Accurate and Complexity-Effective Spatial Pattern Prediction

Chi F. Chen, Se-Hyun Yang, Babak Falsafi
Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
{cfchen, sehyun, babak}@cmu.edu

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

http://www.ece.cmu.edu/~powertap

Abstract

Recent research suggests that there are large variations in a cache's spatial usage, both within and across programs. Unfortunately, conventional caches typically employ fixed cache line sizes to balance the exploitation of spatial and temporal locality, and to avoid prohibitive cache fill bandwidth demands. The resulting inability of conventional caches to exploit spatial variations leads to sub-optimal performance and unnecessary cache power dissipation.

This paper describes the Spatial Pattern Predictor (SPP), a cost-effective hardware mechanism that accurately predicts reference patterns within a spatial group (i.e., a contiguous region of data in memory) at runtime. The key observation enabling an accurate, yet low-cost, SPP design is that spatial patterns correlate well with instruction addresses and data reference offsets within a cache line. We require only a small amount of predictor memory to store the predicted patterns. Simulation results for a 64-Kbyte 2-way set-associative L1 data cache with 64-byte lines show that: (1) a 256-entry tag-less direct-mapped SPP can achieve, on average, a prediction coverage of 95%, over-predicting the patterns by only 8%, (2) assuming a 70nm process technology, the SPP helps reduce leakage energy in the base cache by 41% on average, incurring less than 1% performance degradation, and (3) prefetching spatial groups of up to 512 bytes using the SPP improves execution time by 33% on average and up to a factor of two.

1. Introduction

Conventional cache memories divide storage into cache lines for design simplicity and to optimize for lookup speed. The line size is fixed at design time [20] and is selected to take advantage of spatial locality with large line sizes, while limiting the cache line fetch bandwidth and latency requirements of overly large lines.

Recent research [13, 17, 18, 23, 24] indicates that there are large spatial variations in cache line usage both within and across programs. In the presence of such drastic variations, a fixed cache line size results in a sub-optimal design point. Temporally-close cache references within large contiguous spatial groups favor a single fetch of large lines to eliminate subsequent cache misses and increase performance. On the other hand, temporally-close cache references that are not spatially-close favor using a large number of small cache lines to optimize for bandwidth and placement. Using excessively large cache lines leads to unused sub-blocks that unnecessarily dissipate power both upon fetch (dissipating bitline and bus switching energy) and during cache residency (dissipating bitline and supply-to-ground leakage energy [5, 9, 11, 26]).

In this light, we present the Spatial Pattern Predictor (SPP), a table-based mechanism that accurately and cost-effectively predicts data cache line usage patterns. Upon a cache miss, the SPP predicts exactly which data words are referenced within a spatial group that consists of a contiguous region in memory. We design two cache optimizations using the SPP: (1) selective sub-blocking, in which only the predicted as to-be-referenced sub-blocks are fetched and predicted as unreferenced sub-blocks are disabled to reduce leakage energy, and (2) spatial group prefetching, in which a predicted number of cache lines are simultaneously fetched and placed in the cache to hide memory latency.

We use cycle-accurate simulation of an aggressive out-of-order superscalar processor with twelve SPEC CPU2000 benchmarks, and circuit modeling to show the following contributions. We evaluate the SPP with a 64-Kbyte 2-way set-associative L1 data cache with 64-byte lines.

- **Spatial Pattern Predictor**: We propose an accurate and cost-effective SPP design. A 256-entry direct-mapped SPP achieves a prediction coverage of 95% on average, overestimating referenced data by only 8% for our base cache. Unlike prior proposals, we determine that a combination of program counter (PC) and data reference offset (a few bits) within the cache line correlates accurately to spatial patterns, allowing for a small and accurate SPP.
- **Leakage Energy Reduction**: Using supply-gating [3] and prediction of unreferenced sub-blocks within a cache line, SPP reduces 41% of the leakage energy dissipation in our base cache with 70nm CMOS technology and incurs less than 1% performance degradation in all but one benchmark (less than 2% degradation in

all benchmarks). We measure the energy dissipation of accessing a small SPP (i.e., 1.25-Kbyte) upon a cache miss to be negligible as compared to the leakage savings.

- **Processor Performance Improvement**: Prefetching spatial patterns in groups of up to 512 bytes in our base cache using SPP improves performance in applications measured by up to a factor of two and on average by 33% over the best case execution time results achieved by a statically chosen cache line size.

The rest of the paper proceeds as follows. Section 2 presents an overview of the prior work. Section 3 makes the case for the large spatial variations in cache line usage with empirical data. Section 4 presents the mechanism of our spatial pattern predictor. In Section 5, we describe two system optimizations using the predictor. In Section 6, we quantitatively evaluate the predictor's accuracy, its coverage, storage requirement, and effectiveness in reducing leakage power and execution time. Section 7 concludes this paper.

2. Prior Work

A number of techniques for exploiting spatial locality to improve processor performance have been proposed in recent literature. Vleet et al. [24] proposed using off-line profiling to determine the fetch size upon a cache miss. However, the lack of dynamism render these static approaches less effective when faced with a data set that changes rapidly during program execution.

Many have resorted to hardware mechanisms for exploiting spatial locality at runtime. Temam and Jegou [22] suggested fetching cache lines adjacent to the requested line into a buffer to avoid cache pollution incurred by larger a cache line size while exploiting spatial locality. Gonzalez, Aliagas, and Valero [12] proposed the dual-cache that dynamically divides data across two caches with two distinct line sizes to optimize simultaneously for temporal and spatial locality. Johnson, Merten, and Hwu [13] suggested using a Spatial Locality Detection table to alternate line fetch sizes between a conventional size and a macroblock size for data that processes spatial locality. Dubnicki and LeBlanc [10] proposed an algorithm for monitoring cache line footprint and gradually increasing/decreasing fetch size by a factor of two to reduce false sharing in a cache-coherent shared memory multiprocessor. Veidenbaum et al. [23] proposed using a similar algorithm within a conventional uniprocessor cache. However, these techniques either exploit spatial locality at coarser granularity than our approach or provide limited adaptiveness and result in sub-optimal performance.

Kumar and Wilkerson suggested using the Spatial Footprint Predictor [17] to improve miss ratio in a decoupled sectored cache. Lin et al. [18] proposed using density vectors to filter out superfluous traffic when prefetching from Rambus memory banks. However, their predictors use prediction indices that are a function of data addresses and therefore require either (1) a large amount of predictor memory that is impractical to implement, (2) coarser prediction granularity to reduce storage requirement at the cost of opportunity to optimize bandwidth usage, or (3) lowered prediction accuracy and coverage. Our mechanism contrasts with their approaches by using a prediction index that is based on data reference offsets. Our study shows that employing data-reference-offset based prediction index results in accurate and cost-effective designs.

Table 1. Base system configuration parameters.

