JETTY: Reducing Snoop-Induced Power Consumption in Small-Scale, Bus-Based SMP Systems

Andreas Moshovos, Gokhan Memik, Gaurav Mittal, Amirali Baniasadi, Alok Choudhary

Technical Report No. CPDC-TR-2000-05-007
Center for Parallel and Distributed Computing
May 30, 2000

Center for Parallel and Distributed Computing
Department of Electrical and Computer Engineering
Northwestern University
Technological Institute
2145 Sheridan Road, Evanston, IL 60208

JETTY: Reducing Snoop-Induced Power Consumption in Small-Scale, Bus-Based SMP Systems

Andreas Moshovos, Gokhan Memik, Gaurav Mittal, Amirali Baniasadi, Alok Choudhary
Electrical and Computer Engineering Department
Northwestern University
{moshovos, memik, mittal, amirali, choudhar}@ece.northwestern.edu

Abstract

We propose methods for reducing the power required for handling snoop requests in small-scale, snoop-coherence, bus-based SMP systems. Observing that a large fraction of snoops do not find copies on all other caches, we introduce JETTY, which is a small, cache-like structure. A JETTY is placed in-between the bus and the L2 backside of each processor where it acts as a filter for snoop requests. In particular, it filters the vast majority of snoops that would not find a locally cached copy. We propose a number of alternative JETTY methods that operate by identifying either a subset of non-locally-cached blocks or a superset of locally cached blocks. We demonstrate that for a set of parallel applications and a 4-way SMP system, relatively small JETTY structures can filter up to 77% of all snoops that would miss on the average. This resulted in average power reduction of 41% measured as a fraction of the power required for all snoop-induced tag-array accesses.

1 Introduction

After several decades of research and development in computer architecture much of our accumulated expertise lies in designing computing systems for a desired performance level while taking cost and complexity into consideration. In the recent years, however, power consumption is emerging as an additional, similarly important consideration. Reduced power consumption can greatly improve utility in mobile applications by allowing longer operation times. However, power is also becoming important in desktop/server applications. Here the concerns include feasibility, reliability and cost. While, initially, power dissipation was not a concern for commodity systems, it has been increasing steadily. As a result, modern processors utilize optimizations to keep power dissipation within acceptable limits, e.g., [17, 33]. Based on the current projections on the power dissipation in future systems further power-optimizations will most likely be absolutely necessary [29].

In this work, we are concerned with power optimizations for small-scale snoop-coherence bus-based SMP systems (or, just SMP for clarity). Such systems are becoming increasingly popular in part as a result of their cost-effectiveness [37], of popular software that can exploit them (e.g., database systems, web-servers, etc.) and of increased understanding on how to program and use them, e.g., [3, 15]. Even larger-scale parallel systems are typically build out of small-scale SMP nodes, e.g, [24]. In SMP systems, increasing processor and system integration levels magnify power-dissipation concerns. Moreover, power-optimizations may be essential for future, single-chip implementations of SMPs [18,20,7] where multiple processors and part of the associated memory hierarchy may lie on the same die. Even today, processors used as building blocks for such systems include temperature monitoring hardware so that appropriate action can be taken when necessary, e.g., [13].

Power-optimizations can be attempted at various levels, starting at the high-level programming language/ application down to the semiconductor process. Moreover, there is a multitude of structures where power optimization opportunities exist. Developing an understanding of power-related tradeoffs requires careful examination of these opportunities.

In this paper we are concerned with micro-architectural techniques that are orthogonal to traditional power-reduction techniques (e.g., voltage and frequency scaling). Specifically, we present methods to reduce the average power required by the memory traffic on the bus of an SMP system. In such snoop-coherence-based systems, accesses from one processor may have to check whether copies of the referenced memory location exist in other caches. This is typically done by checking the tags of the appropriate caches in *all* other processors. This is fairly appropriate when power is unimportant. However, when power is a concern possibilities for reduced-power optimizations exist.

We observe that more than often, copies either do not exist or exist only in some of the caches. We exploit this phenomenon by introducing JETTY. JETTY is a tiny structure placed on the bus-side at each processor. JETTY filters out the majority of bus-induced tag lookups that would otherwise miss in the local processor hierarchy. It

operates as follows: When a snoop request is received, it is first sent to the JETTY. The JETTY responds either that no copies exist (this is guarantee) or that copies *may* exist. Only in the latter case it is necessary to access the rest, much more power demanding tag-arrays. We describe a number of possible JETTY variants, which are speculative in nature: They indicate that a block is either *not* cached (guarantee) or that it *may* be cached in the L2. That is, they identify a *subset* of blocks that are not cached and a *superset* of blocks that are cached.

JETTY reduces power on the average provided there is a sufficiently large fraction of snoops that miss in remote caches. In a worst case scenario (for example, when the vast majority of bus-snoops hit in all local hierarchies) JETTY may increase power consumption. While this increase will be very small (as JETTY is much smaller than the rest of the tag hierarchy), care must be taken to keep power dissipation within operational limits. Existing processors already include temperature monitoring hardware and the mechanisms necessary to take action when appropriate, e.g., [13]. A number of remedies are possible including frequency and/or voltage scaling [5, 16] and shutting down parts of the system [25]. Even so, JETTY aims at minimizing the probability that such measures will be necessary, effectively maintaining a lower operating temperature which may lead to higher reliability [13]. As we show in Section 5, for the parallel programs we studied JETTY results in significant power savings. However, it is likely that the potential 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.

Overall, snoop traffic power dissipation represents only a fraction of the total power consumed. Nevertheless, a study of potential optimizations such as those we describe is justified for the following reasons: As caches increase in size, the absolute amount of power required for their operation will also increase. This is especially a concern for single-chip multiprocessor systems and of processors that integrate large on-chip caches. Moreover, as other power-optimization techniques are perfected, tag-related optimizations like JETTY will increase in importance. Finally, it is likely, that similarly to performance optimizations, a plethora of power-optimizations will be needed at multiple levels 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. In Section 3, we discuss the operation of JETTY and present several alternative organizations. In Section 4, we comment on related work. We present experimental results in support of the utility of our methods in Section 5. Finally, in Section 6 we summarize our findings and offer concluding remarks.

