMP consisting of three processors with local caches. Shown are a write-buffer (WB) and L1 and L2 data caches per processor. We omit the instruction hierarchy and any other buffers that may exist between adjacent levels for presentation clarity. A shared bus connects the processors together and to a memory system. A cache-coherence protocol maintains the data integrity among the processor cache hierarchies. Processor memory reads and writes to blocks that are not cached, or writes to blocks that are potentially cached by others result in bus requests and bus snoops from all other caches on the bus.

Figure 1(a) illustrates bus-snooping for a simple producer/consumer sharing between CPU1 and CPU2. In this example, CPU1 (the consumer) reads from address “a” (shown as a shaded block) which misses in its local memory hierarchy. As a result, the request appears on the bus (action 1), resulting in snoops from all other CPUs (action 2). In this paper, we assume inclusion between L2 and L1, and therefore each bus snoop (action 2) first probes all the L2 tag arrays and the WBs (the snoop accesses the L1 tag array only when necessary given the information provided by the L2 and the type of the access).[2] The snoop transaction completes when CPU2 (the producer) responds with a copy of “a” (action 3), inhibiting CPU3 and memory from responding. Because CPU3 does not have a copy of “a”, it incurs a snoop miss and wastes energy.

As the example shows, *all* L2 tag-arrays in *all* CPUs consume energy even though *not all* of them have a copy. Provided that many of the snoop-induced accesses miss, there is an opportunity for reducing energy consumption. JETTY capitalizes on this opportunity by using small structures to identify most snoop-induced tag probes that miss. A JETTY-enhanced system is shown in Figure 1(b). As shown, the JETTY in CPU3 determines that no local copies exist avoiding probing the much larger L2 tag array. JETTY will not filter snoops to the WB. However, the WB is much smaller than the L2 tag array in typical systems.

JETTY relies on the following requirements to be successful:

1. A large enough fraction of snoop-induced L2 accesses should result in a miss.
2. It should be possible to identify most of these (would-be) misses using a small enough structure.
3. We should never report a would-be miss while the data is locally cached.

Fortunately, in the common case most snoops miss in L2. In throughput-oriented server workloads, processors run distinct programs and the only L2 misses resulting in a snoop hit are due to highly infrequent activities such as process migration among processors [25] or sharing of operating system data structures. In many parallel scientific/engineering [31] and commercial applications [12,20] a substantial fraction of L2 misses are to data structures only accessed by a single processor, resulting in snoop misses in all L2s. Moreover, the most common form of (either migratory or producer/consumer) sharing occurs among two processors resulting in snoop misses in all but a single L2 [12,28]. In Section 4, we use simulation to demonstrate that rarely a snoop finds copies in *any* of the L2s for a set of parallel scientific/engineering applications.

In the common case of a snoop miss, JETTY reduces energy consumption, effectively maintaining a lower operating temperature and offering higher reliability. In the infrequent but worst case, e.g., an access to widely-shared data where all caches have a read-only copy of a block, JETTY may increase energy consumption. However, since JETTY is much smaller than the tag hierarchy it will have a small impact on overall power dissipation. Moreover, existing processors already include both temperature monitoring hardware and the mechanisms necessary (e.g., frequency or voltage scaling) to take action when appropriate [6,8].

2.1 Snoops Consume a Sizeable Fraction of Overall Energy

Ultimately, JETTY’s utility depends on whether snoop-induced misses contribute significantly to overall energy consumption. Unfortunately, no published data exists on the *average* power consumed by L2 caches in servers. To estimate energy consumption due to snoop misses, we use a three-fold approach. First, we rely on published data on the *peak* power dissipated by L2 caches for a commercial processor. Then, we present an argument why on the average L2 accesses consume a sizeable fraction of overall energy. Finally, we use a model to estimate the fraction of overall L2 energy consumed by snoop-induced misses.

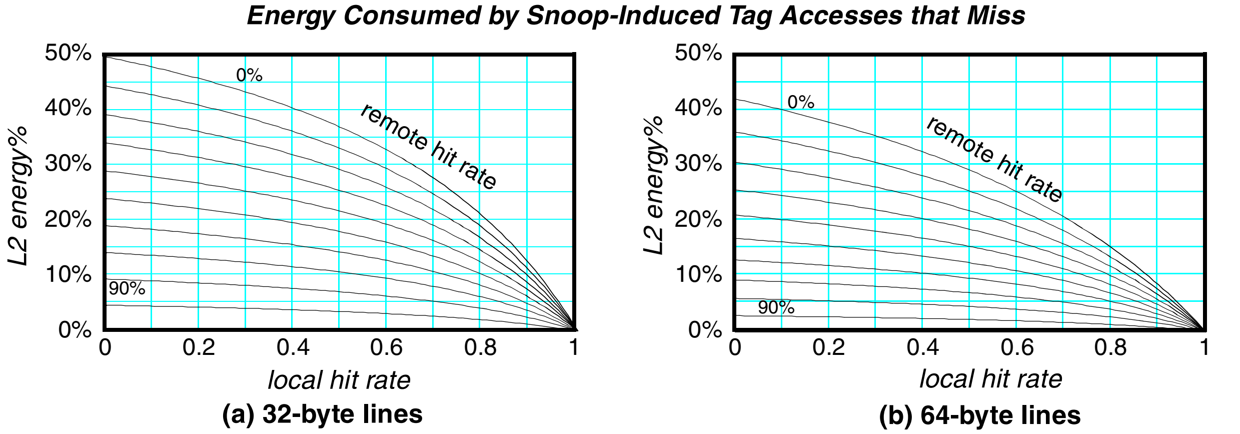
The Intel Xeon II processor contains an L2 comprising a set of external SRAMs [9]. Fortunately, separate power figures exist for the core and the external L2. These are reported in Table 1. For the 1Mbyte part, the L2 (data + tags) accounts for 23% of overall *peak* power. Excluding L2 pad power from the overall peak power, L2’s contribution rises to 28%.Of course, *average* power dissipation depends on how often L2 is accessed. However, L2 power dissipation is comparable to that of the L1 data cache which has been the main focus of much previous work. For the programs we studied about one to 10 in every 100 processor accesses would not hit in L1. While this may seem to imply that L1 would consume a lot more power (by one or two orders of magnitude), this is not true. First, L2 is typically much larger than L1 (e.g., 10 times or more in SMPs) and uses higher associativity. Moreover, a large (if not a dominant) fraction of accesses that would otherwise hit in L1 are typically served by stores pending in the writebuffer [16], or can be served by the addition of line buffers [29]. Furthermore, as we explain, the L2 access count is further amplified by snoops. In particular, as we show in section 4, snoops double and quadruple the L2 access count on a 4-way and 8-way SMP respectively.