Processor Core	128-entry issue queue; 128-entry reorder buffer; 64-entry load/store queue; 8-wide fetch/dispatch/issue
Branch Predictor	Combination predictor with a 2K bimodal and a 2-level predictor table; 2-level predictor with a 2-entry level-1 (10-bit history), 1024-entry level-2, and 1-bit XOR; 1K BTB
Memory System	64-Kbyte 2-way level-1 data cache; 2-cycle hit latency; 64-Kbyte 2-way level-1 instruction cache, 2-cycle hit latency; 2-Mbyte 8-way unified level-2 cache, 12-cycle hit latency; unlimited MSHRs; 4 memory ports
Functional Units	8 integer ALUs; 2 integer multiplier/divider units; 2 floating-point ALUs; 2 floating-point multiplier/divider units

In addition, we study the effectiveness of exploiting spatial variation in line usage to reduce cache leakage power consumption. Prior studies in reducing cache leakage have primarily focused on identifying inactive cache regions at the cache line granularity either through resizing [26] or decay [11, 15]. Azizi, Moshovos, and Najm [4] also suggested using asymmetric SRAM to take advantage of value properties. Our technique contrasts with previous approaches by using spatial pattern prediction in conjunction with a supply gating mechanism [3] to reduce leakage energy dissipation at a sub-block granularity.

3. Variations in Spatial Locality

To measure the variation in cache line usage, we simulate an aggressive out-of-order processor core running a subset of the SPEC CPU2000 benchmark suite [8] using the SimpleScalar tool set [6] with Alpha binaries and reference inputs. Table 1 lists the configuration parameters for the simulated processor. In the interest of simulation turnaround, we focus the evaluation on twelve benchmarks that cover a spectrum of cache line usage behaviors and are representative of the rest of the suite. To mitigate the inaccuracies in measurement often introduced by using abbreviated instruction execution streams, we simulate each of the benchmarks to completion. We measure cache line usage as the fraction of data referenced in a cache line

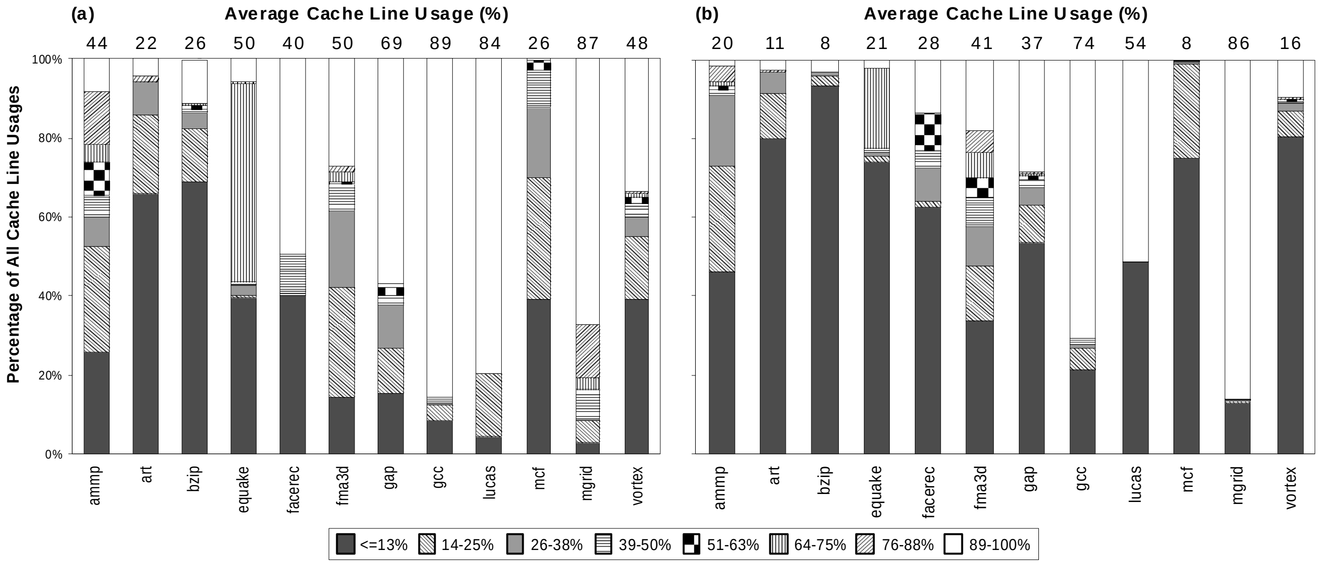

Figure 1. Variations in cache line usage for SPEC CPU2000.

starting from when the line is fetched until it is evicted from the L1 data cache. While spatial variations also exist in instruction caches, the design and evaluation of hardware mechanisms to capture these variations are beyond the scope of this work.

Figure 1 shows the distribution of cache line usage in the data cache with 64-byte and 256-byte cache lines, respectively. A minimum data word reference granularity of 8 bytes is assumed. The stacked bars indicate the fraction of all lines that exhibit a specific fraction of references before eviction. For instance, the legend for 14-25% in a bar indicates the fraction (on the y-axis) of all the lines where 14% ~ 25% of the lines had been accessed before eviction.

The figure shows a large variation in cache line usage within and across programs, regardless of the cache line size. In *gcc* and *lucas*, 80% of all cache lines are fully referenced when using 64-byte lines (Figure 1 (a)). These applications exhibit a compact data reference pattern where they reference almost all fields of their data structures. In contrast, in *art* and *bzip2*, where a single data field is referenced with a long stride, only a single 8-byte data word (i.e., ≤ 13%) is referenced in over 60% of all cache lines when using 64-byte lines. *Ammp*, *fma3d*, *gap*, and *mcf* do not exhibit a dominant line usage pattern and do not benefit from a particular fixed line size. For these applications, different program phases reference different numbers of fields in their data structures.

When using 256-byte lines (Figure 1 (b)), the overall fraction of fully-referenced lines drops as compared to that of 64-byte lines, but large variations in line usage persist. The majority of cache lines in *gcc* are still fully-referenced with 256-byte lines. *Lucas* exhibits a bimodal distribution in which half of the lines only contain up to 32 bytes of referenced data, and the other half are fully referenced. Such usage variation is mainly due to the misalignment of its data structure instances with 256-byte cache lines. In *art*, *bzip2*, and *vortex*, less than 32-byte data items are referenced for most of the cache lines.

The numbers on top of each bar in Figure 1 indicate the average cache line usage. They show that a large fraction of data fetched into L1 caches is not referenced (an average of 45% for 64-byte cache lines and 66% for 256-byte cache lines). Therefore, a significant amount of L1 cache capacity and fetch bandwidth is wasted on not referenced data. Note that the average cache line usage decreases as the cache line size increases, although the actual amount of useful data fetched on cache misses increases. Intuitively, the locality between non-adjacent words is looser than that of adjacent words because applications tend to reference data words whose addresses that are near one another close together in time. Thus, larger cache lines are less efficient in terms of cache capacity and fetch bandwidth utilization.

Instead of using a fixed cache line size, if caches dynamically adapt the amount of data fetched upon a miss based on the spatial locality, the cache space and fetch bandwidth can be exploited by other useful data to improve processor performance, or unused portions of the cache lines can be turned off to reduce energy dissipation.