For clarity, we use the acronym SMP to refer to snoop-coherence, bus-based, small-scale symmetric multiprocessor systems.

2 A Snoop-Traffic Power Optimization Opportunity

In this section we review the operation of typical SMP systems to motivate the JETTY method. We also present experimental evidence showing that a set of parallel programs exhibit part the behavior necessary for JETTY's success.

Figure 1(a) illustrates a typical SMP system consisting of 3 processors with local caches. Shown are a write-buffer (WB) and an L1 and an L2 data caches per processor. For clarity we omit the instruction hierarchy and any other buffers that may exist between adjacent levels. The processors are connected to a common bus, which is also connected to the main memory subsystem. The JETTY method aims at reducing energy required for memory requests that appear on the common bus. To explain our method we start by reviewing how memory requests are typically handled in such a system. In the example shown, CPU1 performs a request on address "a", which misses in the local memory hierarchy. As a result, the request appears on the common bus (action 1). The other two processors snoop this request (action 2). As part of this action all appropriate local tag arrays are probed for a possible match. In our example this includes the L2 tags and the WB (we assume inclusion of L1 in L2). The cycle is complete when all CPUs respond (action 3) so that appropriate action can be taken. As shown, a copy of "a" exists in the L2 of CPU2, however, no copy is found in CPU3's caches. (For example, this scenario is possible if a process in CPU1 is using "a" to communicate values to a process running in CPU2.)

As the example shows, *all* L2 tag-arrays in *all* CPUs consume power even though not all of them have a copy. The larger the L2 caches are, the larger their tag arrays will be, and as a result the higher the power expended on snoops that miss. Provided that many snoops miss, there is an opportunity for reducing power consumption. JETTY capitalizes on this opportunity by introducing a small structure placed in-between the L2 and the bus in every processor. A JETTY enhanced system is shown in Figure 1(b). The intention is to filter most requests that would anyhow miss in the L2 blocking them from reaching the much more power-demanding L2 tag-array (however, assuming that the WB may contain addresses not yet allocated in the L2, the WB tags still have to be snooped). As shown in the example, the JETTY in CPU3 determines that no local copies exist avoiding probing

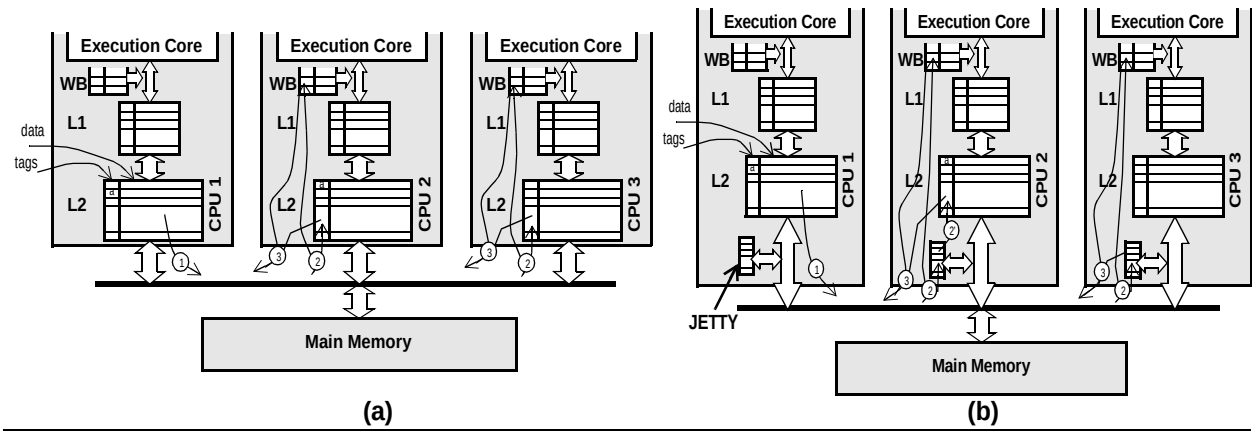


Figure 1: *(a) Traditional snoop-coherence, bus-based SMP system: All L2 tag-arrays consume power for snoops. (b) JETTY enhanced system: the local JETTY filters snoops that would miss. Only if a hit is likely, the L2 tag-array is probed consuming power. Otherwise, power consumption is limited to that required by the local JETTY and the writeback buffer array (we assume inclusion of L1 in L2).*

the much larger L2-tag array. While JETTY will not filter snoops to the WB, the WB will be relatively small as compared to the L2 tag array in typical systems.

For JETTY to be successful there are two primary requirements:

1. A large enough fraction of L2-snoops should result in a miss. Note that we are referring to L2-snoops and not bus-snoops. It is known that a large fraction SPLASH2 find remote cached copies [36]. However, as we show in Section 5, even if cached copies exist in *some* caches, they do not always exist in *all* caches.
2. It should be possible to identify most of these would be misses using a small enough structure.

In Section 5, we demonstrate that this is the case for a set of parallel applications via simulation. Since building working coherence protocols is already complex enough [11], it is desirable to keep any additional complexity introduced to a bare minimum. In the organizations we propose no coherence-related state information (other than presence, i.e., valid) is kept with any of the JETTY blocks. The JETTY simply filters requests for non-existing blocks. Since JETTY appears in series with the L2 it will increase response latency for non-filtered snoops. However, given that JETTY is very small, it is unlikely that the additional latency introduced will impact the typically much slower bus-clock cycle (otherwise, significant performance degradation may be observed making JETTY less attractive). Finally, a potential benefit of our approach is that reduces the number of L2 tag-array requests. This may be beneficial for performance when the number of L2 tag-array ports is limited.

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 in an attempt to get the best of the other two. All variants are speculative in nature: They all 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

The *Exclude-JETTY* (EJ) keeps a record of blocks that have been snooped recently and missed in the local L2 and are still not cached, i.e., a *subset* of blocks that are not locally cached. The EJ is a small array containing (TAG, valid-bit) pairs. If a match is found in the EJ then this is a guarantee that the block is not cached locally. An entry is allocated when a remotely generated snoop results in a local L2 miss. Subsequent accesses to the same block will be successfully filtered so long as the block is not evicted from the EJ. An EJ entry is removed (valid bit reset) when a local miss results in allocation of the corresponding block. This can be done simply by observing the bus traffic generated by the local hierarchy (i.e., the EJ can operate on the backside of the L2).