400Mhz Xeon	**Power (peak)**				
	Absolute			**Relative over CPU + L2**	
L2 size	**Core**	**L2**	**L2 pads**	**L2**	**L2 w/o pads**
512K	23.3W	*4.5W*	*3W*	***14%***	***16%***
1M	23.3W	*9W*	*6W*	***23%***	***28%***
2M	23.3W	*18W*	*12W*	***34%***	***43%***

Table 1: *Breakdown of the power dissipated on a commodity processor used to build glue-less SMP systems (source [6]). In this processor, L2 is implemented using external, custom SRAM chips. We report the peak power (MAX) dissipated by the processor core, and the L2 and the L2 pads. The right-most columns report L2 power as a fraction of overall power (core + L2). Under the “L2” column we include the pads into the overall power. Under the “L2 w/o pads” we exclude pad power in the overall power to get an estimate of L2 power for a hypothetical on-chip L2.*

While L2 power dissipation represents a sizeable fraction of overall power, JETTY can reduce energy consumption only for snoop-induced tag lookups. To gauge JETTY’s potential, we used Kamble and Ghose’s model [11] to estimate energy consumption for the tag and data arrays of a 1Mbyte 4-way set-associative L2 with either 32-byte or 64-byte blocks. For calculating the space required by tags, we assumed an IA-32-like 36-bit physical address space in addition to 2 bits for MOSI stage encoding. Moreover, we used CACTI [30] to determine the optimal number of banks for a 0.18μm process. We also made the conservative assumption that the tag and data arrays are accessed serially to reduce power. Bohwill, et al., estimated

[2] In systems that do not maintain inclusion, a snoop also probes the L1, further increasing snoop-induced power dissipation.



(a) 32-byte lines

(b) 64-byte lines

Figure 2: *Analytical models of energy dissipated by L2 snoop-induced tag lookups that miss as a faction of local L2 hit rate (X axis) and different remote hit rates (curves). The Y axis reports energy as a fraction of the overall energy consumed by all L2 caches (including misses and hits and both the tag and data arrays). Remote hit rates range from 0% (top) to 90% in 10% steps.*

the power savings due to this optimization to 10W for the Alpha 21164 (overall power is at 70W) [4]. Bateman, et al., report a 75% reduction in power for Intel's XEON L2 [2].

The results are shown in Figure 2. The Y axis reports snoop-induced miss energy consumption as a function (X axis) of local hit rate. We define *local hit rate* as the fraction of locally initiated accesses that hit in L2. We report snoop-induced miss energy as a fraction over all energy consumed by all accesses to L2s. We define *remote hit rate* as the fraction of L2 snoop-induced accesses that hit. The remote hit rate range shown is 0% (top curve) to 90% in steps of 10%. Note that a local L2 miss results in three remote L2 accesses since this is a 4-way SMP system. For example, assuming a 50% local miss rate, one out of two local accesses will result in a snoop. Since there are four processors, this implies that for every two local accesses, each L2 cache will also observe three snoops resulting from remote L2 misses. Consequently, in this scenario remote snoops account for the majority of L2 accesses.

It can be seen that the *relative* energy consumed by snoop-induced tag lookups that miss grows as the local and/or remote hit rate decrease. As the local hit rate decreases, snoops increase resulting in a 3X increase in remote L2 snoop-induced accesses. Similarly, as remote hit rates decrease more snoop-induced tag-lookups result in a miss, hence JETTY's potential increases. Snoop-induced miss energy consumption is higher for the 32-byte block cache compared to the 64-byte block cache. This is explained by the lower energy required by the data array.

We have shown that JETTY's potential depends on the local and remote hit rates. For example, assuming a 50% local hit rate and a 10% remote hit rate, snoop-miss tag lookups account for 33% of the power dissipated by all L2s (with 32-byte blocks). In Section 4.4, we will see that JETTY yields significant energy savings even when all L2 accesses are considered.

2.2 Complexity and Latency Considerations

A key advantage of JETTY is that it is readily-applicable to existing SMPs and requires no modifications to the coherence protocol. Coherence protocols can get arbitrarily complex optimizing for various application sharing patterns. Moreover, protocol finite-state-machines are hard to design, debug, and verify [7]. To minimize interaction with the protocol the JETTY designs we propose maintain no coherence state information other than "presence"; JETTY simply filters snoops to blocks not present obviating the need to change the existing protocol. Designs that exploit JETTY's interaction with the protocol to further save power are beyond the scope of this paper.

Because JETTY appears in series with the L2, it will increase response latency for non-filtered snoops. However, we expect that this increase will be an insignificant fraction of overall snoop latency. Specifically, JETTY's latency should be considerably smaller than that of the L2 tag array since JETTY is a much smaller and fairly straightforward structure. In fact, as we will see in Section 4, the largest JETTY structure we use is almost identical to an 8-ported 32 by 32-bit register file. In typical processors, the latency of such structures is a fraction of the processor cycle (half a cycle in many processors to allow both a read and a write). In contrast, it takes several (e.g., 12) cycles to access a reasonably sized L2. Moreover, state-of-the-art snoopy buses are several (4~10) times slower than processors.

3 Jetty Variants

In this section, we discuss three JETTY variants: (1) the exclude-JETTY, (2) the include-JETTY, and (3) the hybrid-JETTY. What differentiates them is the type of information they record. The *exclude-JETTY* contains information on recent blocks that *are not* present in the local L2. The *include-JETTY* contains aggregate information about *all* blocks currently present in the local L2 cache. Finally, the *hybrid-JETTY* combines both approaches. All variants are speculative in nature: They indicate that a block is either *not* cached (guarantee) or that it *may* be cached in the L2. In effect, they identify a *subset* of blocks that are not cached and a *superset* of blocks that are cached.

3.1 Exclude-Jetty

Exclude-JETTY (EJ) keeps a record of blocks that have been snooped recently, missed in the local L2 and are still not cached, i.e., a *subset* of blocks that are not locally cached. EJ is

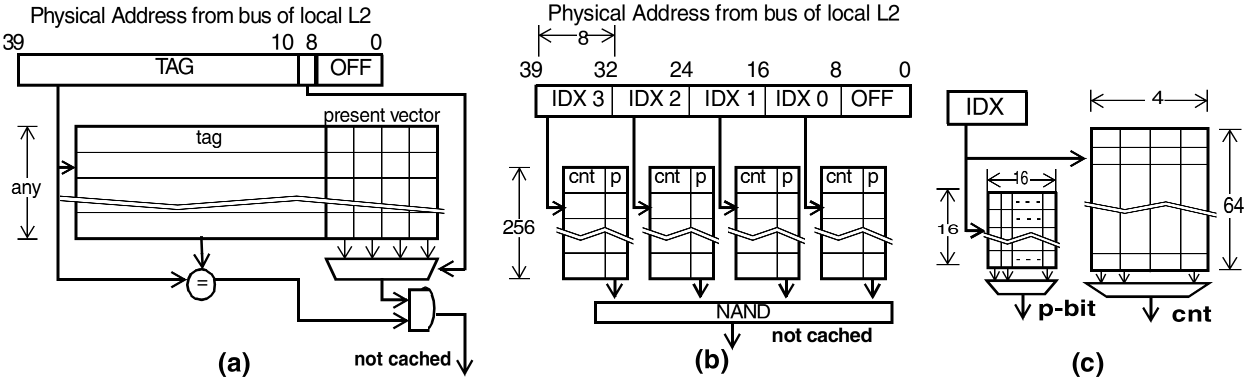


Figure 3: *(a) A vector-exclude-JETTY: it contains information about recent snoops that missed in the local L2. (b) An include-JETTY: it contains information about the blocks currently cached in the L2. (c) Example showing power-optimized IJ sub-arrays. Separate p-bit and cnt arrays are used per each 256-entry sub-array of part (b).*

a small array containing (TAG, present-bit) pairs. A match in EJ upon a snoop is a guarantee that the block is not cached locally. An entry is allocated when a snoop misses in the local L2. Subsequent accesses to the same block will be successfully filtered so long as the block is not evicted from EJ. An EJ entry is evicted (present-bit reset) when a local miss loads in the corresponding block.