4. Predicting Spatial Patterns at Runtime

To exploit variations in spatial locality, hardware (or software) must provide an accurate mechanism to predict referenced data items among the data items in adjacent cache locations. Inaccurate prediction results in wasted processor resources, and a potentially large performance and energy efficiency degradation by incurring extra accesses to lower level caches. These adverse effects may offset gains achieved by accurate predictions.

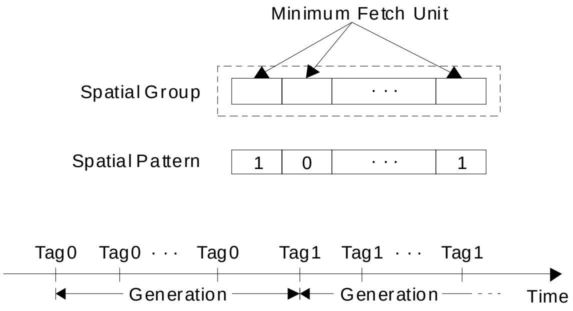

Figure 2. Spatial group, minimum fetch unit, spatial pattern, and spatial group generation.

The prediction mechanism proposed in this work is a simple history-based hardware design; it predicts future spatial locality of the executing program based on the program's behavior in the recent past. In the sections that follow, we first define the basic framework of our spatial locality prediction algorithm and then describe the details of the predictor and discuss its design space.

4.1 Spatial Patterns and Groups

In our hardware mechanism, spatial locality is recorded and predicted at the granularity of the *minimum fetch unit* of the cache. A minimum fetch unit is the smallest replacement unit of the cache and represents the amount of data including the requested word that will be fetched into cache on a cache miss. In a conventional cache the minimum fetch unit is a cache line, whereas in a sub-blocking cache the minimum fetch unit is a sub-block.

To facilitate the recording and prediction of spatial locality, adjacent minimum fetch units are grouped into *spatial groups*. The size of the spatial group is a predictor design parameter and can be independent of the base cache configuration. Each spatial group is assigned a logical tag that is used as if the group were one large cache line. The logical tag of a spatial group is obtained by masking out the least significant bits that are required to index the fetch units within the group from the address tag.

A bit vector is used to represent the recorded or predicted spatial locality of a spatial group as in [17]. We call this vector a *spatial pattern*. Each of the bits in a spatial pattern corresponds to a fetch unit and indicates if the unit is referenced. A spatial pattern is recorded over a *spatial group generation*. Similar to the cache line generation defined by Wood, Hill, and Kessler [25], a spatial group generation is the interval when the group is accessed with a unique logical tag. A spatial group generation starts when a fetch unit within the group is accessed with another logical tag. Figure 2 illustrates this definition.

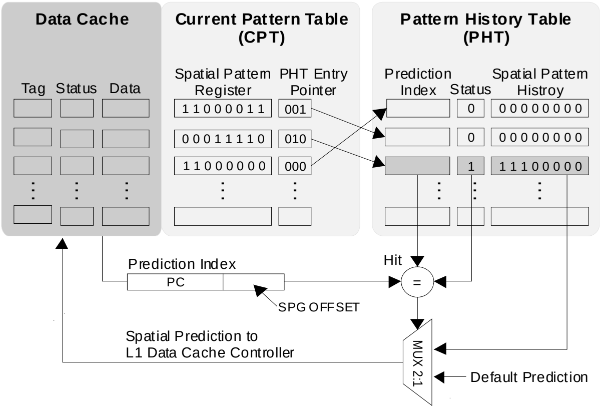

Figure 3. Spatial Pattern Predictor organization.

4.2 The Spatial Pattern Predictor (SPP)

To dynamically predict spatial patterns of spatial groups, the spatial pattern predictor (SPP) consists of a feedback mechanism and a memory to facilitate learning from feedback. The predictor is depicted in Figure 3. The main component of the feedback mechanism is the current pattern table (CPT), which records spatial patterns of the executing program. The CPT's entries are spatial pattern registers that record patterns. During a spatial group generation, the table's spatial group is read and the spatial pattern is updated on every access to that group. To avoid the high cost of implementing a separate tag array for the CPT, the table is combined with the tag array for the cache. The combination also eliminates conflicts among spatial groups because each spatial group in the cache has a dedicated entry for recording its spatial patterns.

Inside the predictor memory is the pattern history table (PHT). The PHT maintains previously-captured spatial pattern histories. At the end of each spatial group generation, the recorded spatial pattern of the spatial group is transferred from the CPT to the PHT. The PHT is read at the beginning of a new generation to make a prediction on the spatial pattern. The pattern histories stored in the PHT are accessed by using the *prediction index*. The prediction index can be a function of the program counter (PC) of memory instructions, a data address that starts a new generation of a spatial group, traces of PCs or data addresses that access a spatial group or a combination of these.

4.2.1 A Working Example. Figure 4 illustrates how a spatial pattern predictor learns and predicts in the presence of a sequence of memory instructions. When a spatial group is accessed with a new logical tag, the predictor probes the PHT that is initially empty. A new PHT entry will be allocated for storing the recorded spatial pattern. The spatial pattern register in the CPT entry for that spatial group is initialized to zero (not shown), and then the bit that corresponds to the minimum fetch unit containing the

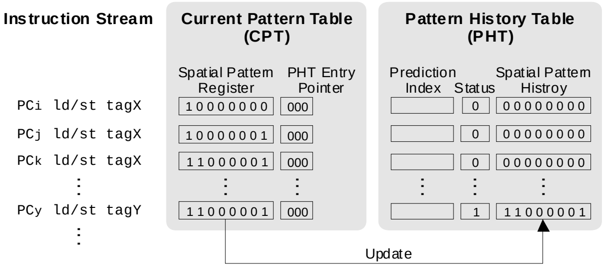

Figure 4. Spatial pattern learning and prediction mechanism.

requested word is set. A pointer to the newly allocated PHT entry will be stored in the CPT, so the recorded spatial pattern can be directly copied into the PHT entry without having to probe the PHT. As each subsequent load or store instruction arrives, the spatial pattern of the spatial group will be captured by setting the bits corresponding to the referenced fetch units. When the spatial group generation ends by referencing another logical tag, the pattern recorded will be transferred to the PHT entry pointed to by the pointer. The next time another spatial group generation starts, the same process will repeat.

If an access to the PHT misses, the predictor returns a default spatial pattern which minimizes either the performance or energy impact of mispredictions, depending on the predictor application. When a spatial pattern missing in the history table is recorded during a new generation of the spatial group, we call it the training phase as opposed to the prediction phase. Unlike many other prediction techniques, an SPP causes two types of mispredictions. If the predicted spatial pattern excludes a minimum fetch unit that is referenced during the generation of the spatial group, this misprediction is referred to as an underprediction. On the other hand, if a predicted minimum fetch unit is not referenced during the generation of the spatial group, it is called an overprediction. Depending on the application of the predictor, its training and misprediction rates affect the performance and energy efficiency in different ways. The implications of misprediction will be discussed in Section 5 with the details of example applications.

