

Accurate and Complexity-Effective Spatial Pattern Prediction

by

Chi Fu Chen

Submitted to the Department of Electrical and Computer Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Engineering

at

CARNEGIE MELLON UNIVERSITY

August 2003

© Chi Fu Chen. All rights reserved.

The author hereby grants Carnegie Mellon University permission to
reproduce and distribute publicly paper and electronic copies of this report
document in whole or in part.

Accurate and Complexity-Effective Spatial Pattern Prediction

by

Chi Fu Chen

Abstract

Recent research suggests that there is a large variation in a cache's spatial usage both within and across programs. Unfortunately, conventional caches typically employ a fixed cache line size to balance exploiting both 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 thesis describes the Spatial Pattern Predictor (SPP), a cost-effective table-based hardware predictor that accurately predicts the reference patterns within a spatial group (i.e., a contiguous region of data in memory) at runtime. The key observation enabling accurate yet low-cost SPP design is that spatial patterns correlate well with instruction addresses and reference offsets within a line, requiring a small number of table entries to store the predicted patterns. We use cycle-accurate simulation of an aggressive out-of-order processor and circuit modeling for a 64-Kbyte 2-way set-associative level-one data cache with 64-byte lines to show that: (1) a 256-entry 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 2% performance degradation, and (3) prefetching spatial groups of up to 512 bytes improves execution time on average by 33% and up to a factor of two.

Acknowledgments

I thank my advisor, Professor Babak Falsafi, for giving me the chance to take on this project and his guidance in this work. It has been a learning experience, to say the least. I also want to express my gratitude and appreciation to his taking risks to give me the freedom of pacing myself.

I thank Professor Juan Restrepo of Mathematics Department, The University of Arizona for encouraging me to go on to graduate school. I also thank him for opening the window to the world of research for me by letting me work on the TaskFarmer project.

I dedicate this thesis to my parents, who have made many sacrifices for my education. Without their guidance, moral, and financial support, I would not have made it this far. I am forever indebted to them for who I am.

To my parents

Contents

1 INTRODUCTION.....	13
2 PRIOR WORK.....	17
3 VARIATIONS IN SPATIAL LOCALITY	19
4 PREDICTING SPATIAL PATTERNS AT RUNTIME.....	25
4.1 SPATIAL PATTERNS AND GROUPS	25
4.2 A SPATIAL PATTERN PREDICTOR (SPP).....	27
5 PREDICTOR APPLICATIONS	33
5.1 ENERGY-AWARE SELECTIVE SUB-BLOCKING.....	33
5.2 SPATIAL GROUP PREFETCHERS (SGP)	37
6 EVALUATION	39
6.1 INDEX FOR PREDICTION.....	40
6.2 SPATIAL GROUP AND FETCH UNIT SIZE	42
6.3 PREDICTOR MEMORY CAPACITY AND ORGANIZATION.....	44
6.4 ENERGY REDUCTION EFFECTIVENESS.....	47
6.5 DATA PREFETCHING PERFORMANCE.....	48
7 CONCLUSION	51

List of Figures

FIGURE 1. CACHE LINE USAGE FOR A 64-KBYTE 2-WAY SET ASSOCIATIVE LEVEL-ONE DATA CACHE WITH 64-BYTE CACHE LINES.	21
FIGURE 2. CACHE LINE USAGE FOR A 64-KBYTE 2-WAY SET ASSOCIATIVE LEVEL-ONE DATA CACHE WITH 256-BYTE CACHE LINES.	21
FIGURE 3. SPATIAL GROUP, MINIMUM FETCH UNIT, AND SPATIAL GROUP GENERATION ILLUSTRATED.	27
FIGURE 4. SPATIAL PATTERN PREDICTOR (SPP) ORGANIZATION ILLUSTRATED.	28
FIGURE 5. A WORKING EXAMPLE (FIGURE NOT FINAL).	29
FIGURE 6. GATED-GROUND CIRCUIT ILLUSTRATED.	35
FIGURE 7. PREDICTION ACCURACY AND COVERAGE OF VARIOUS INDICES FOR PREDICTION.	40
FIGURE 8. PREDICTION ACCURACY AND COVERAGE OF VARIOUS SPATIAL GROUP SIZES.	42
FIGURE 9. PREDICTION ACCURACY AND COVERAGE OF VARIOUS RATIOS BETWEEN SPATIAL GROUP AND FETCH UNIT SIZE.	43
FIGURE 10. PREDICTION ACCURACY AND COVERAGE OF VARIOUS PREDICTOR MEMORY CAPACITY FOR SELECTIVE SUB-BLOCKING CACHES.	45
FIGURE 11. PREDICTION ACCURACY AND COVERAGE OF VARIOUS PREDICTOR MEMORY CAPACITY FOR SPATIAL GROUP PREDICTOR (SGP).	45
FIGURE 12. PREDICTION ACCURACY AND COVERAGE OF 128- AND 256-ENTRY PATTERN HISTORY TABLES OF VARIOUS ORGANIZATIONS.	46
FIGURE 13. EXECUTION TIME INCREASE AND LEAKAGE ENERGY DISSIPATION NORMALIZED TO BASELINE LEVEL-ONE DATA CACHE.	47

Chapter 1

Introduction

“Engineers participate in the activities which make the resources of nature available in a form beneficial to man and provide systems which will perform optimally and economically.”

L. M. K. Boelter

For design simplicity and to optimize for lookup speed, conventional cache memories divide up storage into line frames where data is fetched and stored. The cache line size is fixed at design time [20] and is selected to maximize the exploited spatial locality (due to the likelihood of referencing neighboring data items) while maximizing the exploited temporal locality (due to the likelihood of referencing recently-referenced data items) and limiting the cache line fetch bandwidth requirements.

Recent research [12, 16, 17, 18, 22, 23], however, indicates that there is a large spatial variation 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. On one hand, temporally-close cache references that span large spatial groups favor a single fetch request 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 line frames to optimize for bandwidth and placement. Using excessively large cache lines leads to unused data items 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 [4, 8, 10, 25]).

There have been a number of proposals for improving performance by predicting cache line spatial variations and customizing fetch and placement size upon a cache miss. Unfortunately, these proposals use memory addresses to correlate and predict spatial patterns, and thus either require impractical storage sizes [17], exhibit low prediction accuracy and coverage when using practical storage sizes [16], or lose opportunity in optimizing line size by targeting coarse-grain addressing mechanisms to reduce storage requirement [22].