EJ exploits locality in the snoop stream. For example, producer/consumer sharing often arises among a small number of processors. In such a scenario, EJ can capture the snoop miss stream in the rest of the processors' L2s. Moreover, each consumer read results in a snoop in all consumers' L2. EJ can also capture the snoop misses in all the consumers' caches. Other examples of common snoop streams EJ captures are migratory sharing in small critical sections when data migrates from one processor to another, and conflict traffic in L2s resulting in temporal locality in snoop requests for a small number of block addresses.

A variation of EJ, *Vector-Exclusive-JETTY* (VEJ) exploits spatial locality in the snoop stream, and uses a (TAG, present-vector) pair to encode presence for a chunk of consecutive blocks. The present-vector (PV) is a *n*-bit mask indicating which blocks starting from *TAG* and ending with *(TAG+n - 1)* are currently not in L2. Figure 3(a) illustrates an example assuming a 40-bit physical address space, 256-byte L2 blocks and a 4-bit PV. As shown, instead of storing the full 32-bit tag we store the upper 30 bits only. The lower 2 bits of the TAG are used to select the appropriate bit from the 4-bit PV. Bit 0 of the PV corresponds to address TAG+0 while bit 3 corresponds to address TAG+3. A block is not cached if its TAG matches an entry in the VEJ and the corresponding present bit is set.

3.2 Include-Jetty

While there are many scenarios that result in locality in the snoop stream, capturing the snoop streams from all SMP processors effectively may require a prohibitively large EJ. An alternative to recording blocks that *are not* locally cached (as in EJ), is to keep information about those that *are*. Our alternative JETTY design, Include-JETTY (IJ), contains information that identifies a *superset* of the blocks currently cached in L2. When a snoop misses in IJ, IJ guarantees that a block is not cached in L2.

The various IJs we studied are all derived from the basic organization shown in Figure 3(b). We explain its operation via an example and by assuming a 40-bit physical address space and 256-byte L2 blocks. The example IJ consists of four 256-entry sub-arrays. The relevant 32-bit part of the PA is split into four 8-bit parts (IDX 0 to IDX 3). These parts are used to access the four sub-arrays in parallel. Each entry reports a count (*cnt*) and a present (*p*) bit. Let us ignore the cnt fields for the time being. A p-bit indicates whether there is at least one cached block whose tag matches the corresponding bit pattern. For example, in the left-most sub-array, the p-bit of the 1st entry (entry 0) matches block address of the form *0x00xxxxxxx*. The 256th entry matches block addresses of the form *0xFFxxxxxxx*.

If *any* of the four p-bits retrieved for a block address are zero (IJ miss), then *no* L2 block matches this address. If *all* of p-bits are non-zero (IJ hit), then a block *may* be locally cached. In this case we have to probe the L2 tag-array to determine whether the block is actually cached. In effect, each sub-array represents a superset of all cached blocks (via the non-zero p-bits). Accordingly, the intersection of all these supersets (one per sub-array) is also a superset of all cached blocks.[3]

Because missing in the IJ implies that a block is not in L2, it is imperative to keep IJ's information coherent. To do so, we keep track of the exact number of blocks that match each IJ entry via the *cnt* fields. When a block is allocated or de-allocated all corresponding IJ counters are incremented or decremented respectively. At most one counter per sub-array is updated at any time. Since the p-bit encodes presence, we use a count value of 0 to report 1 matching block, a value of 1 to report 2 matching blocks and so on. A p-bit is reset when we de-allocate a block and the matching counter is zero. A p-bit is set when a matching block is allocated and the p-bit is zero. For this method to work it is necessary to communicate the addresses of replaced L2 blocks to the IJ. This information is available at replacement time in L2 (no additional accesses are

[3] This organization may in effect be an implementation of a hash function. If so, we could use a single p-bit array accessed through a carefully-tuned hash function.

App.	Ab	Input Parameters	Accesses in M	MA	Local Hit Rates		L2 Snoop Accesses
					L1	L2	
Barnes	*ba*	*16K particles*	967.0	57.4 M	97.8%	31.7%	47.1M
Cholesky	*ch*	*tk15.0*	224.4	26.3 M	98.0%	64.2%	9.9M
Em3d	*em*	*76K nodes, 15% remote, degree 2*	333.4	34.4 M	76.5%	23.3%	252.6M
Fft	*ff*	*256K data points*	60.2	12.7 M	96.8%	36.3%	7.5M
Fmm	*fm*	*16K particles*	1,751.2	36.1 M	99.6%	81.2%	8.1M
Lu	*lu*	*512x512 matrix, 16x16 blocks*	188.7	4.6 M	95.7%	82.5%	6.3M
Ocean	*oc*	*258 x 258 ocean*	182.8	41.6 M	83.5%	52.2%	90.0M
Radix	*ra*	*10M keys*	399.4	82.1 M	96.2%	79.4%	42.6M
Raytrace	*rt*	*car*	299.9	69.1 M	98.3%	46.6%	12.3M
Unstructured	*un*	*mesh 2K*	1,693.6	3.5 M	92.4%	78.7%	304.8M

Table 2: *Applications used in our studies. In the interest of space, we will refer to the applications using the two letter abbreviations listed under the "Ab." column. Reported from left to right are the input parameters, the resulting memory accesses (Millions), the amount of memory allocated (in Mbytes), the hit rates and finally, the number of snoop-induced L2 accesses. To measure hit rate, we count all hits and misses in all four processors. The L2 hit rate is measured over those accesses that miss in the L1 including L1 writebacks. Note a snoop might be necessary even on an L2 hit (i.e., write hit on a shared block). This explains why in some cases there more snoops than misses.*

required). A separate tag-sized set of wires can be used to communicate this information to the IJ. *We do take into account these IJ updates in our energy consumption analysis* (Section 4.4). With this counter-based scheme it is desirable to avoid saturation. We make the pessimistic assumption that a single IJ entry may match all L2 blocks (e.g., this can happen with a fully-associative cache). However, depending on the bits used to index a sub-array and the cache organization fewer bits may be required.

In the example of Figure 3(a) we used all relevant PA bits to index the sub-arrays. We also used non-overlapping, continuous, equal-in-length parts of the PA. None of these are requirements. In fact, we found that using partially overlapped indices results in better accuracy. However, a further investigation of IJ index generation schemes is beyond the scope of this paper.

It is important to emphasize that when a snoop probes IJ, only the p-bits are read. The *cnt* fields are read and updated less frequently. Accordingly, to reduce energy consumption we can use an alternative organization where the p-bits and the *cnt* fields are stored in separate arrays. Moreover, the p-bit array can be organized to contain multiple p-bits per entry. For example, as shown in Figure 3(c), instead of using a 256-entry by 1-bit array we could use a 16-entry by 16-bit organization where part of the index is used to select the entry and the other part to select the appropriate p-bit. The same principle can be applied to the *cnt* arrays as shown in the figure. Note that the largest IJ we have evaluated, has four p-bit arrays each having 32x32-bits (similar to a typical four-ported register file).