The EJ is well suited for scenarios where *some* of the processors are heavily using *a small portion* of their address space in a way that results in frequent bus activity. This would be the case, for example, in producer-consumer scenarios of small data structures. Other pro-

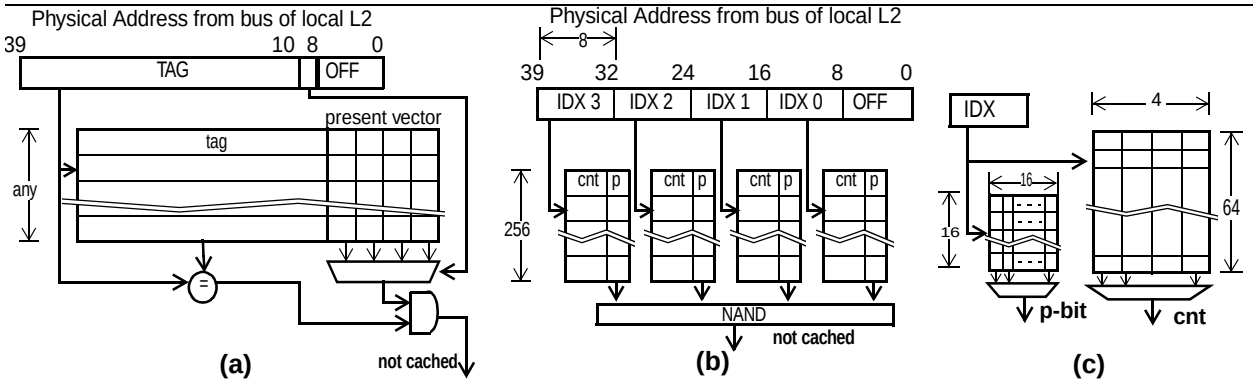


Figure 2: *(a) A vector-exclude-JETTY: it contains information about recent snoops that missed in the local L2. A block is not cached if its tag is matched in the EJ and the corresponding present-vector bit is set (the i-th present-vector-bit corresponds to address TAG+i). (b) An include-JETTY: The upper 32 relevant bits (assuming 256 byte blocks and a 40-bit PA space) of an incoming physical address are split into four 8-bit parts. These are then used in parallel to index into four sub-arrays. Each matching entry reports a presence bit (p-bit) and the exact count of blocks currently present in the L2 that match the corresponding bit combination. A block **cannot** be locally cached if **at least one** of the matching p-bits **is zero**. However, a block **may** be locally cached if **all** matching p-bits **are set**. The count fields are used only on block allocation or de-allocation to keep the p-bits coherent (read text). (c) Example showing power-optimized EJ sub-arrays. Separate p-bit and cnt arrays are used per each 256-entry sub-array of part (b).*

cessors *not* participating in this activity can filter the corresponding snoops by simply recording the blocks involved. Also, EJ will successfully filter snoops when, for example, only some of the processors are accessing the same array (or other data structure) elements in the same order and close in time. Consider, for example, a 4-way SMP and a scenario where CPU1 and CPU2 are traversing the same array *a* in lock-step with CPU1 accessing element *a[i]* first, followed immediately by CPU2 accessing the same element. When CPU1's access to *a[i]* appears on the bus, the EJs on CPU3 and CPU4 will record this address. Then, when CPU2 accesses the same block, the EJs in CPU3 and CPU3 will successfully filter out the snoop.

The number of EJ entries limit its utility. While increased entry count would probably result in a higher fraction of filtered snoops, it would also result in increased size and reduced power savings. However, by exploiting program behavior, it is possible to get increased coverage of snoop misses without a proportional increase in EJ entries. Observing that often snoop-misses are clustered in the memory space, we developed the *Vector-Exclusive-JETTY*, or VEJ. Here, we use a (TAG, present-vector) pair to encode presence for a chunk of consecutive blocks. The present-vector, or PV is a *n*-bit mask indicating which blocks starting from *TAG* and ending with *(TAG+n - 1)* are currently not cached in the L2. An example is shown in Figure 2(a) 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 in the EJ, 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

An alternative to recording blocks that *are not* locally cached (as in EJ), is to keep information about those that *are*. This is what the Include-JETTY (IJ for short) does. The IJ contains information that identifies a *superset* of the blocks currently cached in the L2. If no match is found in the IJ, then this is a guarantee that no matching block exists in the L2.

The various IJ organizations we studied are all variants of the one shown in Figure 2(b). We explain its operation using an example configuration 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) which are then 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-bits of the 1st entry (entry 0) and the 256th entry match blocks with addresses of the form *0x00xxxxxx* and *0xFFxxxxxx* respectively. If *any* of the four p-bits retrieved for a block address are zero, then no L2 block matches this address. If *all* of p-bits are non-zero, then a block *may* be locally

cached and 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.

Since missing in the IJ implies that a block is not in the L2, it is imperative to keep its 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 needs to be 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 on 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 the L2 (no additional accesses are required). A separate tag-sized set of wires can be used to communicate this information to the IJ (actually the set-index bits are not required if this information is communicated when the address of the incoming to be cached block appears on L2's backside). 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 2(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 indexes results in better accuracy. However, an extensive investigation of IJ index generation schemes is beyond the scope of this paper.

Noting that storage requirements are just an indication of power consumption, in general, the ratio of the storage requirements in bits of an IJ vs. a regular tag array is:

$$Ratio = \frac{\sum_i 2^{m_i} \times (lg(Sets \times Assoc) + 1)}{Sets \times Assoc \times (PA - lg(BlockSZ) - lg(Sets) + a)}$$

Where *Sets*, *Assoc*, and *BlockSZ* are the number of sets, the associativity and the block size (in bytes) of the L2 cache, *PA* bit-width of physical addresses, m_i are the IJ sub-array index bit-lengths, and a is at least 2 and 3 for MOSI and MOESI state encoding respectively (ignoring error-detection/correction overheads).