In this thesis, we propose the Spatial Pattern Predictor (SPP), a table-based predictor that accurately and cost-effectively predicts cache line usage patterns in data caches. Upon a cache miss, an SPP predicts exactly which data items are referenced within a spatial group that consists of a contiguous region in memory. We design two cache optimizations using SPP: (1) selective sub-blocking, in which the predicted as referenced sub-blocks are fetched simultaneously to eliminate subsequent misses 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 the memory latency.

We use cycle-accurate simulation of an aggressive out-of-order superscalar, circuit modeling, and ten SPEC CPU2000 benchmarks using a base 64-Kbyte 2-way set-associative level-one data cache with 64-byte lines to show the following contributions:

1. 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 the referenced data by only 8% for our base cache. Unlike prior proposals, we make the key observation that a combination of program counter (PC) and reference offset (a few bits) within the cache line correlates accurately with spatial patterns, thereby allowing for a small and accurate SPP.

2. Leakage Energy Reduction: Using supply-gating [3] and prediction for unreferenced sub-blocks within a cache line, SPP reduces 41% of the leakage energy dissipation in our base cache with 70nm CMOS technology and incurring less than a 1% performance degradation in all but one benchmark (and 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.
3. Processor Performance Improvement: Prefetching spatial patterns in groups of up to 512B in our base cache using SPP improves performance in the applications 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 this thesis proceeds as follows. Chapter 2 presents an overview of the prior work. Chapter 3 makes the case for the large spatial variation in cache line usage with empirical data. Chapter 4 presents the details of our proposed spatial pattern predictor. In Chapter 5, we discuss two system optimizations using the predictor. In Chapter 6, we quantitatively evaluate of the predictor's accuracy, coverage, storage requirement, and its effectiveness to reduce leakage power and execution time. In Chapter 7, we conclude this thesis and suggest future work.

Chapter 2

Prior Work

“Insanity: doing the same thing over and over again and expecting different results.”
Albert Einstein

There have been a number of proposals in recent literature for predicting spatial variations in caches for improving processor performance. Spatial locality can be predicted statically through profiling [23], or a compiler [18], or at runtime [12, 16, 17, 22] using hardware. In this thesis, to exploit spatial variations to the highest possible extent, we focus on the hardware prediction mechanisms.

A number of proposals adapt cache line size globally (for the entire cache) and dynamically by monitoring cache line footprints. Gonzalez, Aliagas, and Valero [11] 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 [12] suggested using a Spatial Locality Detection table to alternate line fetch sizes between a conventional size and a macroblock size for data with spatial locality. Dubnicki and LeBlanc [9] 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. [22] proposed using a similar algorithm within a conventional uniprocessor cache. Unfortunately, these techniques result in sub-optimal exploitation of spatial variations in caches

because they offer either: (1) coarse-grain adaptation to line size, (2) a single size at a time, or (3) both. In contrast, we propose to predict spatial usage patterns to adapt accurately and with a fine granularity the effective line size on a per cache miss basis.

There are also a number of proposals for spatial pattern predictors in the memory system. Kumar and Wilkerson proposed using the Spatial Footprint Predictor [16] to improve miss ratio in a decoupled sectored cache. Lin et al. [17] proposed density vector predictors that reduce unnecessary traffic when prefetching out of Rambus memory banks. Unfortunately, the proposed predictors use prediction indices that are a function of memory addresses and therefore either require: (1) prohibitively large tables that are impractical to build, (2) increased prediction granularity to reduce storage requirement at the cost of opportunity to optimize line usage, or (3) offer low prediction coverage and accuracy. In contrast, we propose spatial pattern predictors that are both accurate and cost-effective.

None of the prior proposals for exploiting spatial variation in line usage have targeted reducing cache leakage power. Prior proposals to reduce cache leakage have primarily focused on identifying inactive cache regions at the cache line granularity, either through resizing [25] or decay [10, 14]. In contrast, we use spatial pattern prediction in conjunction with a supply gating mechanism [3] to reduce leakage energy dissipation in the unused portions of a cache line.

Chapter 3

Variations in Spatial Locality

Just as recent research shows large variations in cache size demand [26], there is much spatial variation in cache line usage (i.e., the number of neighboring bytes accessed) both within and across programs. In this thesis, we present hardware mechanisms that effectively predict these variations in data caches. 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 thesis.

To motivate the variation in cache line usage, we simulate an aggressive out-of-order processor core running a subset of the SPEC CPU2000 benchmark suite [7] using the SimpleScalar tool set [5]. Table 1 depicts the configuration parameters for the simulated processor. In the interests of simulation turnaround time, 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. We measure as cache line usage the fraction of data referenced in a cache line from when the line is fetched until it is replaced in the level-one data cache. To mitigate the inaccuracies in measurement often introduced by using abbreviated instruction execution streams, we simulate each of the benchmarks to completion.

Table 1. Baseline processor configuration parameters.

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

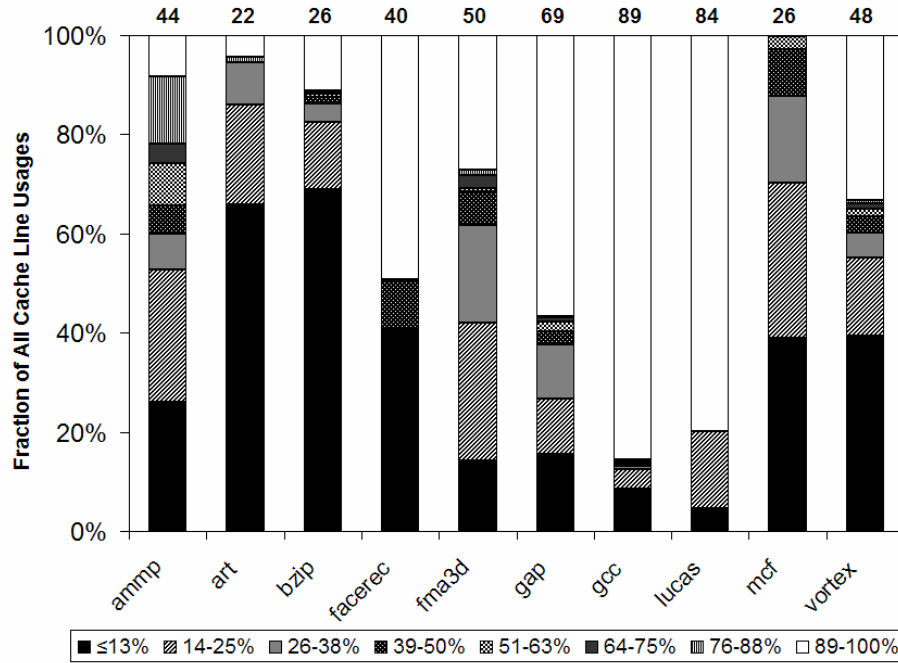


Figure 1. Cache line usage for a 64-Kbyte 2-way set associative level-one data cache with 64-byte cache lines.