3.3 Hybrid-Jetty

IJ contains aggregate information about what is cached locally. However, there are cases where a small set of frequently snooped blocks defies identification by an IJ. At the same time, EJ is well-suited for keeping track of a small number of blocks that are not currently cached but exhibit snoop locality. An obvious alternative is to combine the two methods to increase accuracy in a *Hybrid-JETTY* (HJ). HJ uses both an EJ and an IJ in parallel. When either of the two indicates that no match is possible, we avoid accessing the L2 tag array. Because, EJ serves as backup for IJ, entries are allocated in the EJ *only* when the IJ fails to filter them. To keep JETTY's impact on snoop latency at a minimum (Section 2) HJ accesses both IJ and EJ components in parallel upon a snoop. As we show in Section 4, the hybrid method outperforms IJ and EJ both in accuracy and in energy reduction.

4 Experimental Analysis

In this Section, we evaluate the effectiveness of various JETTY organizations. In Section 4.1, we discuss our methodology including the benchmarks and simulation environment. In Section 4.2, we demonstrate that a large fraction of snoop-induced L2 tag accesses results in a miss. In Section 4.3, we measure the accuracy of various JETTY organizations. In Section 4.4, we measure the energy reduction possible for a set of the better performing JETTY organizations.

4.1 Methodology

We use a set of shared-memory applications widely used in SMP design studies. The applications include those from the SPLASH-2 benchmark suite [31] and two scientific applications from our previous studies [15]: *Em3d,* a shared-memory version of the Split-C benchmark; and *Unstructured*, a computational fluid-dynamics application. Table 2 provides additional information about these applications including the input data sets, the resulting number of memory accesses (in Millions), and the amount of main memory allocated (MA column). Access counts range from about 60 million to 1.7 billion and memory requirements range from 3.5 Mbytes to as much as 82 Mbytes.

We used the Wisconsin Wind Tunnel 2 (WWT2) [17] to collect snoop activity traces and memory reference statistics. WWT2 simulates only a single-level direct-mapped data cache hierarchies.[4] We have modified WWT2 to simulate a 2-level

Application	Remote Cache Hits				L2 Snoop Miss Accesses	
	0	1	2	3	% of Snoop Accesses	% of All Accesses
Barnes	47%	28%	15%	10%	71%	48%
Cholesky	92%	5%	3%	0%	95%	59%
Em3d	80%	17%	2%	1%	92%	69%
Fft	93%	7%	0%	0%	98%	73%
Fmm	82%	15%	2%	1%	93%	39%
Lu	73%	26%	1%	0%	91%	39%
Ocean	97%	3%	0%	0%	99%	66%
Radix	100%	0%	0%	0%	~100%	56%
Raytrace	100%	0%	0%	0%	~100%	69%
Unstructured	33%	55%	4%	8%	71%	28%
AVERAGE	79.6%	15.6%	2.6%	1%	91%	55%

Table 3: *Snoop hit distribution. The column "Remote Cache Hits" depicts the fraction of snoops that miss in all other caches (0 hits), or hit in 1, 2 or all other 3 caches over all snoops. The column "L2 miss snoops" depicts the fraction of snoop-induced L2 tag accesses that result in a miss; these are snoops that JETTY may eliminate. The final column reports the snoop-induced L2 tag accesses that miss as a fraction of all L2 tag accesses. We report statistics for two systems. The first (SB columns) uses subblocking in the L2, while the second (NSB columns) does not.*

on-chip hierarchy per processor. Our base configuration is a 4-way SMP. The memory system is SUN SPARC-like where each processor has 1Mbyte L2 and 64Kbyte L1 direct-mapped caches. L1 blocks are 32 bytes long. L2 blocks are 64 bytes long and consist of two 32-byte subblocks. Coherence is maintained at the subblock level using a MOESI protocol. Subblocking is used in many commercial systems to reduce the tag array size. (We have experimented with a similar configuration that does not use subblocking and we report a summary of these results where appropriate.) We used CACTI [30] to determine the optimal number of banks for a 0.18μm process.

In Table 2, we report the resulting hit rates for L1 and L2. The local hit rates we report in Table 3 include only *local references*, i.e., references initiated by the local processor. Moreover, we report aggregate hit rates over all four processors. The "L2 Snoop Accesses" column reports the number of snoop-induced L2 accesses. In Section 4.2, we report the combined snoop and local accesses hit rate.

To measure energy consumption we have adapted the analytical model developed by Kamble and Ghose [11]. This model calculates power for cache structures as a function of their organization, the number and type of accesses and a set of technology dependent attributes. While this model is an approximation, it has been used extensively in previous power studies for caches, e.g., [1]. In our experiments we have assumed a 0.18μm CMOS technology operating at 1.8V and with the characteristics reported in [5].

4.2 Snoop Activity

In this section, we present empirical evidence that a large fraction of snoop-induced L2 tag accesses result in a miss. Moreover, we present results indicating that snoop accesses constitute a large fraction of — and often dominate — all L2 accesses. The results are shown in Table 3. Under the "Remote Cache Hits" columns we report the remote hit count distribution of snoops: e.g., under column "1" we report the fraction of snoops that find only one remote cached copy. We define a *remote hit* as a snoop transaction that finds at least one cached copy in a processor other the one that generated the transaction. With the exception of *unstructured,* the majority of snoops (79% on the average) do not find remote cached copies. Very few snoops find copies in all other caches (1% on the average). (On a similar configuration where L2 caches were not sublocked 68% of all snoop-induced accesses resulted in a miss.)

The final two columns report summary statistics on snoop-induced L2 tag accesses. Among snoop-induced tag accesses 91% result in a miss. This suggests that there is potential for JETTY to filter a large fraction of snoop accesses. Moreover, when measured as a fraction of all L2 accesses (last column), snoop misses account for about 55% of all accesses. (On a similar configuration where L2 caches were not subblocked snoop-induced misses where 46% of all L2 accesses.) This result suggests that JETTY has the potential for producing significant energy savings as snoop misses constitute a large fraction of all L2 accesses.

4.3 Snoop Miss Coverage

In this section, we evaluate the accuracy of various JETTY organizations. We define snoop miss coverage, or *coverage* for short, as the fraction of snoop-induced L2 tag lookups that miss that are filtered by JETTY. We use coverage as the key metric to evaluate JETTY's ability to filter snoops; we present overall energy consumption including the JETTY's energy in section 4.4. We present coverage measurements first for EJ and VEJ, then for IJ and finally, for HJ organizations.

4.3.1 Exclude-JETTY

We evaluate six different EJ configurations varying both the number of sets and the associativity of the storage array. With

[4] While WWT2 assumes perfect instruction caches, our applications' instruction footprints are very small, and would have minimal impact on L2's performance.

