Exploiting Spatial Locality in Data Caches using Spatial Footprints*

Sanjeev Kumar
Department of Computer Science
Princeton University
skumar@cs.princeton.edu

Christopher Wilkerson
Microcomputer Research Labs
Intel, Oregon
cbwilker@ichips.intel.com

Abstract

Modern cache designs exploit spatial locality by fetching large blocks of data called cache lines on a cache miss. Subsequent references to words within the same cache line result in cache hits. Although this approach benefits from spatial locality, less than half of the data brought into the cache gets used before eviction. The unused portion of the cache line negatively impacts performance by wasting bandwidth and polluting the cache by replacing potentially useful data that would otherwise remain in the cache.

This paper describes an alternative approach to exploit spatial locality available in data caches. On a cache miss, our mechanism, called Spatial Footprint Predictor (SFP), predicts which portions of a cache block will get used before getting evicted. The high accuracy of the predictor allows us to exploit spatial locality exhibited in larger blocks of data yielding better miss ratios without significantly impacting the memory access latencies. Our evaluation of this mechanism shows that the miss rate of the cache is improved, on average, by 18% in addition to a significant reduction in the bandwidth requirement.

1. Introduction

This paper introduces an approach to alleviate the growing memory latency problem [3] by better exploiting the *spatial locality* exhibited by applications. Spatial locality is the tendency of neighboring memory locations to be referenced close together in time. The traditional approach for exploiting spatial locality is to use a cache line (consisting of several words) to fetch data into the cache on a cache miss. This prefetching improves the cache performance when a reasonable fraction of the prefetched words get used. Otherwise the excessive cache pollution hurts performance by prematurely evicting potentially useful data. Since spatial locality varies across different applications as well as within an application, cache designers generally choose a line size with the best average performance across a broad range of reference patterns. Although traditional caches benefit from spatial locality, they exploit only a small fraction of the spatial locality available.

As the available chip area increases, cache effectiveness can be improved by using bigger caches. However, bigger caches increase access latency, thereby requiring more, or longer, cycles to access them. The cache access latency is especially important in first-level (L1) caches where short single-cycle accesses are desirable. Our approach is to use some of the available chip area to implement predictors that better exploit spatial locality. The predictors minimize the impact on the access latency by keeping the bulk of their operations off the critical path.

Several hardware and software based prefetching mechanisms that improve the effectiveness of the caches have been studied. However, most of the work [16, 9, 2, 5, 13] has focussed on numeric applications with well-structured loops and regular access patterns. Other efforts [10, 12] have focussed on using the compiler to prefetch pointer targets on integer applications. However, very little work has been done to exploit the spatial locality in integer applications based on the dynamic behavior of the application. The SLDT [7] mechanism attempts to detect and exploit the spatial locality at run time.

In this paper, we present a new predictor, called the *Spatial Footprint Predictor* (SFP), which predicts the neighboring words that should be prefetched on a cache miss. The accuracy of the predictor allows us to build a cache (which uses small lines and fetches multiple lines on a miss) to exploit the spatial locality available to large cache lines without excessive pollution. We also present mechanisms that use the predictor to improve the miss rates with little impact on the cache access latency. We show that these mechanisms significantly reduce the miss rates and the bandwidth in first-level caches for popular real-world applications. Our simulations show the mechanism can potentially reduce the miss rates by 35% on average. A practical implementation reduces the miss rates by around 18%. The scheme requires

*Research conducted at Microcomputer Research Labs, Intel, Oregon.

no modifications to the Instruction Set Architecture (ISA).

2. Spatial Locality

Memory hierarchies rely on locality in the application's memory reference stream to deliver performance. The locality can be classified into two categories—temporal and spatial. Temporal locality is the property whereby a reference to a memory location indicates that the same location will very likely be referenced again in the near future. Spatial locality is the property whereby a reference to a memory location indicates that a neighboring location will very likely be referenced in the near future. Temporal locality is exploited by having a cache between the processor and the memory that is big enough so that a word gets accessed multiple times before getting evicted. Spatial locality is exploited by using blocks spanning multiple words to move data between the different layers of the hierarchy. We study the spatial locality exhibited by applications at the data cache level and present mechanisms to better exploit it.