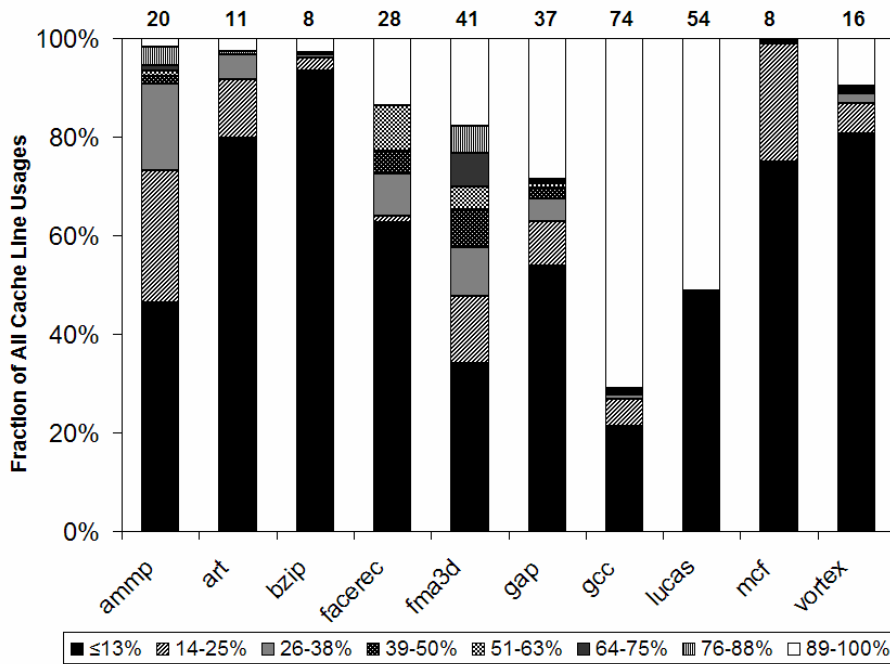


Figure 2. Cache line usage for a 64-Kbyte 2-way set associative level-one data cache with 256-byte cache lines.

Figure 1 and Figure 2 show the distribution of cache line usage in the data cache with 64-byte and 256-byte cache lines, respectively. A minimum data item reference granularity of 8 bytes is assumed. In the interest of presentation clarity, the figure breaks down line usage patterns into a stacked bar histogram. The histogram buckets indicate the fraction of all lines that exhibit a specific fraction of references. 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 figures show a large variation in the efficiency of 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). 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 item (i.e., $\leq 13\%$) is referenced in over 60% of all cache lines when using 64-byte lines. Moreover, *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 2), the overall fraction of fully-referenced lines drops as compared to that of 64-byte lines, but the large variations in line usage persist within and across programs. While the majority of the lines in *gcc* are still fully-referenced, *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 on 256-byte cache lines. In contrast, in *art* and *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 and Figure 2 indicate the average cache line usage. They show that a large fraction (an average of 48% for 64-byte cache lines and 70% for 256-byte cache lines) of data fetched into level-one caches is not referenced. Therefore, a

significant amount of level-one cache capacity and fetch bandwidth is wasted on unreferenced data. Note that the average cache line usage decreases as the cache line size increases, although the actual amount of useful data fetched on a cache miss increases. Intuitively, the locality between the non-adjacent words is looser than that of the adjacent words due to the applications' striding memory access patterns. Thus, larger cache line sizes 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 either exploited for useful data to improve processor performance or unused portions of the cache lines can be turned off to reduce energy dissipation.

Chapter 4

Predicting Spatial Patterns at Runtime

In this chapter we describe a mechanism for predicting spatial locality at runtime. As discussed in Chapter 3, the key observation behind this thesis is that variations in spatial locality exist both across and within programs. Consequently, using statically sized cache lines wastes cache capacity and data fetch bandwidth.

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, potentially a large performance degradation and energy inefficiency by incurring extra accesses to lower level caches. These adverse effects may offset gains achieved by accurate predictions.

The prediction mechanism proposed in this chapter is a simple history-based hardware design; it predicts the future spatial locality of the executing program based on the program's behavior in the recent past. In the sub-chapters 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. For instance, 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 of the spatial groups 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 number of 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 [16]. 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 [24], 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 3 illustrates this definition.

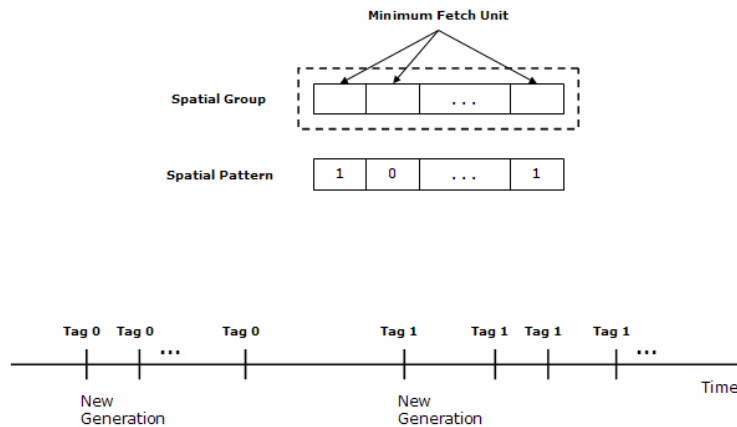


Figure 3. Spatial group, minimum fetch unit, and spatial group generation illustrated.

4.2 A 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 the feedback. The predictor anatomy is depicted in Figure 4. The main component of the feedback mechanism is the current pattern table (CPT), which is used for recording 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.