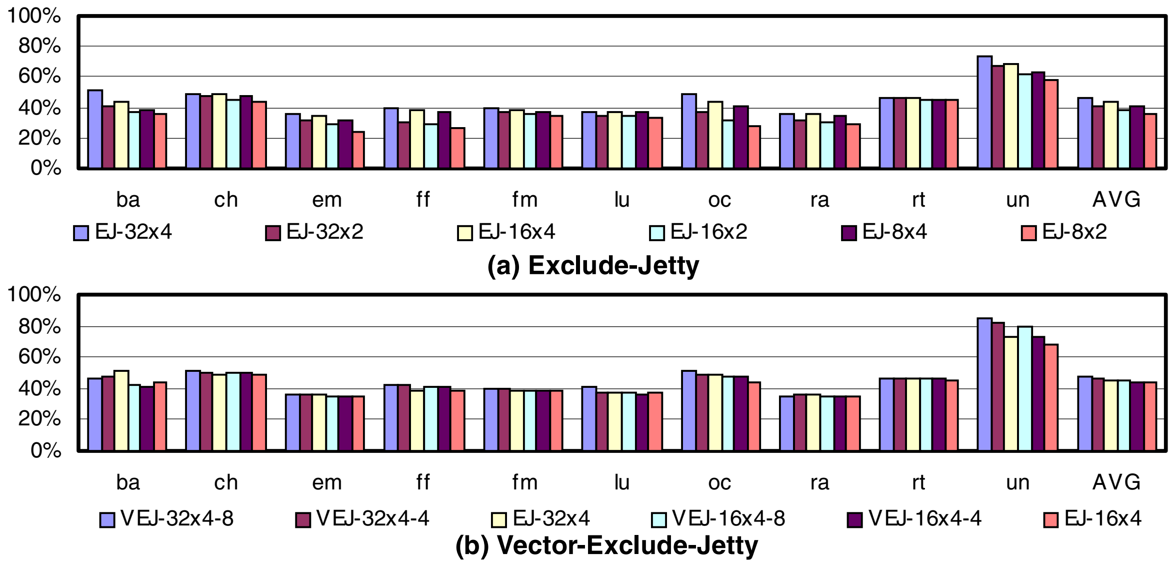


Figure 4: *(a) Exclude-JETTY coverage. Configurations are named as EJ-Sets x Associativity. (b) Vector-Exclude-JETTY coverage. Configurations are named as VEJ-Sets x Associativity x VectorLength.*

EJ-SxA we refer to an S-set and A-way set associative EJ organization (S x A total number of entries). We have experimented with structures having 32, 16, and 8 sets and 2-way and 4-way associativity. Figure 4(a) reports coverage for these configurations.

The various EJ organization perform fairly well suggesting that there is locality in the reference stream as it appears on the bus. For those applications where there is little or no sharing, locality is primarily the result of subblocking. Accesses to the different subblocks within the same L2 block will result in a miss. When sharing exists, coherence actions may force multiple accesses to the same cache block to appear on the bus (e.g., migratory data). As expected using larger EJ organizations or ones with higher associativity results in increased coverage. However, the differences are minor (this is not the case in the system that does not use subblocking where, for example, for *Barnes* EJ-32x4's coverage is 29% while EJ-32x2's coverage is 16.3%). EJ-32x4 performs the best with 45% coverage on the average.

4.3.2 Vector-Exclude-JETTY

We have experimented with VEJ organizations based on the EJ-32x4 and the EJ-16x4 extended with either 8-bit or 4-bit presence vectors. With *VEJ-SxA-V* we refer to an organization having V-bit presence vectors, S sets and is A-way set-associative. The results are shown in Figure 4(b). EJ-32x4 and EJ-16x4 are included for ease of comparison. Using vectors improves coverage over EJ for most applications, albeit only slightly. The highest improvements are observed for *Unstructured.* Unfortunately, it is possible for coverage to decrease compared to an EJ with the same number of entries (e.g., *Barnes*). A VEJ and an EJ with equal number of sets and associativity will use different parts of the PA to determine the set index. This may result in increased pressure for some sets and consequently in thrashing in the VEJ. This also explains why the 4-bit vector VEJ outperforms its 8-bit counterpart in *Barnes.*

4.3.3 Include-JETTY

We evaluate five different IJ organizations. We use an *IJ-ExNxS* naming scheme where 2^E is the number of entries in each sub-array and N is the number of sub-arrays used. To get the N, E-bit wide sub-array indexes we start from the least significant bit of the PA (excluding the block-offset bits). The first index is the E least significant bits. To get the next index, we skip S bits toward the most significant bit. Using S that is less than E results in partially overlapped indexes. No shifters are required for extracting the appropriate indexes, this is done by simply routing the appropriate section of the PA to each sub-array.

We evaluate the following organizations: IJ-10x4x7 (four 1K-entry sub-arrays), IJ-9x4x7, IJ-8x4x7, IJ-7x5x6 and IJ-6x5x6 (five 64-entry sub-arrays). Table 4 reports the space required by each IJ. Recall that for remote snoops, only the p-bit section of each sub-array needs to be accessed. Also note that 14-bits are required (pessimistic assumption) for each cnt-array entry. The space requirements for the p-bit arrays are very small.

The results are shown in Figure 5(a). The IJ-10x4x7, which is the largest IJ evaluated performs the best resulting in 57% coverage on the average. However, the IJ-9x4x7 (which has half the storage requirements of IJ-10x4x7) performs fairly well resulting in about 53% average coverage. For some programs using a larger number of smaller sub-arrays results in better coverage. For example, IJ-7x5x7 (five 16x8-bit sub-arrays) outperforms IJ-8x4x7 (four 16x16-bit sub-arrays) for *Em3d.* As all IJ organizations are speculative in nature representing a superset of all cached blocks, this is possible given an appropriate snoop and cache block address distributions.

There is no direct correlation between IJ and EJ behavior.

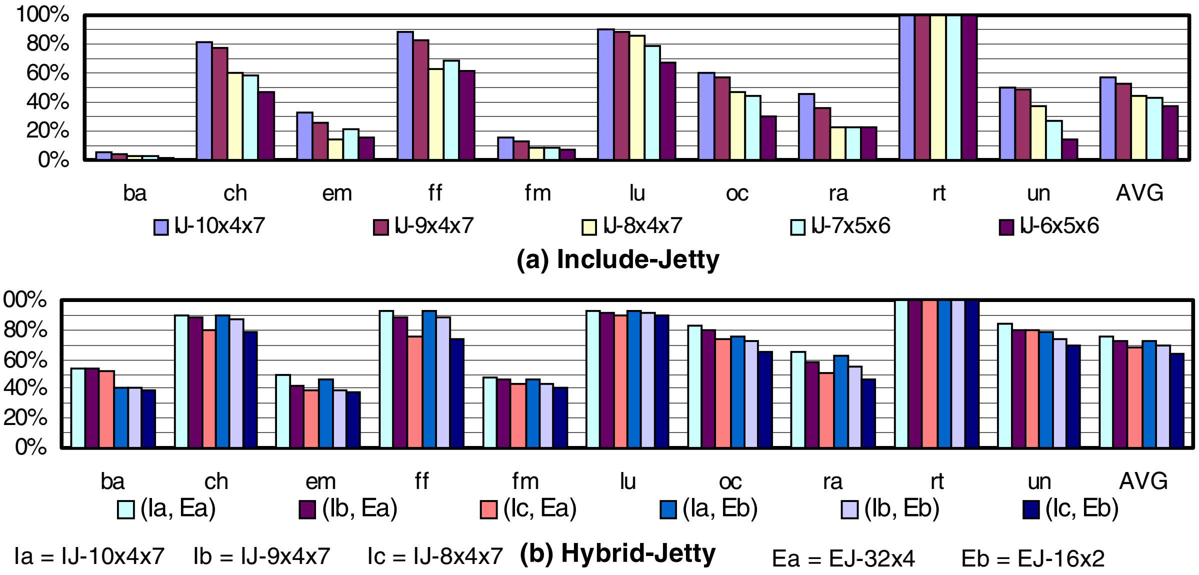


Figure 5: *(a) Coverage results for IJ configurations, named as "IJ-Index x Bits x NoOfSubarrays x SkipBits" (see text). (b) Coverage for HJ configurations named as (IJ, EJ).*