Today's cache designs exploit spatial locality in applications at the cache line granularity. On a cache miss, the cache fetches the entire line containing the reference. The words within the line that get referenced before the line is evicted reduce the number of the misses incurred by the application, while the words that do not get referenced before eviction potentially hurt performance by polluting the cache. However, since the spatial locality exhibited by applications for a given cache size varies significantly, different line sizes yield the best performance for the different applications. So, a line size that performs well across a wide range of applications has to be chosen. In addition, other considerations like reducing the amount of tag storage (which requires larger lines) and reducing the amount of false sharing on multiprocessors (which requires smaller lines) influence the selection of the "best" line size. The result is that caches benefit only from a small fraction of the spatial locality exhibited by the application.

The spatial locality varies not only across different applications but also within the different parts of the same application. A few applications have uniform spatial locality behavior and perform well when using a fixed cache line size. However, for most integer applications, the spatial locality exhibited is fairly non-uniform. This variation is hard to predict statically and, therefore, exploiting the spatial locality effectively requires run-time prediction mechanisms.

2.1. Related Work

Several studies [14, 17] have examined the effects of different cache line sizes on cache performance. One of the studies [14] also investigated the effect of varying the fetch size independent of the line size and concluded that the optimal statically determined fetch size was generally twice the line size. The dual data cache [6] has two independent parts for data that exhibit temporal and spatial locality. However, their locality detection mechanism was geared for numeric codes with constant stride vectors. The Spatial Locality Detection Table [7] attempts to dynamically pick the right fetch size based on the memory reference. However, it does it at a much coarser granularity than our mechanism. They also do not address the problem of increased L1 cache access latency associated with small cache lines.

Abraham *et al.* [1] show that a large percentage of misses are caused by a small percentage of instructions. They use this information to prefetch data referenced by a few instructions identified through profiling. Tyson *et al.* [18] use this information to dynamically make cache bypassing decisions. Another study [8] shows that, in some applications, few instructions reference large amounts of data with widely varying data access patterns. So they use the data address to dynamically make cache bypassing and prefetching decisions [8, 7]. We study the effectiveness of using both instruction and data addresses to predict spatial locality.

2.2. Approach

The goal is to exploit the spatial locality available to long cache lines without excessive pollution and without significantly impacting the cache access latency. For this, we use a cache with small lines and fetch multiple neighboring cache lines on a cache miss based on the run-time behavior of the application. To achieve this at a reasonable cost, we need to address two problems. First, we need a predictor that predicts the neighboring lines that are very likely to be used in the near future. Second, we need to reduce the amount of tag space required by caches with small lines.

We introduce some terminology used in this paper. A cache line is the smallest replaceable unit in a cache. Cache lines are grouped into aligned groups of adjacent lines in the virtual address space called *sectors*. Sectors are divided into two groups—*inactive* and *active*. All sectors are initially inactive. The cache lines within a sector that get referenced while it is active define the *spatial footprint* of that sector. A spatial footprint for a sector is stored as a bit vector with a bit for every line in the sector.

We propose a class of predictors called Spatial Footprint Predictors that exploit spatial locality at sector granularity. While a sector is active, its spatial footprint is recorded. When a sector is inactive, a cache miss activates it and causes the predictor to predict the spatial footprint for that sector based on previously recorded footprints. This allows the cache to fetch only those lines within the sector that are very likely to be used in the near future.

Tagging a cache with small lines at cache line granular-

Benchmark	Instructions (millions)	References (millions)	Miss Ratio	MBytes Fetched
draw	38.9	1.7	5.80	54.0
gcc	26.5	0.2	1.88	7.6
go	18.6	0.4	4.19	12.6
pc_db	30.0	0.6	2.77	19.0
pres1	46.8	0.7	2.48	22.3
pres2	43.7	0.9	3.92	29.1
specweb	69.1	2.7	11.81	69.3
sprdsht	36.0	0.5	2.09	15.6
tpcc	32.4	0.3	6.69	9.9
tpcc_long	148.8	1.4	3.27	45.8
wdproc1	41.4	0.4	1.54	13.3
wdproc2	30.0	0.4	1.79	12.0

Figure 1. Benchmark Characteristics

ity requires bigger tag arrays and longer access times. An alternative is to use sectored caches which tag the cache at sector granularity and use a bit-vector to store validity information for the cache lines within the sector. Although this approach would benefit from reduced bandwidth requirements, the unused cache lines within the sectors would result in poor utilization of the cache, resulting in worse hit ratios and, ultimately, worse cache performance. We use a decoupled sectored cache [15] (described in Section 6.2) that allows us to use smaller arrays with little impact on the cache access latency.