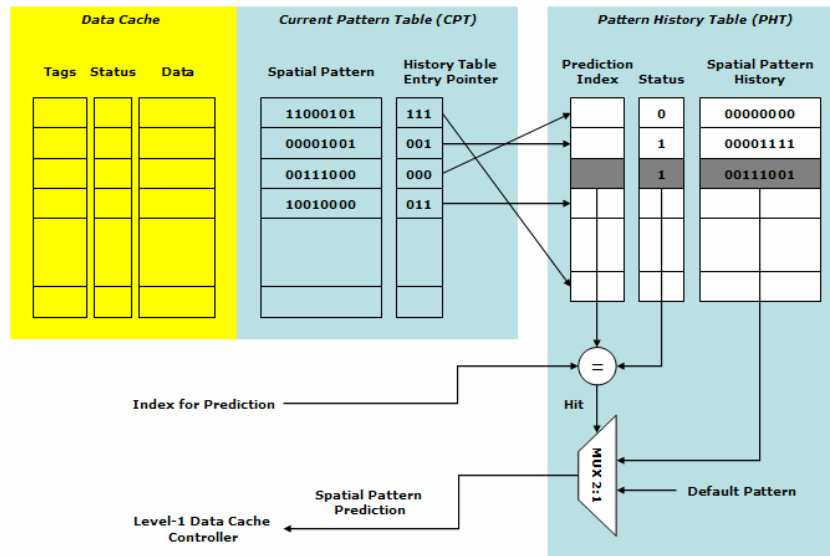


Figure 4. Spatial Pattern Predictor (SPP) organization illustrated.

Inside the predictor memory is the pattern history table (PHT). The PHT is used to maintain 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. At the beginning of a new generation the PHT is read 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 based on the program counter (PC) of the memory instruction or a data address that starts a new generation of the spatial group, traces of PCs or data addresses that access the spatial group, or a combination of these.

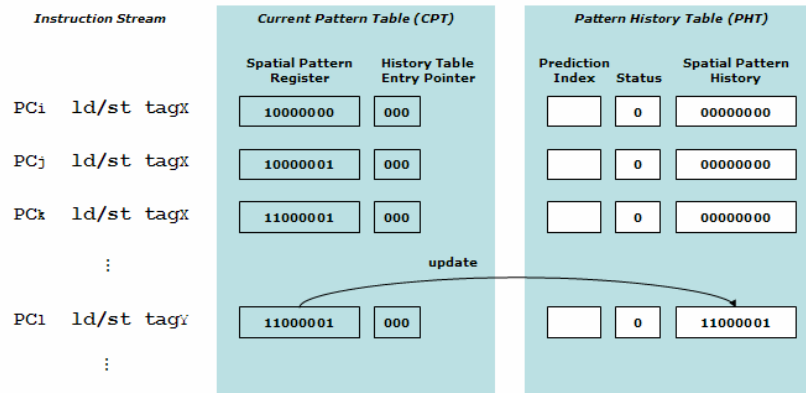


Figure 5. A working example.

4.2.1 A Working Example

Figure 5 illustrates how a spatial pattern predictor learns and predicts in the face 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 the spatial group is initialized to zero (not shown), and then the bit that corresponds to the minimum fetch unit containing the requested word will be set. The 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 with 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.

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.

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 PC of the instruction may be a good indicator for the spatial pattern. However, data structure instances are not necessarily aligned with the spatial group's boundaries, so data address information may be required, significantly increasing the history table size and decreasing predictor coverage. In this thesis, instead of using the full data address, we employ the offset in the spatial group to indicate the alignment of the data field, resulting in efficient predictor designs (see Chapter 6 for discussion).

Another important design choice 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.

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 on 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.

As previously mentioned the predictor probes the PHT at the beginning of a spatial group generation and makes a prediction if a corresponding pattern is found in the table. If the access to the PHT misses, the predictor returns a default spatial pattern which minimizes the performance or energy impact of mispredictions depending on the predictor application (see Chapter 5). Because 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, two types of mispredictions need to be defined to understand the predictor's behavior. 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 the design space will be discussed in Chapter 5 with the application details.

Chapter 5

Predictor Applications

We have shown in Chapter 3 that statically-sized cache lines of conventional designs are not sufficient to capture variations in spatial locality and result in a poor utilization of microprocessor resources. The SPP described in Chapter 4 dynamically captures and predicts the variations in spatial locality. Therefore, the predictor can be applied in various ways to compensate the shortcomings of a fixed cache line size.

In this thesis, we apply the SPP to (1) reduce energy dissipation by disabling the data portion that are fetched but not referenced before eviction, and (2) improve processor performance by prefetching neighboring data that will be referenced soon on a cache miss.

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 [10, 14, 25]. Historically, the primary source of energy dissipation in CMOS transistor devices has been the switching energy; this is due to the charging/discharging of load capacitances when a device switches. However, scaling down transistor supply and threshold voltages to reduce switching energy consumption and maintain performance gives rise to significant leakage energy dissipation due to an exponential increase in subthreshold leakage current [4, 8]. Moreover, state-of-the-art microprocessor designs now

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 [15] and 60% of the StrongARM processor is devoted to cache and memory structures [19]. Because leakage energy is a function of the number of on-chip transistors, independent of their switching activity, leakage energy dissipation in caches accounts for an increasingly large component of energy dissipation in recent designs and will continue to do so in the future.

There have been a number of architectural techniques proposed to reduce leakage energy dissipation in cache memories in recent literature [10, 14, 25]. These techniques control the leakage energy dissipation at the granularity of one or more cache lines. The technique proposed in this thesis contrasts with previous approaches in that we use the SPP to exploit the usage variation and underutilization within a single cache line that are quantified in Chapter 3. The SPP is used to determine the cache line's underutilization and reduce leakage energy dissipation from the data portions of a cache line that are not referenced before eviction.

To exploit usage variations 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. Therefore, a single spatial pattern is recorded and predicted for each cache line at the granularity of the sub-block. A spatial group generation is thus the same as the cache line generation [24], an interval between two consecutive misses on the same cache line frame. 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 energy mode. The rest of the sub-blocks are put into a standby energy mode to reduce leakage energy dissipation. We call this cache a selective sub-blocking cache. In training, the predictor conservatively enables all the sub-blocks to minimize the performance impact. The details of the circuit technique and the energy/performance implications will be discussed in following sub-chapters.

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]. 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, essentially “gating” the cell’s supply voltage. Gated-ground maintains the performance advantages of lower supply and threshold voltages, while reducing leakage. Figure 6 depicts the anatomy of a data retention gated-ground technique with the gated-ground transistor shared across a sub-block. The circuit-level details of the gated-ground technique can be found in [3, 21].