IJ	p-bit Array		cnt array
	in bits	Org. (bits)	
IJ-10x4x7	4 x 1024	4 x 32 x 32	7168 bytes
IJ-9x4x7	4 x 512	4 x 16 x 32	3548 bytes
IJ-8x4x7	4 x 256	4 x 16 x16	1792 bytes
IJ-7x5x6	5 x 128	5 x 8 x16	869 bytes
IJ-6x5x6	5 x 64	5 x 4 x16	448 bytes

Table 4: *Storage requirements of various IJ configurations. Note that on a snoop, only the p-bit array is accessed.*

While EJ performs similarly for most programs, IJ appears to work a lot better for some than others. For example, for *raytrace*, IJ captures virtually all snoops that miss while EJ captures only about half. This suggests a potential synergy of the two methods and serves as the motivation for experimenting with hybrid organizations.

4.3.4 Hybrid-JETTY

As explained in Section 3.3, an HJ contains both an IJ and an EJ operating in parallel. We experimented with various methods. Here we report six methods which are derived by combining various IJ and EJ organizations (VEJ organizations performed slightly better than EJ ones but the differences where small). We use an (IJ, EJ) naming scheme. IJ is any of IJ-10x4x7, IJ-9x4x7 and IJ-8x4x7. EJ is EJ-32x4 or EJ-16x2. We selected these configurations as they represent a spectrum of mechanisms with varying storage requirements and coverage characteristics.

Coverage results are shown in Figure 5(b). As expected HJ results in increased coverage compared to its IJ and EJ constituents. In fact, for some programs, the resulting coverage is close or even exceeds (i.e., (IJ-8x4x7, EJ-16x2)) the sum of the coverage possible by the individual IJ or EJ when operating in isolation. This is because the IJ acts as filter reducing the blocks that are allocated in the EJ.

Overall, (IJ-10x4x7, EJ-32x4) HJ performs the best resulting in 75.6% average coverage (using a VEJ-32x4-8 resulted in 77% average coverage). However, even an (IJ-8x4x7, EJ-16x2) that requires much less storage yields a 65% average coverage. We have also experimented with an 8-way SMP. Due to space limitations, we summarize the results. In an 8-way SMP, snoop-induced misses account for a larger fraction of all L2 accesses, 76.4% on the average vs. 54.5% for the 4-way SMP. Moreover, average coverage becomes 79%.

The results of this section suggest that we can get high coverage with modestly sized JETTY organizations. We have seen that for some applications IJ organizations work better than EJ ones and vice versa. However, combining IJ and EJ mechanisms into an HJ improves coverage over all applications, often significantly. The best mechanism in terms of coverage is an (IJ-10x4x7, EJ-32x4) resulting in about 76% average coverage (on a similar system that does not use subblocking the coverage for this HJ was 68%). However, much smaller configurations result in competitive coverage. For example, an (IJ-9x4x7, EJ-32x4) yields about 74% average coverage.

4.4 Energy Measurements

In this Section, we report results on the energy savings for various L2 and JETTY organizations. We report energy savings both as a fraction of all snoop accesses and as a fraction of all L2 accesses. In this analysis, we take into account JETTY's energy dissipation including L2 replacement updates for IJ components. Moreover, we take into account the actual mix of reads and writes.

The energy reduction results are shown in Figure 6. We model two L2 organizations. The first (parts (a) and (b)) is energy optimized where the tag and data arrays are accessed