4.3 Predictor Design Space

The design space for the SPP includes the prediction index (the index used to access entries in the PHT), spatial group size, and the size and organization of the predictor memory. The selection of these parameters affects prediction accuracy, coverage, and the predictor memory's requirements. The implications of the parameters discussed in this section will be investigated in Section 6, Evaluation.

In this design space, the most critical design choice is the prediction index. Memory instructions typically reference a single field of a program's data structure. Therefore, as long as the data structure is aligned with the spatial group, the program counter (PC) of the instruction may be a good indicator for the spatial pattern. The key observation behind using a PC-based index for prediction is that instructions provide a concise representation of history [14]. Moreover, it has been observed that most cache misses are caused by a very small number of instructions [2], and thus repetitive code fragments may help capture and predict the spatial patterns of a program.

Unfortunately, data structure instances are not necessarily aligned with the spatial group's boundaries, so data address information may be required. Using a PC concatenated with the data address as the prediction index can effectively track different alignments of data structures. However, the PC-data-address combination can impose a high demand on the capacity of the PHT; tracking spatial patterns of active memory instructions at each unique data location requires a large number of entries to maintain satisfactory prediction coverage in the presence of a large working set.

To address the high predictor memory capacity requirement imposed by the PC-and-data-address based prediction index and alignments, one can use the index based on the PC concatenated with the offset within a spatial group of requested data. The key observation behind using the PC along with the spatial group offset as the prediction index is that a given sequence of memory instructions accesses the same fields of a data structure. Although different instances of a data structure may have different offsets within a spatial group, the number of different offsets of an instance with respect to a fix-sized spatial group is bounded. Thus, the spatial pattern of a sequence of load and store instructions following a faulting memory instruction can be captured by simply tracking the PC of the faulting memory instruction and the offset of the requested word within the spatial group.

This approach requires more predictor memory than a PC-only based prediction index because each unique offset will require a separate entry in the PHT. Nevertheless, the prediction index based on PC and spatial group offset will have higher coverage than that of the PC-data-address combination with approximately the same prediction accuracy because multiple data locations may have the same alignment within a spatial group. Thus, the effort of learning of a particular PC-offset combination can be amortized over all data with the same offset that is accessed by the same memory instruction. In this work, instead of using the full data address, we employ the offset in the spatial group to indicate the alignment of the data field. The spatial group offset is concatenated with the PC of the faulting memory instruction. To keep the resulting prediction index to a reasonable size, a number of most significant bits are masked out from the PC. The PC-spatial-group-offset combination results in efficient predictor designs (see Section 6.1 for evaluation).

Another important design parameter is the spatial group size. Enlarging the spatial group increases the number of minimum fetch units that can be fetched together. Data items that are placed far from each other in a larger spatial group are less likely to exhibit dense spatial locality. As such, with a fixed minimum fetch unit size, larger spatial groups require longer spatial patterns which are harder to predict accurately. Similarly, spatial patterns for larger spatial groups might not be as predictable as those for smaller spatial groups, even if a larger minimum fetch unit is employed and the size of spatial pattern remains constant.

As in any table-based predictor, a key design parameter affecting the SPP's accuracy and coverage is the size and organization of the predictor memory. The required predictor memory size and organization are determined by the prediction index and spatial group size. The number of stored spatial pattern histories directly affects accuracy and coverage. To avoid aliasing, the PHT can be built with tags extracted from the prediction index. Using tags may decrease the predictor's coverage but increase accuracy. In this case, to reduce the history table access misses, the predictor can be organized with a high associativity. Because the history table is not frequently accessed (only at the start and the end of a spatial group generation) and therefore is tolerant of high latency, a high associativity does not adversely affect the predictor's performance.

5. Predictor Applications

The spatial pattern predictor can be applied in various ways to compensate for the shortcomings of a fixed cache line size. This section applies the SPP to (1) reduce energy dissipation by disabling the sub blocks that are fetched but not referenced before eviction, and (2) improve processor performance by prefetching neighboring data that would otherwise cause cache misses.

5.1 Energy-Aware Selective Sub-Blocking

Leakage energy dissipation in high-performance cache memories has emerged as a critical issue in a wide spectrum of microprocessor designs [11, 15, 26]. Historically, the primary source of energy dissipation in CMOS transistor devices has been the switching energy from charging/discharging load capacitances when a device switches states. However, scaling down transistor supply and threshold voltages to reduce switching energy consumption and maintain performance exponentially increases sub-threshold leakage current [5, 9]. Moreover, modern state-of-the-art microprocessor designs devote a large fraction of the chip area to memory structures. For instance, more than 50% of the die area of the Intel Itanium2 processor [16] and 60% of the StrongARM processor is designated to cache and other memory structures [19]. Because leakage energy is a function of the number of on-chip transistors, independent pendent of their switching activity, leakage energy dissipation in caches accounts for an increasingly large component of energy dissipation and will continue to do so in the future.

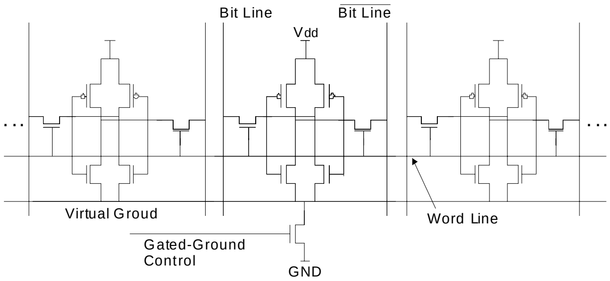

Figure 5. Gated-ground illustrated.

Prior architectural techniques for reducing leakage energy dissipation in cache memories [11, 15, 26, 27] have focused on controlling the energy dissipation at a granularity of one or more cache lines. Our technique contrasts with, yet complements previous approaches by using the SPP to exploit usage variation and underutilization within a single cache line. The SPP determines the cache line's underutilization and reduces leakage energy dissipation from the unreferenced fractions of a cache line.

To exploit usage variation within a cache line, we assume a sub-blocking cache where each of the cache lines is partitioned into a number of sub-blocks. In the context of SPP, the minimum fetch unit is a sub-block, and we elect to use a cache line as its spatial group. A single spatial pattern is recorded and predicted for each cache line at a sub-block granularity. A spatial group generation is the same as the cache line generation [25]: the interval between two consecutive misses on the same cache line. The predictor probes the PHT on every cache miss and makes a spatial pattern prediction for each incoming cache line. All sub-blocks are fetched on a cache miss. However, once a cache line is filled, only the sub-blocks that are predicted to be referenced are kept in an active mode. The rest of the sub-blocks are put into a standby mode to reduce leakage energy dissipation. We call this cache a selective sub-blocking cache. In training, the predictor conservatively enables all sub-blocks to minimize the performance impact.