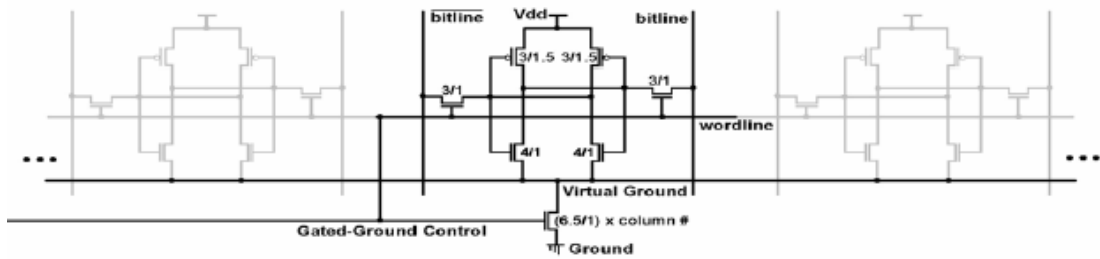


Figure 6. Gated-Ground circuit illustrated.

Data retention gated-ground enables a selective sub-blocking cache to effectively turn off the supply voltage and virtually eliminate the leakage in disabled sub-blocks while preserving the data values. By preserving the data in the disabled sub-blocks, 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 circuit 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 to turn on the gated-ground transistor shared by a 16-byte sub-block is only 0.20ns, which is one processor cycle in an aggressive 5-GHz microprocessor.

5.1.2 Impact on Energy and Processor Performance

As reported in Chapter 3, a large fraction of data brought into a cache line is unreferenced. Thus, the SPP is expected to save a significant amount of leakage energy dissipation using the data retention gated-ground technique. However, mispredictions and the training of the predictor have implications on the total energy dissipation and performance of the caches.

Overprediction and underprediction in the case of a misprediction have different impacts on energy and performance. Overprediction, which mistakenly enables unreferenced sub-blocks, decreases the effectiveness of the predictor and reduces energy savings. In underprediction, the disabled sub-blocks must be enabled before the sub-blocks can be accessed. Therefore, 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 speculative loads that are typically applied in highly speculative modern processors.

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. However, the high coverage and accuracy of the predictor (see Chapter 6 for evaluation) ensures that the existing opportunity for energy savings is closely tracked, without incurring a significant performance degradation.

The hardware structures used in the predictor also consume energy. The energy overhead of the CPT can be minimized by combining the table with the cache tag array. Therefore, 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, and thus the energy overhead is minimal. The PHT's energy overhead is also minimal. The results in Chapter 6.3 show that the PHT can be designed as a 256-entry tag less RAM table, with each entry containing one spatial pattern. Moreover, the table is accessed infrequently, only on cache misses. The effectiveness of using the SPP to reduce leakage energy will be evaluated in Chapter 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 shortcoming of the conventional memory hierarchy organization. Prefetching uses predictions on future memory usage patterns to help fetch data in advance of its use to hide the memory latency. In this chapter, we present the Spatial Group Prefetcher (SGP) as an application of the SPP for improving processor performance. An SGP predicts neighboring data items that should be fetched with respect to 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 cache lines tends to decrease as the cache line size increases. Although larger cache lines may help exploit the high spatial locality of executing programs and improve miss rates, they also impose higher bus bandwidth requirements and greater miss penalties. As a result, a processor with a larger cache line size 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 wide lines and the bandwidth efficiency of smaller cache lines [6].

To apply an SPP to data prefetching, we group a number of 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 number of 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 is different from the tag whose spatial pattern is being recorded. Note that a request to a tag different from the recording tag does not necessarily occur during a miss; this is 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 tag that differs with the current recording tag can indicate that the reference stream in the vicinity of the current recording tag has terminated.

An SGP makes a prediction on 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 using the SGP to improve processor performance will be evaluated in Chapter 6.5.

Chapter 6

Evaluation

“Where are my numbers?!”
Babak Falsafi

In this chapter we conduct a sensitivity study to explore the design space of an SPP to search for a best practical configuration. The effectiveness of using an 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 Chapter 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 sub-chapters, we assume a PHT with an unlimited number of entries to filter out artifacts introduced by having an insufficient predictor memory. In Chapter 6.3, we consider a variety of practical implementations and organizations of a PHT. In Chapter 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 each of the example applications.

In the sensitivity study, with the purpose of mitigating the unrepresentative measurements often introduced by using abbreviated instruction execution streams, we simulate each of the benchmarks to completion. In the interest of simulation turnaround time, we elect to skip the first 10 billion instructions and simulate the next 500 million instructions for each of the benchmarks with our timing simulator. The configuration parameters for the simulated processor are depicted in Table 1.

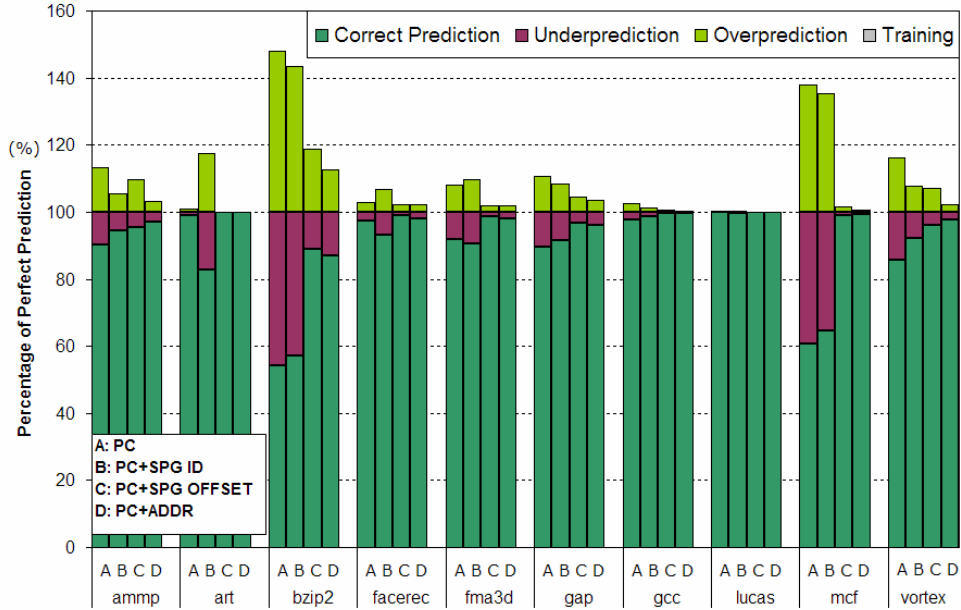


Figure 7. Prediction accuracy and coverage of various indices for prediction.