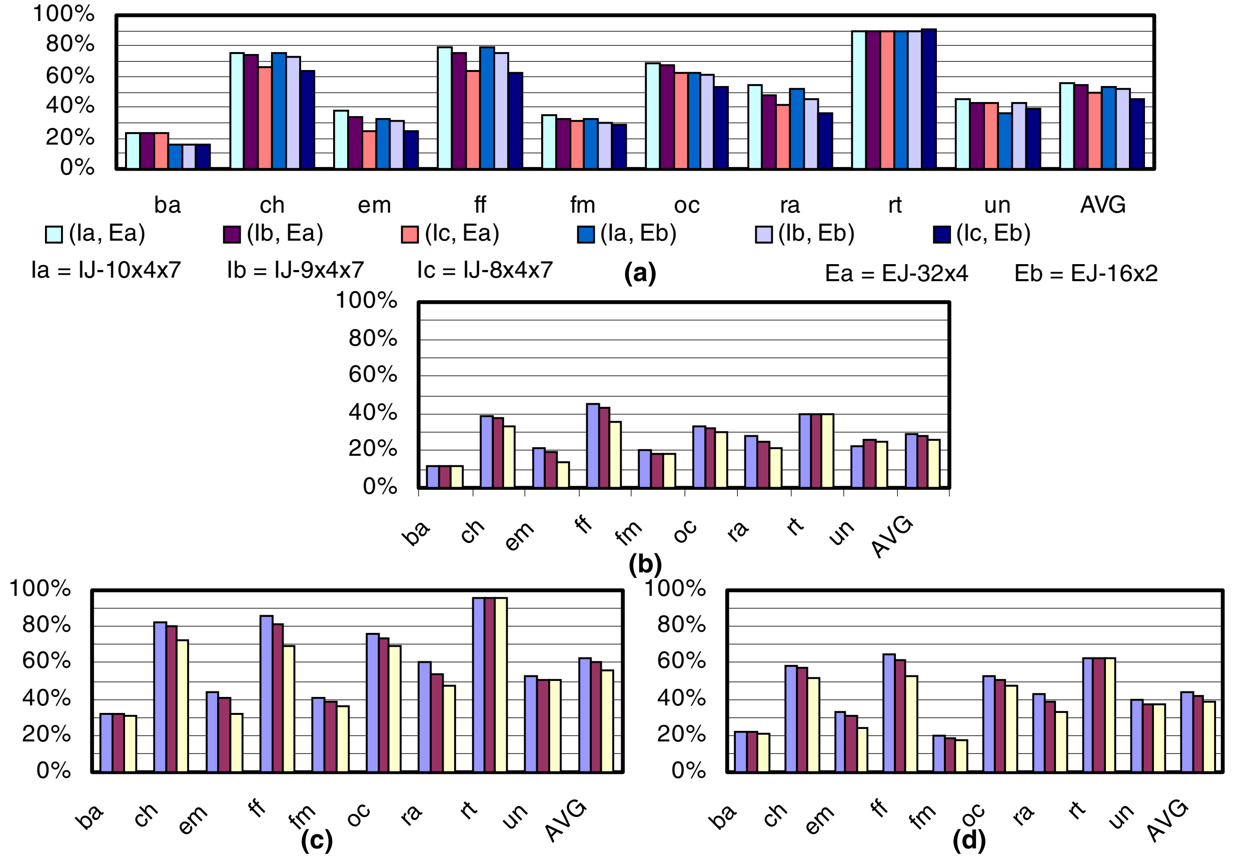


Figure 6: *Energy reduction with various* JETTY *organizations.* ***Serially Accessed Tag and Data Arrays:*** *(a) Over all snoop accesses. (b) Over all L2 accesses.* ***Tag and Data Arrays Accessed in Parallel:*** *(c) Over all snoop accesses. (d) Over all L2 accesses. The HJ methods listed in (a) are the same as in Figure 5(b). Part (a) uses the same HJ configurations as Figure 5(b). Parts (b), (c) and (d) are for (IJ-10x4x7, EJ-32x4), (IJ-9x4x7, EJ-32x4) and (IJ-8x4x7, EJ-32x4) from left to right.*

serially, as is done in Alpha 21164 [4] and in Intel Xeon [2]. The second (parts (c) and (d)) assumes that tags and data are accessed in parallel. We limit our attention to HJ organizations. We model all accesses to L2 and JETTY including L2 replacement information for updating the cnt-arrays. (Note that some differences may be observed compared to the analytical model of Section 2.1. Here we use the actual read/write reference counts along with writeback traffic.)

In part (a) we report energy reduction over all snoop accesses. We limit our attention to the HJ organizations we reported in Figure 5(b). The differences among the various HJ organizations are relatively small. (IJ-10x4x7, EJ-32x4) performs the best, offering a 56% energy reduction over all snoop accesses. Using the much smaller (IJ-8x4x7, EJ-32x4) we observe a 49% energy reduction. In general, energy reductions are correlated to snoop-miss coverage. However, even when coverage is close to 100% (e.g., in raytrace) we do not completely eliminate energy consumption as JETTY itself consumes energy. In fact, in *raytrace* where virtually all JETTY organizations offer the same coverage, we observe energy savings that are inversely proportional to JETTY's energy dissipation (closely related to its size). In the interest of space, in the rest of this section we restrict our attention to HJ organizations containing the EJ-32x4 only (first 3 bars of part (a)).

In part (b) we report energy reduction as a fraction over all L2 accesses. (IJ-10x4x7, EJ-32x4) results in a 30% energy reduction, while (IJ-9x4x7, EJ-32x4) results in a 29% energy reduction on the average.

Parts (c) and (d) report energy reduction over all snoops and all L2 accesses respectively. Here we model an L2 where the tag and data arrays are accessed in parallel (this might be done for reducing latency). Overall, energy reductions are higher compared to the serial L2 organization of parts (a) and (b). (IJ-10x4x7, EJ-32x4) now results in a 63% energy reduction on the average in the energy required by snoop-induced accesses. When all access are considered, this HJ organization results in a 41% energy reduction on the average.

5 Related Work

A number of previous studies have focused on architectural/microarchitectural techniques to reduce energy dissipation in the memory hierarchy [1,3,10,13,14,18,19,26,32]. Most of these techniques directly target reducing power dissipation induced by processor memory accesses — rather than snoop-induced accesses which are the focus of this paper. Many of the techniques propose using tiny energy-efficient devices to capture small program working sets and filter references to larger and more power-intensive structures such as L1 caches and TLBs. Other techniques focus on cache resizing to reduce energy dissipation (e.g., varying set-associativity [1] or the number of sets [19]). JETTY may easily co-exist with such optimizations and will still be valuable especially when the application requires use of all L2 cache resources.

Techniques that reduce tag array sizes (e.g., CAT [27], Seznec's tag indirection [23] and sectored tags [22]) can help reduce tag lookup power dissipation. While these techniques reduce the tag array size they also place restrictions on the block address distribution. Moreover, these techniques may impact L2 access latency.

6 Conclusion

In this work, we were motivated by the increasing importance of power dissipation in computer system design, and in particular for servers. Accordingly, we have proposed methods for reducing the power required to perform snoops on snoop-coherence, bus-based SMP servers. In particular, we introduced JETTY, a small structure placed on the backside (bus-side) of each L2. The JETTY acts as a filter preventing snoops that would miss in the L2 from percolating up in the hierarchy. Our method is speculative in nature (it may fail to filter some of the snoops that would miss) and reduces power on the average. In developing JETTY we were motivated by the relatively large fraction of snoop-induced L2 tag accesses that miss which we found to be 54% of all L2 accesses for a 4-way SMP and a set of commonly used shared-memory benchmarks.We described a number of alternative organizations that either record a subset of blocks that are not cached in the local L2 and/or a superset of the blocks that are. We have evaluated the potential of our proposed method and found that a relative inexpensive organization filters about 76% of all snoops that would miss on the average. The corresponding energy savings were 30% measured as a fraction of all L2 accesses (both tag and data arrays). A hybrid JETTY comprising an IJ with four 512-entry subarrays and a 32-set, 4-way set associative EJ resulted in energy savings of 29%.

Filtering of snoops that would miss is only one type of power-related optimizations that might be possible. Other possibilities may exist. Nevertheless, JETTY represents a valuable first step toward developing power-optimizations for snoop traffic in SMP systems. Moreover other applications of *snoop-filtering* structures such as JETTY might be possible including performance and cost optimizations.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. We also thank Seda Ogrenci for her help with the CMOS process parameters. This research was supported by DARPA under contract F33615-00-C-1631.

Appendix A - Snoop Miss Energy Model

We present an analytical model for snoop-induced L2 access energy consumption. We used this model only to evaluate the relative trends in snoop-induced energy consumption with varying cache hit rates and sharing degree (in Section 2.1). As such, we make several simplifying assumptions as compared to the more detailed model we use in Section 4.4. We denote the energy required per access to the tag and data arrays by *TAG* and *DATA* respectively. The number of processors *Ncpu*, the local hit rate *L* and the remote hit rate *R* (defined in Section 2.1). This model ignores L2 writebacks. Moreover, while not shown, we assumed a 1 to 2 write to read distribution. In our analysis of Section 4, we include both L2 writebacks and the actual distribution of reads and writes. The model (for presentation clarity, we omit the read/write terms) is as follows:

$$TagSnoopMiss = TAG \times (Ncpu - 1) \times (1 - L) \times (1 - R)$$

$$Data = DATA \times (1 + (Ncpu - 1) \times (1 - L) \times R))$$

$$SnoopE = TagSnoopMiss + TAG \times (Ncpu - 1) \times (1 - L) \times R$$

$$TagAll = SnoopE + TAG \times (1 + (1 - L))$$

$$SnoopMissE = \frac{TagSnoopMiss}{Data + TagAll}$$

TagSnoopMiss is the energy required by snoop-induced accesses that miss. A local access will generate a snoop with probability (1-L), this will result in (Ncpu - 1) remote tag lookups, each of which will miss with (1-R) probability. *SnoopE* is the energy consumed by all snoop-induced accesses to the tag array. In addition to *TagSnoopMiss* it also includes the energy required by snoop-induced accesses that hit. *Data* is the energy required by data array accesses. Every local access eventually accesses the data array (hits immediately, misses eventually when the requested data returns), hence the 1 term. Moreover, the data array is accessed when a snoop-induced access hits ((Ncpu-1) x (1-L) x R term). This is a pessimistic assumption as a snoop hit may only require changes to the status bits (read on an exclusive block if the E state exists), or no changes at all (read to a shared block). *TagAll* is the energy required by *all* tag accesses (local and snoop-induced). Besides the energy required by snoop-induced accesses, the tag array is accessed once for local accesses that hit, and once more for every local miss to update the tag information. Note that this model ignores changes to the status bits when a snoop-induced access hits. The last equation expresses the energy required by snoop-induced accesses that miss as a fraction of all L2 accesses.