5.1.1 Data Retention Gated-Ground. To reduce the leakage energy dissipation in a selective sub-blocking cache with aggressive threshold-voltage scaling, we elect to use a circuit-level technique called data retention gated-ground [3, 21]. The technique introduces an extra transistor in the leakage path from the supply voltage to the ground of the cache's SRAM cells; the extra transistor is turned on only in sub-blocks that are in active energy mode, "gating" the cell's supply voltage. Gated-ground maintains the performance advantages of lower supply and threshold voltages, while reducing leakage. Figure 5 depicts the anatomy of a data retention gated-ground scheme in which the gated-ground transistor is shared across a sub-block.

Data retention gated-ground enables a selective sub-blocking cache to virtually eliminate the leakage in disabled sub-blocks while preserving the data values. By preserving the data, mispredictions of selective sub-blocking do not incur additional lower-level cache accesses; if the words within a disabled sub-block are referenced, the request can be serviced after the sub-block is enabled. Using SPICE simulation tools, we measured the leakage energy reduction and performance impact of the data retention gated-ground with an assumption of 70nm CMOS technology and a 1V supply voltage [1]. When the gated-ground transistor is off and the leakage path between the supply voltage and ground is cut off, the leakage energy dissipation of the SRAM cell falls by a factor of 19. The transition delay of turning on a gated-ground transistor shared by a 16-byte sub-block is only 0.20ns, or one clock cycle in a 5-GHz microprocessor.

5.1.2 Impact on Energy and Processor Performance. Mispredictions and predictor training have implications on the total energy dissipation and performance. Overprediction and underprediction in the case of mispredictions have different impacts on energy and performance. Overprediction (mistakenly enabled unreferenced sub-blocks) decreases the effectiveness of the predictor and reduces energy savings. Underprediction (mistakenly disabled referenced sub-blocks) incurs an access delay, impacting processor performance. Accesses on underpredicted sub-blocks must be delayed until the gated-ground transistor of the sub-blocks turns on. Those delayed memory accesses affect the performance in two ways: (1) the data acquired from the memory instruction is ready after a cycle delay, and (2) instruction scheduling is complicated due to an increased variation in load latency (modern processors speculate on load latency to enable aggressive scheduling of dependent instructions).

In training, the predictor enables all sub-blocks in the cache line. Although this strategy reduces the energy savings, it minimizes the performance degradation from accessing disabled sub-blocks. The high coverage and accuracy of the predictor (see Section 6 for evaluation) ensures that the opportunity for energy savings is closely tracked without incurring significant performance degradation.

The predictor's own hardware structures also consume energy. The energy overhead of the CPT is minimal, as the table is essentially just a few bits added in the tag array of the cache. The number of extra bits is the size of the spatial pattern, which is between 4 and 16 bits. The added bits increase the overall switching energy of a 64-Kbyte 2-way set-associative L1 data cache by approximately 2%. The PHT's energy overhead is also minimal. Our study shows (see Section 6.3) that the PHT can be designed as a 256-entry tag-less RAM table with each entry containing one spatial pattern. Although such a structure has a per access power consumption that is approximately equal to 12% of that of the L1 data cache, the table is accessed infrequently, only on cache misses. Simulation results show that the PHT increases the overall L1 data cache switching energy by only 1% on average and up to 4%, in the case of *art*. The effectiveness of using SPP to reduce leakage energy will be evaluated in Section 6.4.

5.2 Spatial Group Prefetchers (SGP)

Increasing processor clock speeds and microarchitectural innovations have led to a growing disparity between processor and memory performance. Many chip designers have resorted to prefetching techniques to mitigate the shortcomings of the conventional memory hierarchy organization. Prefetching uses predictions based on past memory usage patterns to fetch data in advance of its use to hide the memory latency. In this section, we present the Spatial Group Prefetcher (SGP) as an application of the SPP for improving processor performance. An SGP predicts neighboring data items to be fetched with the data requested on a cache miss. The key observations behind an SGP are that: (1) data reference streams exhibit spatial locality, and (2) the utilized fraction of a cache line tends to decrease as the cache line size increases. Although larger cache lines may help exploit high spatial locality and improve miss rates, they also impose higher bus bandwidth requirements and greater miss penalties. As a result, a processor with larger cache lines may spend much time stalled due to increased bus bandwidth demands, offsetting the benefit of reduced miss ratios. With its high accuracy and coverage, the SGP enables building a data cache that has the high hit rate benefits of a cache with larger lines and the bandwidth efficiency of smaller cache lines [7].

To apply an SPP to data prefetching, we group adjacent cache lines into a spatial group. Each of the cache lines within a group has its own tag, and therefore a cache line is still the smallest replacement unit. To facilitate the feedback mechanism, each spatial group is assigned a logical tag that is used as if the group were one large cache line. The logical tag of a spatial group is obtained by masking out the least significant bits from the address tag that are required to index the cache lines within the group.

Unlike selective sub-blocking caches, the spatial group generation for an SGP starts with a request to a logical tag that differs from the tag whose spatial pattern is being recorded. Note that such a request does not necessarily occur during a miss because the requested data may already reside in one of the member cache lines in the group. The spatial group generation is chosen to exploit spatial locality where items whose addresses are close in space are likely to be accessed close in time. A request to a different tag can

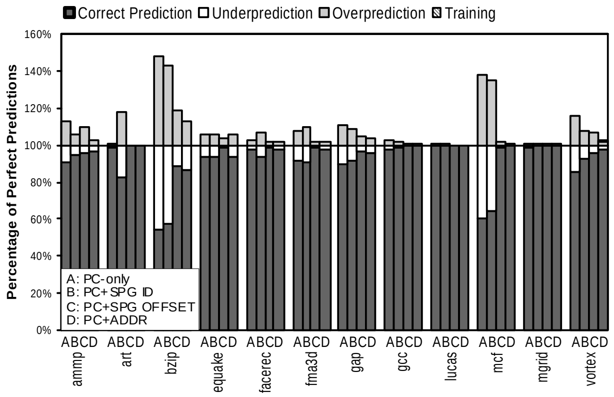

Figure 6. Prediction accuracy and coverage of various prediction indices with infinite large predictor memory.

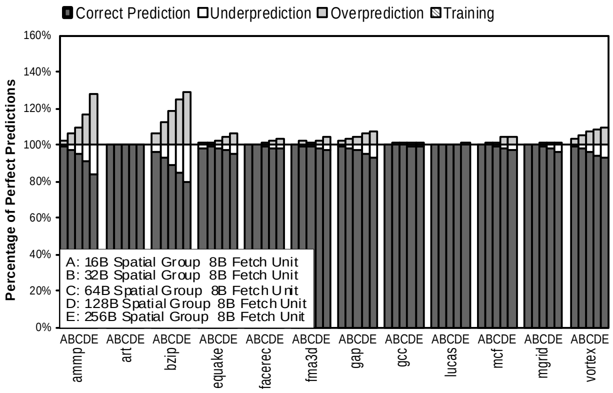

Figure 7. Prediction accuracy and coverage of various spatial group sizes with constant fetch unit size.

indicate that the reference stream in the vicinity of the current recording tag has terminated.