6.1 Index for Prediction

We consider a variety of indices for prediction that are based on the program counter (PC) of faulting memory instructions. The key observation behind using a PC-based index for prediction is that instructions provide a concise representation of history [13]. 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. Figure 7 presents the prediction accuracy and coverage of a spatial group predictor for a 64-Kbyte 2-way set associative level-one data cache. The graph plots the percentage of correct spatial pattern predictions, the fraction of underpredictions, and the fraction of overpredictions. Figure 7 also shows the fraction of spatial patterns not predicted due predictor training with respect to a perfect predictor that always knows exactly what fetch units will be referenced, given an index for prediction. Because we assume a PHT with unlimited capacity, there are few spatial patterns not predicted due to training. The graph indicates that the index for prediction selection can have a profound impact on the predictor's accuracy. Using a 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. 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 locations requires a large number of entries in order to maintain satisfactory prediction coverage in the presence of a large working set. On the other hand, using just a PC as the index for prediction yields the poorest prediction accuracy among all the considered indices for prediction. Our observation is in agreement with Kumar and Wilkerson [16]. 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 spatial group tag yields low prediction accuracy because spatial group tags do not contain the alignments of the data locations being accessed by a particular memory instruction. With *bzip2* and *mcf*, in particular, the indices that do not pertain to alignments within a spatial group of data locations cause 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.

To address the high predictor memory capacity requirements imposed by the PC-and-data-address based index for prediction 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 index for prediction is that a given memory instruction sequence 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 may require more predictor memory than a PC-only

based index for prediction because each unique offset will require a separate entry in the PHT. Nevertheless, the index for prediction 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 that is accessed by the same memory instruction and has the same offset within a spatial group.

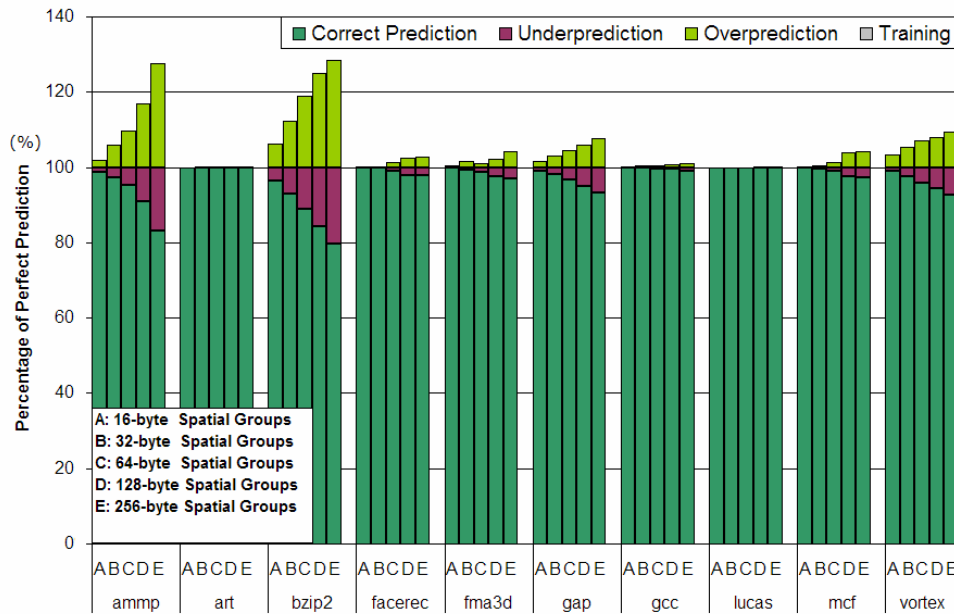


Figure 8. Prediction accuracy and coverage of various spatial group sizes.

6.2 Spatial Group and Fetch Unit Size

The spatial group size can affect prediction accuracy of an SGP as well. Figure 8 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 size of the spatial group increases is that memory instructions within a basic control block tend to access data in close vicinity; as the size of a spatial group increases it might cover data that will be

accessed by multiple basic control blocks with different memory behaviors. Since the predictor uses only the program counter of faulting memory instructions and spatial group offset as the index for prediction, 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 index for prediction recorded during different generations may not be the same. Prediction accuracy with larger spatial group sizes might also decrease if the spatial groups span across data sets accessed from multiple basic blocks that are separated by branch instructions.

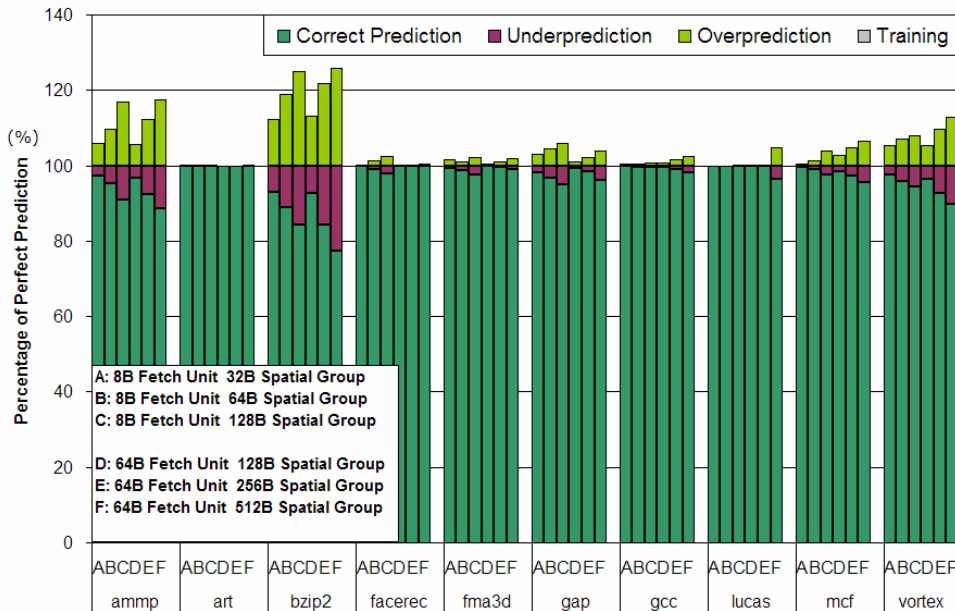


Figure 9. Prediction accuracy and coverage of various ratios between spatial group and fetch unit size.

In addition, 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 control 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, on the other hand, tend to be less sensitive to the

size a spatial group with the exception of *ammp*. To contain the effect of aliasing instruction sequences, one can combine multiple spatial pattern histories captured in different spatial group generations to generate a prediction for each index, in which case the predictor will be biased towards overprediction. Alternatively one can increase coarseness of the prediction by using a larger fetch unit size as indicated by Figure 9.

6.3 Predictor Memory Capacity and Organization

Figure 10 shows the prediction accuracy and coverage of an SPP for a selective sub-blocking cache with a PHT of various 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 11 presents the prediction accuracy and coverage of spatial pattern predictor with various pattern history table capacities for an SGP. 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 different spatial data structure offsets referenced in a given program phase.