It is important to emphasize that when an external snoop probes the IJ only the p-bits are read. The *cnt* fields are read and updated less frequently. Accordingly, to reduce power 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 2(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.

Some ISAs define cache-related operations to facilitate cache management in software (e.g., [10]). In the presence of such operations, it is important to maintain the information kept in the IJ coherent with that in L2. A possible solution is to provide primitives that allow software inspection and control of the information present in the JETTY. However, in this case, care must be taken to weight the power benefits of JETTY in normal operation vs. the additional power required for updating it on multiple-block cache actions. For our purposes, we limit our attention to systems when multiple-block cache actions are not supported in hardware.

3.3 Hybrid-Jetty

The IJ contains aggregate information about what is cached locally and as we show in Section 5, it is fairly accurate most of the time. However, there are cases where a small set of frequently snooped blocks defies identification by an IJ. At the same time, the EJ is well suited for keeping track of a small number of blocks that are not currently cached. An obvious alternative is to combine the two methods to increase accuracy. In this *Hybrid-JETTY*, or HJ organization both an EJ and IJ operate in parallel. When either of the two indicates that no match is possible we avoid accessing the L2 tag array. Entries are allocated in the EJ *only* when the IJ fails to filter them (as opposed for all local snoop-misses as in the original EJ method). As we show in Section 5, a hybrid method outperforms the other two both in accuracy and in power reduction.

4 Related Work

Any of the plethora of techniques to minimize false sharing and to maximize cache utilization will reduce bus-traffic and also reduce the power dissipated by snoops. Here we limit our attention to techniques that either aim at reducing the power consumption of caches or to minimizing the storage requirements for tag-arrays. There is a plethora of other power consumption reduction techniques.

A number of techniques have been proposed to reduce the power consumption of instruction and data caches and TLBs [8,14,4,22,19,23,28,32]. In most cases, additional caching levels are introduced (often employing intelligent caching strategies) to reduce access frequencies for higher-level, much larger and hence more-power consuming caches or TLBs. JETTY is mostly orthogonal to these techniques as they minimize power for accesses originating from the local processor. Since, the size of higher-level caches remains unchanged, these techniques may not reduce power for snoops. Albonesi has also looked at adjusting the associativity of the L1 data cache by selectively shutting down parts of the cache[6]. JETTY may easily co-exist with such optimizations and will still be valuable especially when the application requires use of all L2 cache resources.

We can reduce the power required by tag-lookups by using alternative tag organizations. Two closely related techniques are the CAT method [34] and a method proposed by Seznec [31]. In these schemes, a level of indirection is introduced in determining cache block tags. A first level tag table provides part of the tag and an index pointing to a second level table providing the rest of the tag. Related to these techniques are sectored caches and Seznec's decoupled sectored caches [30]. While these techniques reduce the space required for the tag-array they also place restrictions on the block address distribution that can be concurrently in the cache. Moreover, they may increase L2 access latency. It is likely that JETTY will still remain a relatively small fraction of the resulting tag-array.

5 Experimental Analysis

In this Section, we evaluate the effectiveness of various JETTY organizations. In Section 5.1, we discuss our methodology including the benchmarks and simulation environment. In Section 5.2, we demonstrate that a large fraction of snoop-induced L2 tag accesses results in a miss. As we explained in Section 2, this provides an indication of our method's potential. In Section 5.3, we measure the accuracy of various JETTY organizations that are based on the methods described in Sections 3.1 through 3.3. In Section 5.4, we measure the power reduction possible for a set of the best performing JETTY organizations. Finally, in Section 5.5, we report accuracy and power reduction results with smaller L2 caches sizes.

5.1 Methodology

In our experiments we have used the *Barnes, Fft, Fmm, Lu, Radix* and *Ocean* shared-memory applications from the SPLASH-2 benchmark suite [36], and *Unstructured* [27] and *Em3d* [12]. Table 1 provides additional information about these shared-memory applications including the input data set (we will explain the other columns shortly).

To evaluate the utility of our approach we have used the Wisconsin Wind Tunnel 2 simulator [26] to collect snoop activity traces and memory reference statistics. Since only the binary distribution of WWT2 was available to us, we have implemented a separate simulator for simulating the JETTY methods. The simulator accepts the WWT2 generated traces and simulates all L2 caches with and without various JETTY mechanisms (simulating of L2 caches is necessary to obtain block replacement information). Our base configuration is a 4-way SMP. Each processor has an 1MB direct–mapped, 64-byte L2 cache. Each L2 cache contains 8 banks. Using the CACTI tool [35] we have determined that this is optimal for the particular configuration. WWT2 assumes perfect instruction caches, cannot simulate caches of higher associativity and does not simulate L1 caches. The absence of an L1 can have an effect on snoop activity. In future work, we will modify WWT2 to model an L1-L2 on-chip hierarchy. Since the instruction footprint of these applications is very small it is unlikely that instruction misses would significantly affect our results. The resulting L2 tag array accesses and L2 tag array accesses induced by snoops are reported in columns 3 ("L2 accesses") and 4 ("Snoop Accesses") of Table 1 respectively and in millions. The figure for the L2 accesses includes snoop-induced accesses and is reported directly by WWT2.

For most of the evaluation, we restrict our attention to 1M L2 caches. In Section 5.5, we report results with 512k L2 caches. To measure power consumption we have adapted the analytical model developed by Kamble and Ghose [21]. 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. In our experiments we have assumed a 0.18μm technology with the characteristics reported in [9].