An SGP predicts which lines within the group should be fetched upon each cache miss. Because the SGP does not store data addresses, the addresses to prefetch are obtained from the requested address, based on the spatial pattern prediction. In training, the predictor fetches only the requested data to avoid data pollution and bus bandwidth waste commonly seen in overly-aggressive prefetching schemes [6]. The effectiveness of the SGP on improving processor performance will be evaluated in Section 6.5.

6. Evaluation

We conduct a sensitivity study over the design space of the SPP to find the best practical configuration. The effectiveness of using the SPP for reducing leakage energy of an on-chip cache memory and for data prefetching to improve processor performance is also evaluated using an optimally configured predictor. In Section 6.1 and 6.2, we investigate the impact of various prediction indices and spatial group sizes on the predictor's accuracy and coverage. In both subsections, we assume a PHT with an unlimited number of entries to filter out artifacts introduced by having an insufficient predictor memory. In Section 6.3, we consider a variety of practical PHT implementations and organizations. In Section 6.4 and 6.5, the predictor configured with attributes chosen in the sensitivity study will be used to evaluate the effectiveness of an SPP in the example applications.

In the sensitivity study, we simulate each benchmark to completion. In the interest of simulation turnaround with our timing simulator, we elect to skip the first 10 billion instructions and simulate the next 500 million instructions for each benchmark. The configuration parameters for the simulated processor are in Table 1.

6.1 Prediction Index

The prediction index selection can have a profound impact on the predictor's accuracy. We consider several prediction indices that are based on the program counter (PC) of faulting memory instructions. Figure 6 presents the prediction accuracy and coverage of a spatial group predictor for a 64-Kbyte 2-way set associative L1 data cache. The graph shows the percentage of correct spatial pattern predictions, the fraction of underpredictions, and the fraction of overpredictions. Figure 6 also shows the fraction of spatial patterns not predicted due to predictor training with respect to a perfect predictor that always knows exactly what fetch units will be referenced, given an index for prediction. Since we assume a PHT with unlimited capacity, few spatial patterns are not predicted due to training.

As indicated by the graph, using the PC (program counter of the faulting memory instruction or the memory instruction accessing a different logical tag) concatenated with the data address as the prediction index produces the best overall prediction accuracy. On the other hand, using just the PC as the prediction index yields the poorest prediction accuracy among all considered indices for prediction. Our observation is in agreement with Kumar and Wilkerson [17]. The reason why the index based on PC has low prediction accuracy is that, although the same memory instructions always access the same field of a particular data structure, different instances of the data structure may have different alignments with respect to a spatial group. Similarly, using a PC concatenated with a spatial group tag yields low prediction accuracy because spatial group tags do not contain the data structure alignments. With *bzip2* and *mcf*, in particular, indices that do not pertain to alignments within a spatial group have noticeably lower prediction accuracy than other candidates because the benchmarks perform computations on sets of arrays whose elements tend to have different spatial group alignments. The results also show that using a PC concatenated with the spatial

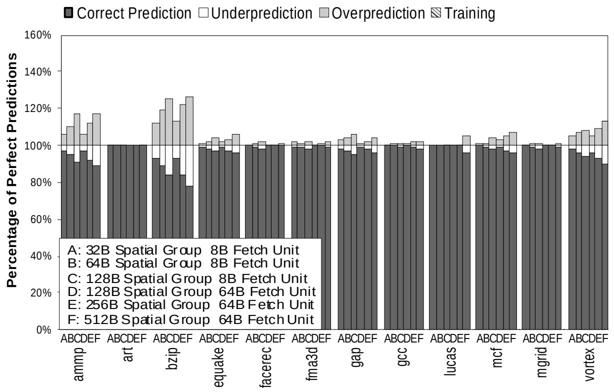

Figure 8. Prediction accuracy and coverage of various spatial-group-fetch-unit ratios.

group offset produces a prediction accuracy that is comparable with the PC-data-address based prediction index.

6.2 Spatial Group and Fetch Unit Size

Spatial group size can affect the prediction accuracy of an SGP as well. Figure 7 presents the prediction accuracy and coverage of a spatial group predictor with various spatial group and fetch unit sizes. As indicated by the graph, the predictor's accuracy degrades as the size of the spatial group increases. The key intuition behind why prediction accuracy decreases as the spatial group size increases is that memory instructions within a basic block tend to access data in close vicinity; as the size of a spatial group increases it might cover data accessed by multiple basic blocks with different memory behaviors. Since the predictor uses only the program counter and spatial group offset as the prediction index, different instruction sequences that share the same faulting memory instruction and spatial group offset are not uniquely tracked. Therefore, the spatial patterns corresponding to a particular prediction index from previous generations may not be the same. Prediction accuracy with larger spatial group sizes may also decrease if the spatial groups span across a data set accessed from multiple basic blocks.

Modern out-of-order superscalar engines can issue multiple accesses in parallel. This further increases the likelihood that spatial patterns captured for a particular index may be polluted by instruction sequences from other basic blocks whose data sets are contained in the same spatial group. The effect is especially profound with integer applications such as *bzip2*, *gap*, and *vortex*. Floating point applications tend to be less sensitive to the spatial group size, with the exception of *ammp*. To contain the effect of aliasing instruction sequences, one can combine multiple spatial pattern histories from different spatial group generations to generate a prediction for each index. This may bias the predictor towards overprediction. Alternatively, one can increase coarseness of the prediction by using a larger fetch unit size, as indicated by Figure 8.

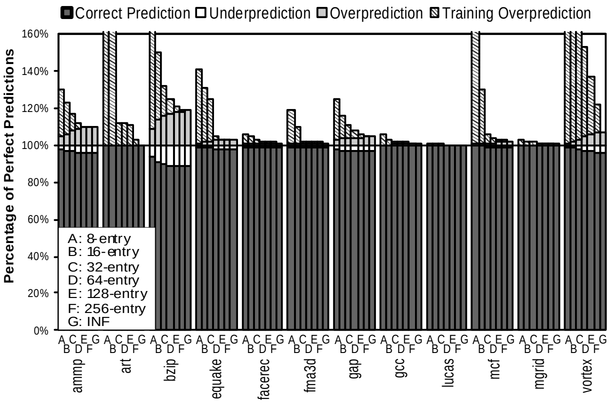

Figure 9. Prediction accuracy and coverage of various PHT sizes for selective sub-blocking.

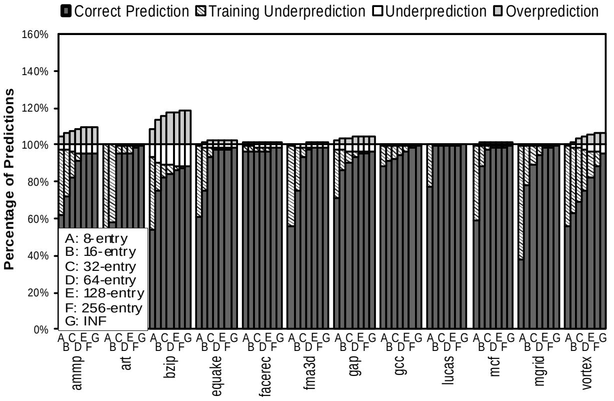

Figure 10. Prediction accuracy and coverage of various PHT sizes for the Spatial Group Prefetcher.