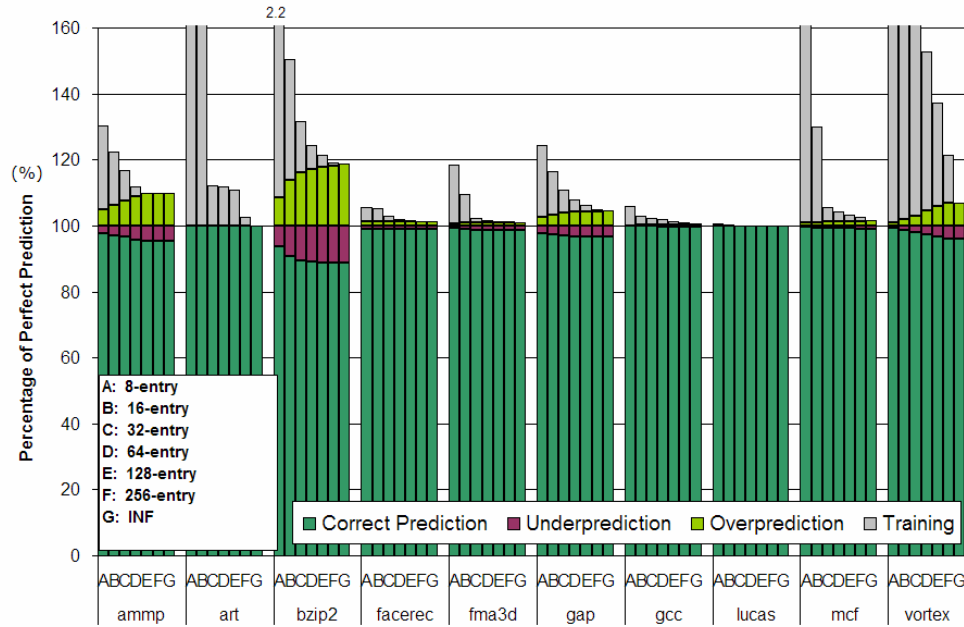


Figure 10. Prediction accuracy and coverage of various predictor memory capacity for selective sub-blocking caches.

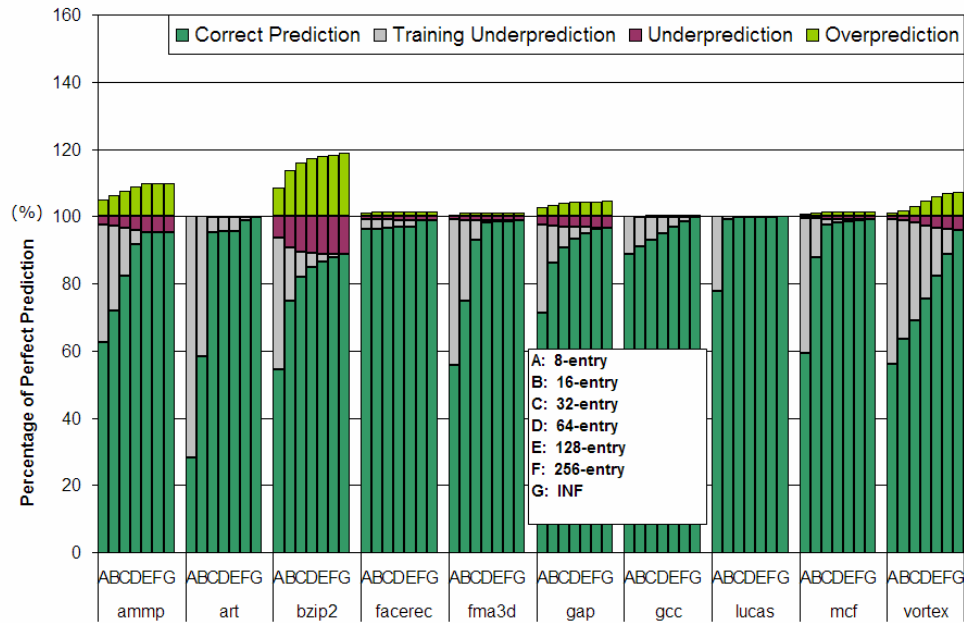


Figure 11. Prediction accuracy and coverage of various predictor memory capacity for Spatial Group Predictor (SGP).

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 a table with an unlimited number of entries, in general prediction coverage of an SPP degrades noticeably 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 memory instruction sequences that are concurrently active. Similarly, *vortex*, which is 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.

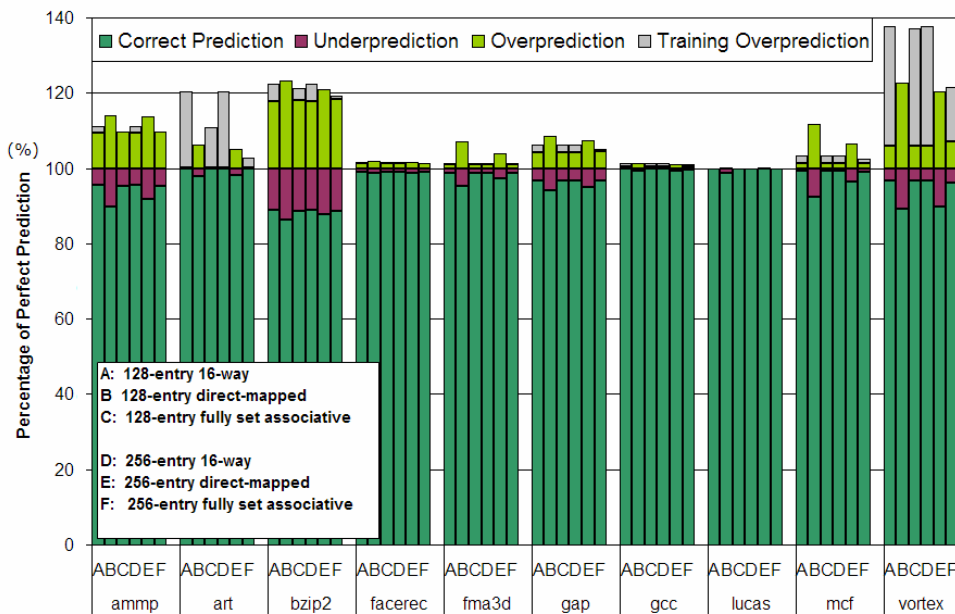


Figure 12. Prediction accuracy and coverage of 128- and 256-entry pattern history tables of various organizations.