5.2 Snoop Activity Measurements

In this Section, we demonstrate that a large fraction of snoop-induced L2 tag accesses result in a miss. This metric provides an indication of JETTY's potential. For this purpose we have measured the total number of snoop-induced L2 tag accesses and the fraction of those that result in a miss. The results are shown in columns 4 ("snoop accesses") and 5 ("Snoop Misses") of Table 1. Since we assume a 4-way SMP system, every snoop request results in 3 remote snoop accesses (there are three other processors). As shown, with the exception of *unstructured*, most (66% or higher) snoop accesses

Application	Input Data	L2 Accesses	Snoop Accesses	Snoop Misses
Barnes	*4K particles*	96.01 M	16.79 M	73.3%
Em3d	*8K nodes, 20% remote*	133.98 M	46.85 M	66.1%
Fmm	*16K particles*	583.44 M	11.03 M	72.3%
FFT	*256K data points*	60.24 M	9.94 M	75.3%
Radix	*512K keys*	14.35 M	1.96 M	75.0%
Ocean	*130 x 130 ocean*	46.32 M	4.19 M	76.1%
Unstructured	*mesh 2K*	1693.32 M	406.54 M	34.4%
Lu	*256x256 matrix, 16x16 blocks*	29.69 M	0.25 M	72.0%

Table 1: *Applications used in our experimental studies. Reported are the input parameters, the resulting L2 cache accesses (Millions), the number of L2 tag accesses due to snoops (in Millions) and the fraction of those that miss. Since we do not simulate L1 caches, the L2 tag access figure includes all memory references (including those that would hit in an L1).*

result in a miss. On average, over all program studied about 68% of all snoop accesses results in a miss. This result suggests that there is potential for JETTY to filter a large fraction of snoop access.

5.3 Snoop Miss Coverage

In this Section, we evaluate the accuracy of various JETTY organizations. For this purpose we utilize *snoop miss coverage*, or simply *coverage* as our metric. We define *coverage* as the fraction of snoop-induced L2 tag lookups that are filtered by a given JETTY mechanism. Higher coverage implies greater potential for power reduction. However, coverage in itself is not a direct metric of power. We have to also take into account the power consumed by the corresponding JETTY mechanism. (For example, we could get 100% coverage by utilizing a copy of the L2 tag array. But then power consumption would increase.) We do so in Section 5.4.

We present coverage measurements first for EJ and VEJ, then for IJ and finally, for HJ organizations (see Section 3 for a description).

5.3.1 Exclude-JETTY

We have experimented with eight different EJ configurations varying both the number of sets and the associativity of the storage array. For clarity, we use an EJ-SxA naming scheme to refer to an S-set and A-way set associative EJ organization (S * A total number of entries). We have experimented with structures having 32, 16, 8, and 4 sets and 2-way and 4-way associativity. Figure 3(a) reports coverage for these configurations. Notably, EJ organizations work fairly well only for *Barnes*, *Em3d* and *Unstructured* while they fail for all others. As expected using larger EJ organizations with higher associativity results in increased coverage. The EJ-32x4 results in about 14% coverage on the average and coverage is relatively low for all organizations.

5.3.2 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. For clarity, we use a VEJ-SxA-V naming scheme, where V is the bit-length of the presence vector and S and A are the sets and associativity of the array. The results are shown in Figure 3(b). EJ-32x4 and EJ-16x4 are included for convenience. Using vectors increases coverage considerably for *Barnes* and *Unstructured* suggesting that some snoop misses exhibit spatial locality. However, it is possible for coverage to decrease compared to a similar EJ when the number of entries is kept the same. This is observed for *Em3d*. Note that a VEJ and a 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 for the VEJ. This also explains why the 4-bit vector VEJ out performs its 8-bit counterpart in *Em3d.* On the average, the coverage increase by VEJ is about 2% for the VEJ-32x4-8 over the EJ-32x4.

While the results for both EJ and VEJ organizations seem discouraging, we will see that in hybrid organizations their coverage increases considerably.

5.3.3 Include-JETTY

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

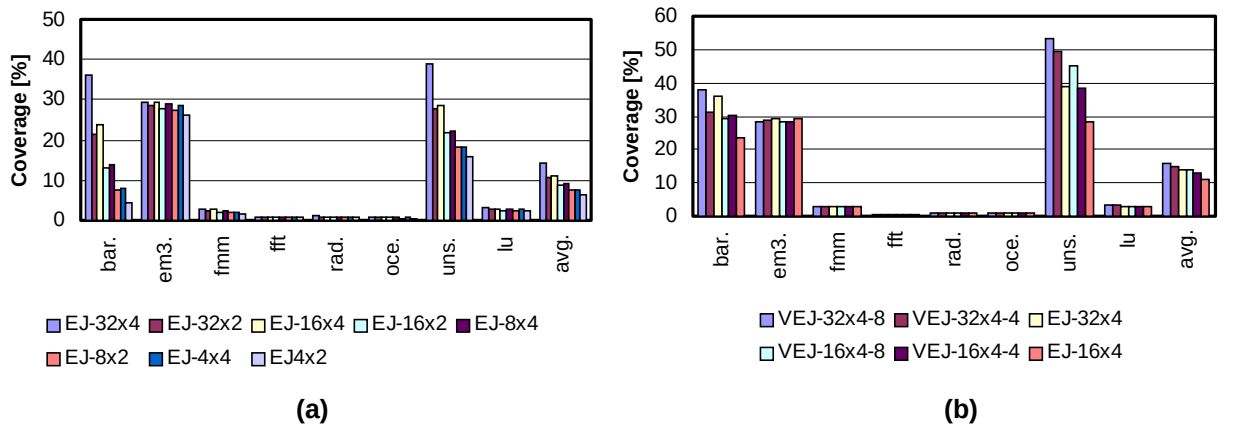


Figure 3: *(a) Exclusive-JETTY coverage. Results are shown for EJ configurations with 32, 16, 8 and 4 sets and of 2-way or 4-way set associativity. Configurations are named as EJ-SetsxAssociativity. (b) Vector-Exclusive-JETTY coverage. Results are shown for 32-set and 16-set 4-way set associative organizations with 8- or 4-bit presence vectors. Configurations are named as VEJ-SetsxAssociativityxVectorLength.*

the appropriate section of the PA to each sub-array.

We have experimented with the following organizations: IJ-10x4x7 (four 1K-entry sub-arrays), IJ-9x4x7, IJ-8x4x7, IJ-7x5x6 and IJ-6x5x6 (five 64-entry sub-arrays). Recall that for remote snoops, only the p-bit section of each sub-array needs to be accessed. The results are shown in Figure 4(a). The IJ-10x4x7, which is the largest IJ evaluated performs the best resulting in 56% coverage on the average. However, the IJ-9x4x7 (which has half the storage requirements of IJ-10x4x7) performs fairly well resulting in about 50% average coverage. For some programs using a larger number of smaller sub-arrays results in better coverage. For example, IJ-7x5x7 (five 128-entry sub-arrays) outperforms even IJ-9x4x7 (four 512-entry 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.