6.3 Predictor Memory Capacity and Organization

Figure 9 shows the prediction accuracy and coverage of the SPP for a selective sub-blocking cache with various PHT sizes. In a selective sub-blocking cache, the predictor conservatively enables all sub-blocks in a cache line when given an index and no corresponding spatial pattern is found. Thus, the predictor may cause additional overpredictions when in training. Figure 10 presents the prediction accuracy and coverage of an SPP for an SGP with various PHT sizes. In an SGP, the predictor fetches only the requested data if the requested spatial pattern is not found in the PHT. Therefore, the predictor may cause additional underpredictions when in training.

To gauge the full potential of each capacity, the tables are assumed to be fully associative. Practical implementation for the table sizes that produce the best prediction accuracy and coverage will be considered later in this section. Ideally, the number of PHT entries is the product of the unique active memory instruction sequences and the

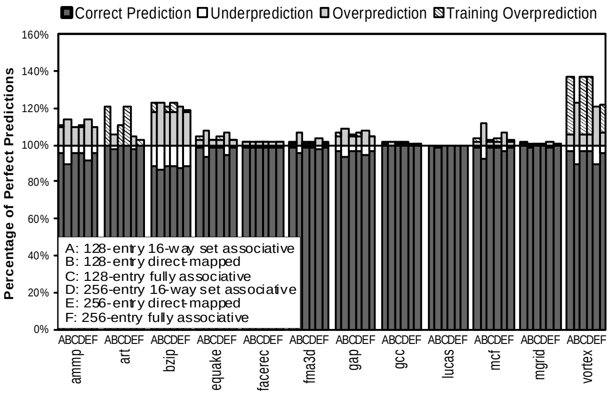

Figure 11. Prediction accuracy and coverage of various PHT organizations for selective sub-blocking.

different spatial data structure offsets referenced in a given program phase.

As indicated by the graphs, the capacity of the PHT has a profound impact on the prediction coverage of the predictor. The results show that, while a PHT with 128 and 256 entries yields prediction accuracy and coverage that is comparable to that of an infinite table. In general the prediction coverage of an SPP degrades as the capacity of the PHT decreases. *Bzip2* and *vortex*, in particular, show little tolerance for tables with insufficient capacity. Although *bzip2*, a lossless, block-sorting data compressor, has a small code base, its algorithm performs multiple transformations that have a large number of concurrent memory instruction sequences. Similarly, *vortex*, a single-user object-oriented database, performs a mix of database operations, and therefore has many memory instructions that cause cache misses to data with different alignments within spatial groups.

Figure 11 presents the prediction accuracy and coverage of 128- and 256-entry PHTs with various organizations for selective sub-blocking caches. In the interest of space, the results of the predictor for the data prefetching application are omitted because they display similar behavior. We consider a 16-way set associative and a direct-mapped tag less implementation for each capacity. The results show that while the 16-way set associative tables produce a prediction accuracy comparable to an ideal fully-associative table, they incur a severe prediction coverage degradation. The impact on prediction coverage is especially noticeable in *art* and *vortex*. On the other hand, direct-mapped tables have a lower prediction accuracy, but they yield performance closer to an ideal fully associative table.

6.4 Energy Reduction Effectiveness

Figure 12 shows the leakage energy dissipation in a 64-Kbyte, 2-way set associative L1 data cache (upper bars) and the increase in execution time of the programs (lower bars). The leakage energy dissipation and execution time are normalized to a processor with a conventional data cache. The graph does not account for the predictor energy dissipation overhead because the leakage energy dissipated by the CPT and the PHT is negligible, as explained in Section 5.1. The graph indicates that selective sub-blocking is effective in reducing leakage energy dissipation; on average the technique reduces the leakage energy dissipation by 41%, with less than a 1% performance impact. The most notable benchmarks are *art*, *bzip2*, and *mcf*, where selective sub-blocking achieves a leakage energy reduction of 73%, 64%, and 68%, respectively, thanks to the programs' sparse spatial patterns.

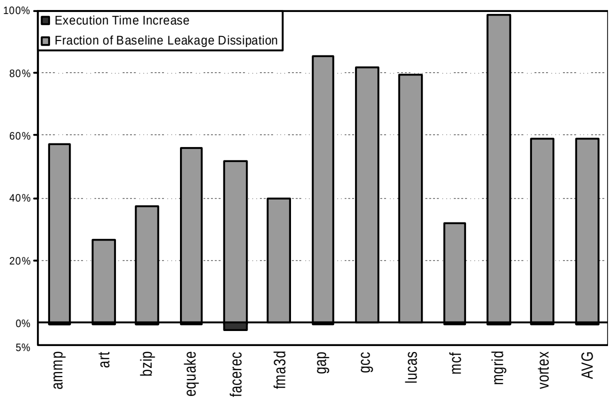

Figure 12. Normalized leakage energy dissipation in a 64-Kbyte, 2-way set associative L1 data cache and program execution time increase for selective sub-blocking.

We compare SGP against conventional data caches with cache line sizes equal to that of the predictor's spatial group. In order to measure the impact of a longer cache fill time and L1/L2 bus contention incurred by larger cache line

Table 2. Performance comparison of SGP against demand-fetched systems with various cache line sizes.

	256B	512B	1024B	SGP 256B	SGP 512B	SGP 1024B
ammp	-9	-41	-63	7	10	-25
art	4	32	96	15	121	305
bzip	-10	-43	-49	3	6	8
equake	-2	-34	-41	14	59	99
facerec	-4	-13	-3	9	58	103
fma3d	-2	-9	-9	-2	0	0
gap	7	20	31	9	31	47
gcc	0	-2	-2	0	1	1
lucas	0	-23	-67	8	34	51
mcf	-6	-27	-32	6	38	67
mgrid	10	6	12	14	36	53
vortex	-4	-27	-43	0	1	1
AVG	-1	-13	-14	7	33	59

sizes, the simulator is augmented to model a realistic bus implementation. The bus between L1 and L2 cache is 256 bits wide. On a cache miss, the critical word is serviced first. The remaining words are transferred in round-robin fashion with each word taking 2 additional cycles. The transaction completes when the last word of the cache line is filled. The bus always gives priority to processor requests over SGP prefetch requests.

Table 2 presents percent speedup of an SGP with 256-, 512-, and 1024-byte spatial groups and demand-fetched L1 data caches with various cache line sizes over a demand-fetched data cache with 64-byte cache lines. Our first observation is that although larger cache lines can help exploit high spatial locality and improve miss rates, they might cause performance degradation as a result of longer cache fill time and increased bus traffic. As indicated by the results, many benchmarks see their execution time increase as the processor spends more time stalled due to limited bus bandwidth. For benchmarks that march through arrays of data structures such as *art*, *gap*, and *mgrid*, larger cache lines indeed help eliminate misses caused by undersized cache lines and improve performance. For the rest of benchmarks, the performance degrades as the cache line size increases, due either to increased bus contention or data conflicts, or both.