3. Experimental Setup

We study Spatial Footprint Prediction for L1 data caches. In the base configuration, the L1 data cache used is a 16 K-byte four-way set-associative cache. Each sector contains 16 lines while each line contains 8 bytes. The cache is also write-through and fetch-on-write.

Analysis was performed using the Intel MRL reference traces which includes the following:

- 2 integer traces from SPEC (go and gcc)
- 2 transaction processing server traces—one with 325 warehouses and 60 clients (tpcc) and another with 50 warehouses and 36 clients (tpcc_long).
- 1 web server trace obtained from SPECweb96.
- 7 popular PC-based applications including a database application, 2 presentation applications, 2 word processing applications, a spreadsheet application and a drawing program. All application traces were obtained from the benchmark suite released by BAPCO sysmark32. Typically, the applications are interactively driven by a user via a graphical user interface. During tracing, the applications were automated using scripts. The functionality exercised in the applications was chosen to be representative of commonly used ones like spell-checking and reformatting.

All 12 traces are full system address traces including all activity from both the OS (NT4.0) and the application including any activity in dynamically linked libraries. The traces include all instruction and data addresses generated by the program but specify no other instruction information. All 12 traces were gathered on an Intel Architecture processor running a complex instruction set architecture, and were chosen both for their popularity in the PC market segment and for their high Misses Per Instruction.

The cache simulator measures the cache miss ratio and the fetch bandwidth. The fetch bandwidth is the number of bytes fetched into the cache due to both read and write operations. It does not include the store bandwidth due to the writes because it is not affected by our implementation. The miss ratio and the fetch bandwidth for the base configuration are presented in Figure 1. The miss ratios are lower than on most other platforms. This is because the small number of architectural registers available on the Intel platform causes a larger fraction of local variables to be stored in memory instead of registers. The extra references generated, usually, have very good locality properties and hit in the first-level cache. The result is that the first-level caches sees more memory references and a lower miss ratio. However, the performance of the first-level cache has a significant impact on the performance of the applications. All numbers presented in the remainder of the paper are normalized to the base configuration.

Our primary goal is to reduce the miss ratio of the L1 cache by exploiting spatial locality available to long cache lines while keeping pollution at a minimum. The difference between fetch bandwidth in the different configurations gives us an idea of the amount of pollution.

4. Spatial Footprint Predictors

Before describing the predictors, we introduce some more terminology. A memory reference that activates an inactive sector is referred to as a *nominating reference*. The sector containing the nominating reference is the *nominating sector*; the cache line containing the nominating reference is the *nominating line*; and the instruction that generated the nominating reference is the *nominating instruction*.

Spatial footprint prediction consists of two parts: a mechanism to measure the spatial footprint of the active

sectors, and a mechanism that uses the spatial footprint history to predict future footprints. We use two tables (Figure 3) in our implementation—a Spatial Footprint History Table (SHT) and an Active Sector Table (AST). The SHT is used to store some of the previously recorded footprints. Each entry consists of a tag to resolve conflicts among the entries and one or more of the previously recorded footprints depending on the amount of history used by the predictor. The index into the SHT also depends on the predictor. The AST is used to record the footprint while a sector is active in the cache and is indexed by the sector address. Each entry consists of a tag to resolve conflicts, the index into the SHT (SHTi), and a bit vector to record the footprint. In addition, it contains the nominating line number[1] (LN) and a flag whose use is described in Section 5.

The predictors we studied differ primarily by how they index the SHT and the amount of spatial footprint history used. They use some combination of bits from the nominating reference address and the nominating instruction to generate the SHT index. In addition, they use the last one or two footprints stored in the corresponding SHT entry. When the history consists of more than one footprint, a bitwise OR is used to generate the predicted footprint.