Interestingly, there is no direct correlation among IJ and EJ behavior. Comparing with the results of Figure 2, it can be seen that there are applications where IJ organizations perform fairly well and where EJ organizations fail (e.g., *Lu* and *Fft*) and vice versa (e.g., *Barnes*). This suggests a potential synergy of the two methods and serves as the motivation for experimenting with hybrid organizations.

5.3.4 Hybrid-JETTY

As explained in Section 3.3, an HJ contains both an IJ and an EJ operating in parallel. However, we have found that it is beneficial to update the EJ only when the IJ fails to filter a snoop that misses. We have experimented with 9 methods which are derived by combining various IJ and EJ organizations. For clarity, we use an (IJ, EJ) naming scheme, where IJ is any of IJ-10x4x7, IJ-9x4x7 and IJ-8x4x7 and where EJ is any of VEJ-32x4-8, EJ-32x4 and EJ-16x2. We selected these configurations as they represent a spectrum of mechanisms with varying storage requirements.

Coverage results are shown in Figure 4(b). As expected for all programs an HJ results in increased coverage compared to its IJ and EJ constituents. The (IJ-10x4x7, VEJ-32x4-8) HJ performs the best resulting in 77% average coverage. However, even an (IJ-8x4x7, EJ-16x2) that requires much less storage results in a respectable 60% average coverage. In fact, for most programs, the resulting coverage is close or even exceeds (i.e., (IJ-8x4x7, EJ-16x2)) the sum of the coverage possible with the individual IJ and EJ operating in isolation. Recall, that in the hybrid organization the IJ acts as filter reducing the blocks that will be allocated in the EJ. This greatly improves EJ's utility. This behavior also suggests that IJ and EJ organizations, when operating in a hybrid configuration cover a different set of addresses for the most part. Note that, although a stand-alone VEJ has higher coverage compared to a similarly sized EJ, the differences are minor when used in hybrid configuration.

The results of this section suggest that we can get reasonably 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, most of the time significantly. The best mechanism in terms of coverage is an (IJ-10x4x7, VEJ-32x4-8) resulting in about 77% average coverage. However. much smaller configura-

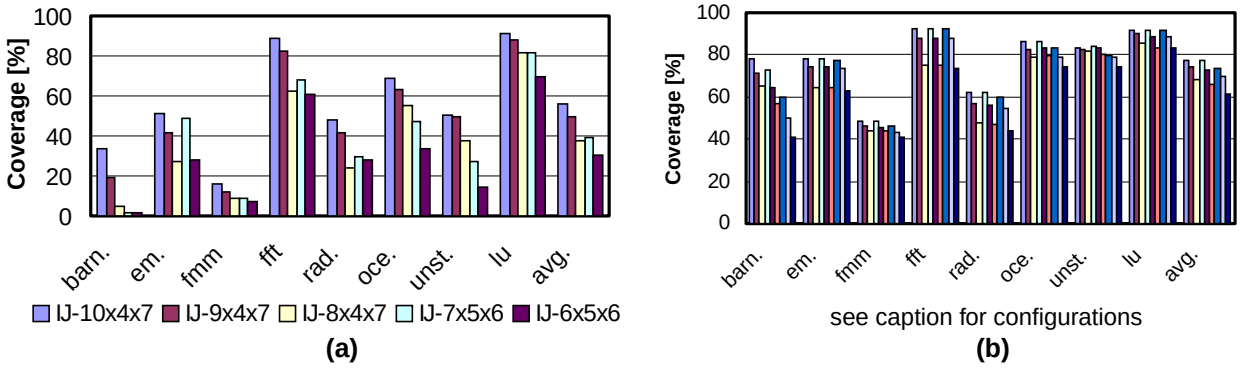


Figure 4: *(a) Coverage results for IJ configurations, named as IJ-IndexBitsxNoOfSubarraysxSkipBits (see text). (b) Coverage for the following (left to right) HJ configurations: (IJ-10x4x7, VEJ-32x4-8), (IJ-9x4x7, VEJ-32x4-8), (IJ-8x4x7, VEJ-32x4-8), (IJ-10x4x7, EJ-32x4), (IJ-9x4x7, EJ-32x4), (IJ-8x4x7, EJ-32x4), (IJ-10x4x7, EJ-16x2), (IJ-10x4x7, EJ-16x2) and (IJ-8x4x7, EJ-16x2).*

tions result in competitive coverage. For example, an (IJ-9x4x7, EJ-32x4) yields about 73% average coverage.

5.4 Power Measurements

While coverage provides an indication of the potential power savings, in itself is not a direct metric of power. We have to also factor in the power consumed by the corresponding JETTY mechanism and the power consumed by the L2 tag array on snoop accesses. In this Section, we report results on the power benefits obtained by the following JETTY organizations: IJ, (IJ, EJ-16x2), (IJ, EJ-32x4), (IJ, VEJ-32x4-8), where IJ is either IJ-10x4x7 or IJ-9x4x7. These organizations exhibit varying degrees of coverage and space requirements.

The power reduction results are shown in Figure 5, parts (a) and (b). In these experiments we took into account all snoop accesses and all JETTY updates necessary on L2 block allocation and de-allocation. We report power reduction as a fraction of the power consumed by all snoop-induced L2 tag accesses. In part (a) we report power reduction over an L2 for an Intel IA-32-like physical address space (36-bits) [2] while in part (b) we do the same over an L2 for an Alpha-like physical address space (43-bits) [1]. With the exception of *Barnes*, the (IJ-9x4x7, EJ-16x2) HJ comes very close to what is possible with the best method in terms of coverage (IJ-10x4x7, EJ-32x4-8), In general, using larger EJ or VEJ does not yield significant power improvements. While this improves coverage, it comes at the expense of higher power consumption by the JETTY. On the average, (IJ-10x4x7, EJ-32x4x-8) results in 41% power reduction, while the much smaller (IJ-9x4x7, EJ-16x2) offers a power reduction of about 38%. As shown in part (b), assuming a larger physical address space (i.e., larger L2 tags) results in slightly improved power benefits in most cases. However, the differences are minor. We have also experimented with stand-alone EJ configurations and found that in some cases they result in increased power consumption (about 6% in the worst case and for the *Lu* application) for applications where coverage is very small.