Figure 12 presents the prediction accuracy and coverage of 128- and 256-entry PHTs with various organizations for selective sub-blocking caches. For the interest of space, the results of the predictor for 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 prediction accuracy that is comparable to an ideal fully associative table, they incur a severe degradation in prediction coverage. The impact on prediction coverage is especially noticeable in art and vortex. On the other hand, the direct-mapped tables have lower prediction accuracy, but they yield performance closer to an ideal fully associative table.

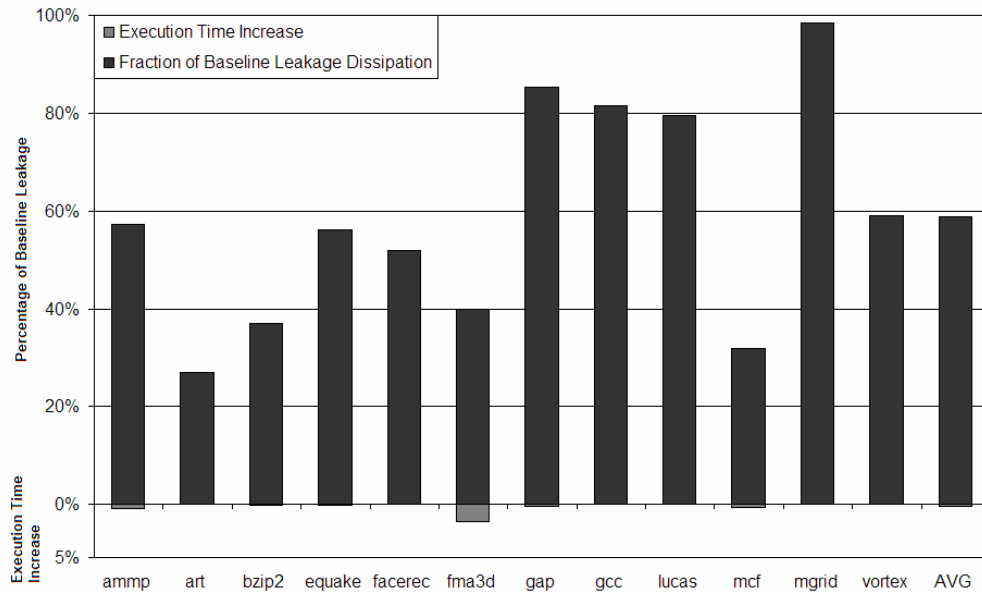


Figure 13. Execution time increase and leakage energy dissipation normalized to baseline level-one data cache.

6.4 Energy Reduction Effectiveness

Figure 13 shows the leakage energy dissipation in a 64-Kbyte, 2-way set associative level-one data cache (upper bars) and the increase in execution time of the programs (lower bars). The leakage energy dissipation is normalized to a conventional data cache. The graph does not take into account the predictor energy dissipation overhead because the leakage energy dissipated by the CPT and the PHT is negligible as explained in Chapter 5.1. The graph indicates that selective sub-blocking is effective in reducing leakage energy dissipation; on average the technique helps

reduce 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.

6.5 Data Prefetching Performance

In this section, we evaluate the Spatial Group Prefetcher's (SGP) effectiveness in improving application performance. 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 longer cache fill time and contention at level-one/level-two incurred by larger cache line sizes, the simulator is augmented to model a realistic bus implementation. The bus always gives priority to processor requests over SGP prefetch requests.

Table 2. Performance comparison of SGP against demand-fetched level-one data caches with various cache line sizes.

	<i>ammp</i>	<i>art</i>	<i>bzip2</i>	<i>equake</i>	<i>facerec</i>	<i>fma3d</i>	<i>gap</i>	<i>gcc</i>	<i>lucas</i>	<i>mcf</i>	<i>mgrid</i>	<i>vortex</i>	AVG
256B	-9	4	10	-2	-4	-2	7	0	0	-6	10	-4	-1
512B	-41	32	-43	-34	-13	-9	20	-2	-23	-27	-27	6	-13
1024B	-63	96	-49	-41	-3	-9	31	-2	-67	-32	12	-43	-14
SGP-256B	7	15	3	14	9	-2	9	0	8	6	14	0	7
SGP-512B	10	121	6	59	58	0	31	1	34	38	36	1	33
SGP-1024B	-25	305	8	99	103	0	47	1	51	67	53	1	59

Table 2 presents percent speedup of an SGP with 256-, 512-, and 1024-byte spatial groups and demand-fetched level-one 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 higher bus traffic. As indicated by the graph, many benchmarks see their execution time increase as 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 by the program, it can eliminate bus traffic caused by fetching redundant data. On average, 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*, 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*, SGP does not improve performance because SGP prefetches approximately the same amount of data as a conventional cache with 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 using a prefetch buffer where the prefetched cache lines are stored until they are referenced. Design and evaluation of such a buffer is beyond the scope of this work.

Chapter 7

Conclusion

In this thesis, we described the Spatial Pattern Predictor (SPP), cost-effective table-based hardware predictors that accurately predict the reference patterns within a spatial group at runtime. The key observation enabling accurate yet low-cost SPP design is that spatial patterns correlate well with instruction addresses and reference offsets within a line, requiring a small number of table entries to store the predicted patterns. Using SPP, we designed two cache optimizations: (1) selective sub-blocking, in which the predicted as 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 a predicted number of 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 level-one 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 2% performance degradation, and (3) prefetching spatial groups of up to 512 bytes improves execution time on average by 33% and up to a factor of two.

Bibliography

- [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.
- [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] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, July 1999.
- [5] 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.
- [6] 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.
- [7] Standard Performance Evaluation Corporation. SPEC CPU2000 Benchmark Suite. In <http://www.spec.org>, 2000.
- [8] 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.
- [9] 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.
- [10] 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.

- [11] 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.
- [12] T. Johnson 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.
- [13] 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.
- [14] 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.
- [15] K. Krewell. Itanium 2 arrives with a benchmarking bang. In *Microprocessor Report*, Aug. 2002.
- [16] 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.
- [17] 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.
- [18] C.-K. Luk and T. C. Mowry. Compiler-based prefetching for recursive data structures. In *ASPLOS96*, Oct. 1996.
- [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] J. Hennessey and D. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, 2001.
- [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] 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.
- [23] 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.
- [24] 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.
- [25] 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.
- [26] S.-H. Yang, M. D. Powell, B. Falsafi, and T. N. Vijaykumar. Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay. In *Proceedings of the Eighth IEEE Symposium on High-Performance Computer Architecture*, pages 151–161, Feb. 2002.