The predictors are named based on the SHT index (superscript) and the amount of history used (subscript). The data address based predictors use either the nominating sector address (SFP_1^{SA}) or the nominating line address (SFP_1^{LA} and SFP_2^{LA}). Using the instruction address alone as the index yields poor results because the different data locations accessed from a single instruction are not aligned at the same offset within the sector. To deal with the different alignment, the footprints can be shifted so that the nominating lines are aligned with each other. However, shifting the footprint hurts the predictor accuracy because it requires guessing the few bits that get shifted in. The alternative is to use the nominating instruction address along with the nominating line number ($SFP_1^{IA,LN}$). Finally, we present a predictor ($SFP_1^{IA,DA}$) which uses the nominating instruction address along with the nominating reference address.

The predictors are quite similar. On a cache miss in an inactive sector, the predictor predicts a spatial footprint that is used to fetch lines. It activates that sector, allocates an entry in the AST for it and initializes the footprint bit vector and the SHT index. Every subsequent reference to a line within that sector causes the corresponding bit in the footprint to be set. When the sector is deactivated, the recorded footprint is migrated into the SHT and the AST entry is invalidated.

[1] The 4 bits in the data address that specifies the nominating line within the sector.

4.1. Experimental Evaluation

In this section, we evaluate the various Spatial Footprint Predictors using a sectored cache. All the lines within a sector are fetched on a miss—the predictors are not used to fetch lines selectively. In this setup, a sector is active while it is in the cache and it becomes inactive when it is evicted. While the sector is active, its footprint in recorded. When the sector is evicted, the recorded footprint is compared with the predicted footprint from each of the predictors[2] to measure their accuracy.

This approach has two benefits. First, it allows us to study the predictors in isolation from the specific design choices made in a practical implementation (Sections 5 & 6). Second, the measurements made in this section are useful in explaining the results obtained in Sections 5 & 6.

We use a 16 KBytes cache throughout this paper. However, in this section, we use a 32 KBytes cache to compensate for the inefficient use of cache resources due to the use of long sectors. This is done to get a better correlation between the results in this section and the results in the other sections. As always, the cache has 8 byte lines and 16 lines per sector. Also, we use infinite tables for AST and SHT to avoid any conflict among entries.

We measured three different aspects of the various predictors—the number of lines missed on average, the number of extra lines fetched on average, and the percentage of times the predictor failed to make a prediction for lack of history (Figure 2). The first two measure the predictor accuracy while the last one, coupled with the size and associativity of the SHT, determines the coverage of the predictor.

Comparing the line-based predictor (SFP_1^{LA}) against the sector-based predictor (SFP_1^{SA}) shows that SFP_1^{LA} is consistently more accurate that SFP_1^{SA}. The difference is especially dramatic in the case of *pres2*. This is probably the result of several different footprints being associated with the same sector. The surprising result, however, is that the SFP_1^{LA} suffers only twice as many cold misses in the SHT as the SFP_1^{SA}. This implies that, on average, only two of the 16 lines in a sector are nominating lines.

The instruction-based predictor ($SFP_1^{IA,LN}$) performs about as well as the sector-based predictor (SFP_1^{SA}). However, in one of the benchmarks (*specweb*), it is more accurate and suffers fewer cold misses than any of the data address based predictors.

The predictor that uses both the instruction address and the data address ($SFP_1^{IA,DA}$) is consistently more accurate than all other predictors. However, it also suffers more cold

[2] Predictors are usually invoked at sector activation time. However, invoking them at deactivation time (which is when they are needed in this section) generates the same footprint.