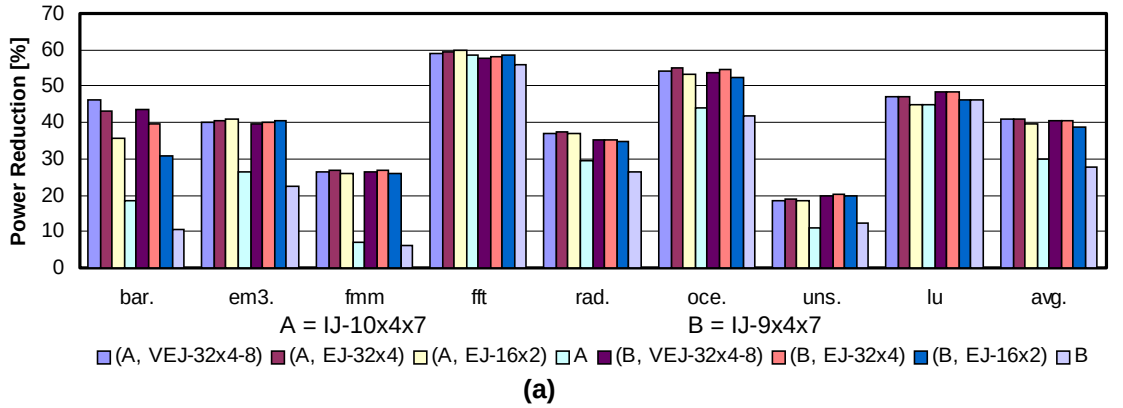
The results of this section suggest that relatively inexpensive JETTY organizations result in significant power savings, up to 41% or 44% on the average for a 36-bit- and a 43-bit-wide physical address space architectures respectively.

5.5 Smaller L2 Sizes

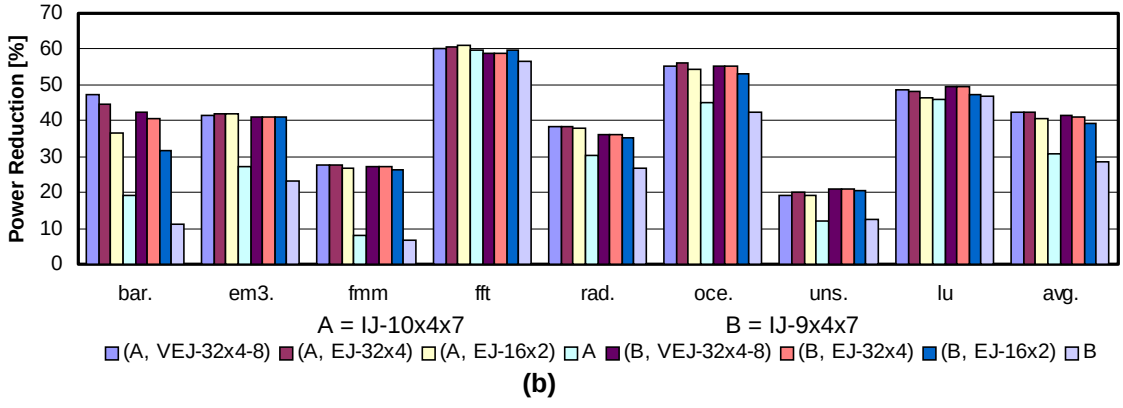
Finally, we report coverage and power reduction results for a 4-way SMP with 512K direct-mapped L2 caches. We do so to demonstrated that even with higher bus traffic, JETTY still remains a viable power reduction mechanism. Figure 6(a) reports coverage while Figure 6(b) reports power reduction. Overall, power reduction is higher compared to the 1M L2 cache configuration. There are two primary causes. First, as shown in Figure 6(a), the IJ configurations demonstrate improved coverage which results from reduced number of blocks that are cache resident. Second, the average snoop miss rate for the 512K L2 system is about 73% up from 68% for the 1M L2 system. Although reducing L2 size reduces its power consumption, the power consumption of the JETTY configurations studied remains only a tiny fraction of the power consumed by the L2. The (IJ-9x4x7, EJ-32x4) HJ achieves the highest power reduction of about 44.3% on the average.

6 Conclusion

In this work, we were motivated by the increasing importance of power consumption in computer system



(a)



(b)

Figure 5: *Power Reduction with various* JETTY *organizations. (a) Over an L2 tag-array for a 36-bit physical address space. (b) Over an L2 tag-array for a 43-bit physical address space.*

design. We argued that power consumption is also important for server applications. Accordingly, we have proposed methods for reducing the power required to perform snoops on small-scale, snoop-coherence, bus-based SMP systems. 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 blocks 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 result in a miss which we found to be 68% on the average for a 4-way SMP and a set of commonly used shared-memory benchmarks.

We have 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 using a set of shared-memory parallel applications and have found that a relative inexpensive organization filters about 77% of all snoops that would miss on the average. The corresponding power savings were 41% measured as a fraction of all snoop-induced L2 tag array lookups (for a 36-bit wide physical address space). A hybrid JETTY comprising an IJ with four 512-entry sub-arrays and a 16-set, 2-way set associative EJ resulted in power savings of 38%. We have also evaluated our approach for a system with smaller L2 caches where snoop activity was higher. We found that JETTY remains a viable power reduction technique.

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.

Acknowledgments

We wish to thank Babak Falsafi for his help with setting up and using WWT2.

7 References

[1] *Alpha 21264 Microprocessor Hardware Reference Manual, Rev. 4.2.* COMPAQ Computer Corp., Order No. EC-RJRZA-TE.

[2] *Intel Architecture Software Developer's Manual, Vol. 3.* INTEL Corp., Order no 243192, available through developer.intel.com.