References

[1] D. H. Albonesi. Selective cache ways. *In Proc. Intl. Symposium on Microarchitecture*, Nov. 1999.

[2] B. Bateman, C. Freeman, J. Halbert, K. Hose, and E. Reese. A 450Mhz 512kB Second-Level Cache with a 3.6GB/s Data Bandwidth. In *IEEE Intl. Solid-State Circuits Conference*, 1998.

[3] N. Bellas, I. Hajj, C. Polychronopoulos, and G. Stamoulis. Architectural and compiler support for energy reduction in the memory hierarchy of high performance processors. *In Proc. Intl. Symposium on Low Power Electronics and Design*, 1998.

[4] W. J. Bowhill and et al. Circuit Implementation of a 300Mhz 64-bit Second Generation Alpha CPU. *Digital Journal, vol 7., 1995.*

[5] J. Cong, L. He, Y. Khoo, and Z. Pan. Interconnect design for deep submicron ICs. *Invited Tutotal, IEEE Intl. Conference on Computer Aided Design*, Nov. 1997.

[6] K. Diefendorff. Xeon Replaces Pentium Pro. *Microprocessor Report, vol. 12, no. 9*, June 1998.

[7] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang. Protocol verification as a hardware design aid. In *1992 IEEE Intl. Conference on*

Computer Design: VLSI in Computers and Processors,, 1992.

[8] G. Gerosa, M. Alexander, J. Alvarez, C. Croxton, M. D'Addeo, A. R. Kennedy, C. Nicolette, J. P. Nissen, R. Phillip, P. Reed, H. Sanchez, S. A. Taylor, and B. Burgess. A 250-MHZ 5-W PowerPC Microprocessor with On-Chip L2 Cache Controller. *IEEE Journal of Solid-State Circuits*, 32(11), Nov. 1997.

[9] Intel Corp. PENTIUM II XEON PROCESSOR AT 400 and 450 Mhz. *Order No: 243770-003*, June 1998.

[10] T. Juan, T. Lang, and J. J. Navaro. Reducing TLB Power Requirements. *In Proc. Intl. Symposium on Low Power Electronics and Design*, Aug. 1997.

[11] M. B. Kamble and G. Ghose. Analytical Energy Dissipation Models for Low Power Caches. *Proc. Intl. Symposium on Low Power Electronics and Design*, Aug. 1997.

[12] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker. Performance characterization of a quad pentium pro smp using oltp workloads. In *Proc. of the 25th Annual Intl. Symposium on Computer Architecture*, June 1998.

[13] J. Kin, M. Gupta, and W. H. Mangione-Smith. The Filter Cache: An Energy Efficient Memory Structure. In *Proc. Annual Intl. Symposium on Microarchitecture*, Dec. 1997.

[14] U. Ko, P. T. Balsara, and A. K. Nanda. Energy Optimization of Multilevel Cache Architectures for RISC and CISC Processors. *In Proc. Intl. Symposium on Low Power Electronics and Design*, 1998.

[15] A.-C. Lai and B. Falsafi. Memory sharing predictor: The key to a speculative coherent DSM. , In *Proc. Annual Intl. Symposium on Computer Architecture*, May 1999.

[16] A. Moshovos and G. Sohi. Streamlining inter-operation communication via data dependence prediction. In *Proc. Annual Intl. Symposium on Microarchitecture-30*, Dec. 1997.

[17] S. S. Mukherjee, S. K. Reinhardt, B. Falsafi, M. Litzkow, S. Huss-Lederman, M. D. Jill, J. R. Larus, and D. A. Wood. Wisconsin Wind Tunnel II: A Fast and Portable Architecture Simulator. *Workshop on Performance Analysis and its Impact on Design*, June 1997.

[18] R. Panwar and D. Rennels. Reducing the frequency of tag compares for low power I-Cache design. *In Proc. Intl. Symposium on Low Power Electronics and Design*, 1995.

[19] M. D. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-Vdd: A circuit technique to reduce leakage in cache memories. In *Proc. Intl. Symposium on Low Power Electronics and Design,* July 2000.

[20] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso. Performance of database workloads on shared-memory systems with out-of-order processors. In *Proceedings of the Eighth Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*, Oct. 1998.

[21] Semiconductor Industry Association. Intl. technology roadmap for semiconductors. 1999.

[22] A. Seznec. Decoupled sectored caches: Conciliating low tag implementation cost and low miss ratio. In *Proc. 21st Annual Intl. Symposium on Computer Architecture*, Apr. 1994.

[23] A. Seznec. Don't use the page number, but a pointer to it. In *Proc. Annual Intl. Symposium on Computer Architecture*, May 1996.

[24] D. Singh and V. Tiwari. Power challenges in the internet world. Cool Chips Tutorial in conjunction with the 32nd Annual Intern ational Symposium on Microarchitecture, Nov. 1999.

[25] M. S. Squillante and E. D. Lazowska. Using processor-cache affinity information in shared-memory multipr ocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):131–143, Apr. 1990.

[26] C.-L. Su and A. M. Despain. Cache design trade-offs for power and performance optimization: A case study. *In Proc. Intl. Symposium on Low Power Electronics and Design*, 1995.

[27] H. Wang, T. Sun, and Q. Yang. CAT – caching address tags: A technique for reducing area cost of on-chip caches. In *Proc. Annual Intl. Symposium on Computer Architecture*, June 1995.

[28] W.-D. Weber and A. Gupta. Analysis of cache invalidation patterns in multiprocessors. In *Proceedings of the Third Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, Apr. 1989.

[29] K. M. Wilson, K. Olukotun, and M. Rosenblum. Increasing cache port efficiency for dynamic superscalar microprocessors. In *Proc. 23rd Annual Intl. Symposium on Computer Architecture*, May 1996.

[30] S. Wilton and N. Jouppi. An Enhanced Access and Cycle Time Model for On-Chip Caches. *Technical Report 93/5, Digital Western Research Laboratory*, July 1995.

[31] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. Annual Intl. Symposium on Computer Architecture*, June 1995.

[32] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance i-caches. *In Proc. Intl. Symposium on High-Performance Computer Architecture, Jan 2001.*