Because the SGP only fetches cache lines that will probably be referenced, it can eliminate bus traffic wasted by fetching unused data. On average, the SGP achieves a 33% and 59% speedup with 512- and 1024-byte spatial group sizes, respectively. For benchmarks with high, yet sparse, spatial locality such as *art*, *equake*, and *gap*, the SGP further improves upon larger cache lines by prefetching only the data that will be referenced and reducing bus contention, and achieves a 305%, 99%, and 47% speedup respectively. For benchmarks that have high and dense spatial locality such as *gcc*, the SGP does not improve performance because SGP prefetches approximately the same amount of data as a conventional cache with a cache line size equal to its spatial group. In *ammp*, the SGP with 1024-byte spatial groups causes a performance degradation of 25% due to untimely prefetching; the prefetcher fetches data too early and causes replacements of data items that are later referenced again. Such an effect can be avoided by either using a prefetch buffer where the prefetched cache lines are stored until they are referenced or employing confidence counters. Design and evaluation of such buffers and confidence counters is beyond the scope of this work.

7. Conclusion

We described the Spatial Pattern Predictor (SPP), a cost-effective table-based hardware mechanism that accurately predicts the reference patterns within a spatial group at runtime. The key observation enabling the accurate, low-cost SPP design is that spatial patterns correlate well with instruction addresses and data reference offsets within a line, requiring a small number of table entries to store the predicted patterns. We presented two cache optimizations using an SPP: (1) selective sub-blocking, in which the predicted as to-be-referenced sub-blocks are fetched simultaneously to eliminate subsequent misses and predicted as unreferenced sub-blocks are disabled to save leakage energy, and (2) spatial group prefetching, in which predicted cache lines are simultaneously fetched and placed in the cache.

Using cycle-accurate simulation of an aggressive out-of-order processor and circuit modeling for a 64-Kbyte 2-way set-associative L1 data cache with 64-byte lines, we showed that: (1) a 256-entry tag-less direct-mapped SPP can achieve, on average, a prediction coverage of 95%, overpredicting the patterns only by 8%, (2) assuming a 70nm process technology, our SPP reduces leakage energy in the base cache by 41% on average, incurring less than 1% performance degradation, and (3) prefetching spatial groups of up to 512 bytes improves execution time by 33% on average and up to a factor of two.

8. Acknowledgments

We would like to thank Jared Smolens, Deepti Srivastava, Roland Wunderlich, and the anonymous reviewers for their insightful comments on earlier drafts of this paper. This work is supported in part by the SRC contracts 2003-HJ-1086 and 2001-HJ-901, the DARPA PAC/C contract F336150214004-AF, an IBM faculty partnership award, and donations from Intel. Andreas Moshovos is also supported by grants from the National Sciences and Engineering Research Council of Canada, and the University of Toronto.

9. References

[1] http://www-device.eecs.berkeley.edu/ptm/.

[2] S. G. Abraham, R. A. Sugumar, D. Windheiser, B. R. Rau, and R. Gupta. Predictability of Load/Store Instruction Latencies. In *Proceedings of the 26th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 26)*, Dec. 1993. http://www.acm.org/sigs/pubs/proceed/template.html.

[3] A. Agarwal, H. Li, and K. Roy. DRG-Cache: A Data Retention Gated-Ground Cache for Low Power. In *Design Automation Conference*, June 2002.

[4] Navid Azizi, Andreas Moshovos, Farid N. Najm. Low-leakage asymmetric-cell SRAM. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design* (ISLPED), pages 90–95, July 2002.

[5] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, 19(4):23–29, July 1999.

[6] D. Burger and T. M. Austin. The SimpleScalar Tool Set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin–Madison, June 1997.

[7] D. Burger, J. R. Goodman, and A. Kagi. Memory Bandwidth Limitations of Future Microprocessors. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, May 1994.

[8] S. P. E. Corporation. SPEC CPU2000. In http://www.spec.org, 2000.

[9] B. Davari, R. Dennard, and G. Shahidi. CMOS Scaling for High Performance and Low Power- the Next Ten Years. *Proceedings of the IEEE*, 83(4):595, June 1995.

[10] C. Dubnicki and T. J. LeBlanc. Adjustable Block Size Coherence Caches. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 170–179, June 1992.

[11] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002.

[12] A. Gonzalez, C. Aliagas, and M. Valero. A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality. In *International Conference on Supercomputing*, July 1995.

[13] T. Johnson, M. Merten and W.-M. Hwu. Run-Time Spatial Locality Detection and Optimization. In *Proceedings of the 31st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 31)*, 1998.

[14] S. Kaxiras and J. R. Goodman. Improving CC-NUMA Performance Using Instruction-Based Prediction. In *Proceedings of the Fifth IEEE Symposium on High-Performance Computer Architecture*, pages 161–170, Feb. 1999.

[15] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, July 2000.

[16] K. Krewell. Itanium 2 Arrives with a Benchmarking Bang. In *Microprocessor Report*, Aug. 2002.

[17] S. Kumar and C. Wilkerson. Exploiting Spatial Locality in Data Caches Using Spatial Footprints. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998.

[18] W.-F. Lin, S. K. Reinhardt, D. Burger, and T. R. Puzak. Filtering Superfluous Prefetches Using Density Vectors. In *International Conference on Computer Design*, 2001.

[19] S. Manne, A. Klauser, and D. Grunwald. Pipeline Gating: Speculation Control for Energy Reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 132–141, June 1998.

[20] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 1994.

[21] 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 *Proceedings of the 2000 International Symposium on Low Power Electronics and Design* (ISLPED), pages 90–95, July 2000.

[22] O. Temam and Y. Jegou. Using Virtual Lines to Enhance Locality Exploitation. In *International Conference on Supercomputing*, July 1994.

[23] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji. Adapting Cache Line Size to Application Behavior. In *International Conference on Supercomputing*, July 1999.

[24] P. V. Vleet, E. Anderson, L. Brown, J.-L. Bear, and A. Karlin. Pursuing the Performance Potential of Dynamic Cache Line Sizes. In *International Conference on Computer Design*, Oct. 1999.

[25] D. A. Wood, M. D. Hill, and R. E. Kessler. A Model for Estimating Trace-Sample Miss Ratios. In *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 79–89, May 1991.

[26] 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 *Proceedings of the Seventh IEEE Symposium on High-Performance Computer Architecture*, Jan. 2001.

[27] S.-H. Yang and B. Falsafi. Near-Optimal Precharging in High-Performance Nanoscale CMOS Caches. In *Proceedings of the 36th ACM/IEEE International Symposium on Microarchitecture (MICRO-36)*, December 2003.