[3] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, Dec. 1996.

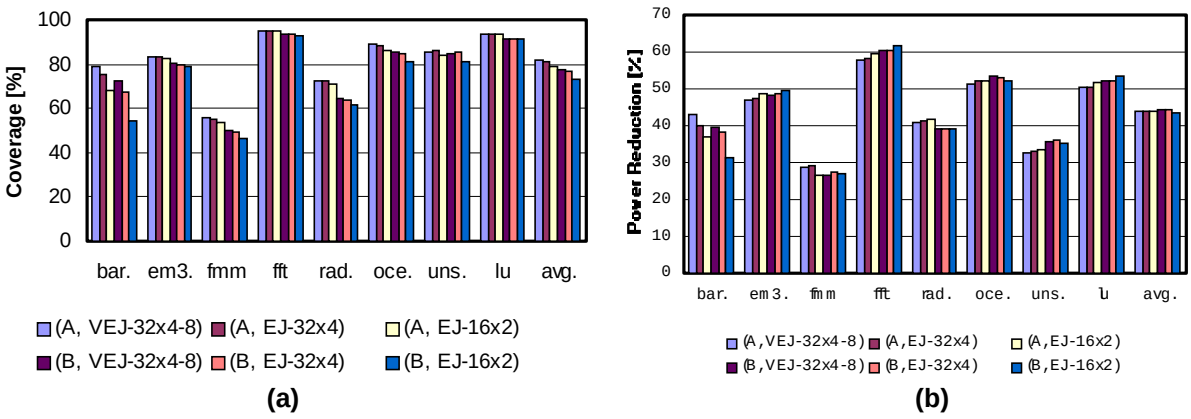


Figure 6: *Results with a 512K L2 and a 36-bit-wide physical address space. (a) Coverage. (b) Power Reduction. A is IJ-10x4x7 and B is IJ-9x4x7.*

[4] G. Albera and I. Bahar. Power and peformance tradeoffs using various cache organizations. *Power Driven Microarchitecture Workshop*, 1998.

[5] D. H. Albonesi. Dynamic IPC/clock rate optimization. In *Proc. 25th Annual International Symposium on Computer Architecture*, pages 282–292, June-July 1998.

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

[7] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A scalable architecture based on single-chip multiprocessing. *In. Proc. Annual International Conference on Computer Architecture*, 2000.

[8] 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. International Symposium on Low Power Electronics and Design*, 1998.

[9] J. Cong, L. He, Y. Khoo, C. K. Koh, and Z. Pan. Interconnect design for deep submicron ics. *Proc. IEEE International Conference on Computer-Aided Design, Invited Tutorial*, Nov. 1997.

[10] M. Corp. MIPS 4 Instruction Set Definition.

[11] D. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1999.

[12] D. E. Culler and A. D. at al. Programming in Split-C. *Supercomputing '93*, Nov. 1993.

[13] K. Diefendorff. Xeon Replaces Pentium Pro. *Microprocessor Report, vol. 12, no. 9*, June 1997.

[14] K. Ghose and M. Kamble. Energy efficient cache organizations for superscalar processors. *Power Driven Microarchitecture Workshop*, 1998.

[15] C. Gniady, B. Falsafi, and T. N. Vijaykumar. Is SC + ILP = RC? *Proc. of the Annual International Conference on Computer Architecture*, June 1999.

[16] R. Gonzalez and M. Horowitz. Supply and threshold voltage scaling for low power CMOS. *IEEE Journal of Solid-State Circuits*, Aug. 1997.

[17] M. K. Gowan, L. L. Biro, and D. B. Jackson. Power Considerations in the Design of the Alpha 21264 Processor. *In Proc. 35th Conference on Design Automation*, June 1998.

[18] L. Hammond, B. Hubbert, M. Siu, M. Prahbu, M. Chen, and K. Olukotun. The Stanford Hydra CMP. *IEEE MICRO Magazine*, Mar. 2000.

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

[20] J. Kahle. The IBM Power4 Processor. *Microprocessor forum presentation reported in: K. Diefendorff "Power4 Focuses on Memory Bandwidth", Microprocessor Report*, Oct. 1999.

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

[22] J. Kin, M. Gupta, and W. H. Mangione-Smith. The Filter Cache: An Energy Efficient Memory Structure. In *Proc. Annual International Symposium on Microarchitecture-30*, Dec. 1997.

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

[24] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The stanford FLASH multiprocessor. In *Proc. 21st Annual International Symposium on Computer Architecture*, pages 302–313, May 1994.

[25] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: speculation control for energy reduction. In *Proc. 25th Annual International Symposium on Computer Architecture*, pages 132–141, June-July 1998.

[26] 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, journal = Workshop on Performance Analysis and its Impact on Design, month=jun, year = 1997.

[27] S. S. Mukherjee, S. D. Sharma, M. D. Hill, J. R. Larus, A. Rogers, and J. Saltz. Efficient Support for Irregular Applications on Distributed-Memory Machines. *Proc. Symposium on Principles and Practices of Parallel Programming*, July 95.

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

[29] Semiconductor Industry Association. International technology roadmap for semiconductors. 1999.

[30] A. Seznec. Decoupled sectored caches: Conciliating low tag implementation cost and low miss ratio. In *Proc. 21st Annual International Symposium on Computer Architecture*, pages 384–393, Apr. 1994.

[31] A. Seznec. Don't use the page number, but a pointer to it. In *Proc. 23rd Annual International Symposium on Computer Architecture*, pages 104–113, May 1996.

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

[33] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Intel Corp., Reducing Power in High-Performance Microprocessors. *In Proc. 35th Conference on Design Automation*, June 1998.

[34] H. Wang, T. Sun, and Q. Yang. CAT – caching address tags: A

technique for reducing area cost of on-chip caches. In *Proc. 22nd Annual International Symposium on Computer Architecture*, pages 381–391, June 1995.

[35] 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.

[36] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. 22nd Annual International Symposium on Computer Architecture*, pages 24–37, June 1995.

[37] D. A. Wood and M. D. Hill. Cost-effective parallel computing. *IEEE Computer, 28(2)*, Feb. 1995.