	Benchmark	SFP_1^{LA}	SFP_2^{LA}	SFP_1^{SA}	$SFP_1^{IA,LN}$	$SFP_1^{IA,DA}$
Number of Lines Missed Per Sector	draw	0.06	0.03	0.35	0.37	0.03
	gcc	0.88	0.53	1.07	1.23	0.32
	go	0.38	0.29	0.40	0.40	0.27
	pc_db	0.45	0.28	0.64	0.42	0.21
	pres1	0.19	0.12	0.29	0.25	0.11
	pres2	0.23	0.12	1.25	1.70	0.16
	specweb	0.31	0.19	0.38	0.13	0.07
	sprdsht	0.38	0.23	0.55	0.40	0.23
	tpcc	0.44	0.21	0.68	0.40	0.26
	tpcc_long	0.34	0.16	0.58	0.53	0.23
	wdproc1	0.36	0.20	0.67	0.51	0.21
	wdproc2	0.43	0.26	0.58	0.46	0.26
	Mean	0.40	0.24	0.68	0.62	0.21
Number of Extra Lines Fetched Per Sector	draw	0.06	0.08	0.57	0.37	0.04
	gcc	0.93	1.42	1.27	1.22	0.35
	go	0.38	0.66	0.76	0.40	0.26
	pc_db	0.47	0.71	0.83	0.42	0.20
	pres1	0.20	0.30	0.39	0.25	0.11
	pres2	0.23	0.34	1.58	1.70	0.16
	specweb	0.31	0.50	0.48	0.13	0.07
	sprdsht	0.39	0.59	0.72	0.39	0.22
	tpcc	0.44	0.64	0.82	0.40	0.26
	tpcc_long	0.35	0.49	0.70	0.53	0.23
	wdproc1	0.39	0.56	0.89	0.51	0.21
	wdproc2	0.43	0.64	0.74	0.45	0.25
	Mean	0.42	0.63	0.89	0.62	0.21
Percentage of Cold Misses in SHT	draw	3.03	3.03	1.77	0.24	3.73
	gcc	14.27	14.27	6.44	11.78	43.30
	go	2.42	2.42	0.67	5.08	34.60
	pc_db	10.15	10.15	4.94	6.02	17.48
	pres1	9.01	9.01	6.45	3.78	12.09
	pres2	3.79	3.79	2.08	2.28	5.49
	specweb	2.43	2.43	1.37	0.51	3.56
	sprdsht	5.92	5.92	2.73	5.83	10.82
	tpcc	6.60	6.60	3.95	3.28	10.11
	tpcc_long	8.06	8.06	5.63	1.44	17.93
	wdproc1	12.31	12.31	6.56	6.93	19.35
	wdproc2	10.18	10.18	5.46	9.26	17.23
	Mean	8.02	8.02	4.37	5.13	17.79

Figure 2. Characteristics of the Spatial Footprint Predictors

misses in the SHT. The difference is huge in the case of the two SPEC benchmarks (*gcc* and *go*).

Using more spatial footprint history allows additional line fetches to be traded for fewer missed lines. By using a bitwise OR to combine the footprints, we bias the predictors towards reducing the number of lines missed. Comparing the entries for SFP_1^{LA} and SFP_2^{LA} shows that, on average, the numbers of lines missed by the predictor is decreased by 50%, while the number of extra lines fetched increases by approximately the same amount.

Since we do not measure the miss ratios in this section, we cannot directly compare these predictors with traditional caches. However, to put things in perspective, we present some back of the envelope calculations to compare the amount of prefetching and pollution due to SFP predictors with the amount of prefetching and pollution that occurs in traditional caches which use 32-byte lines. We use 8 bytes as the smallest unit here.

When running *gcc*, the traditional cache brings in 4 units of data each time and does not use around 40% (1.6) of them before eviction. So each miss prefetches 3 units—1.4 of which get used while the remaining 1.6 units contribute to pollution. This is in contrast with the $SFP_1^{IA,DA}$ predictor which brings in around 5 units on average while missing 0.32 units (which incur additional misses) and not using about 0.35 units before eviction. i.e. 1.32 misses fetch 5.32 units. This translates into each miss prefetching 3.03 ($= (5.32-1.32)/1.32$) units—2.78 of which get used while only 0.35 units contribute to pollution.

5. Implementation

Using the spatial footprint predictor with a sectored cache might reduce the fetch bandwidth. However, it will hurt the miss ratio because of underutilization of the cache. So we use a cache tagged at cache line granularity. This allows the lines within the same sector in the virtual memory to reside in the cache independent of each other. The implementation in this section (Figure 3) uses a larger tag array (because of the smaller lines) and infinite tables for SHT and AST. In Section 6, we study the effect of using smaller tag arrays and reasonable table sizes.

A default predictor is needed for the cases when the SFP predictor fails to make a prediction because no spatial history is available. On a cache miss in an inactive sector, the SFP predictor or the default predictor is invoked depending on whether an entry was found in the SHT. Lines missed due to mispredictions result in cache misses in an active sector. The recovery mechanism, though simple, is fairly different for the two predictors. This is because the default predictor is not a "real" footprint predictor.

The AST has a flag field to indicate which of the two predictors was used when the sector was activated. On a cache miss in an active sector, this predictor flag is used to determine which predictor was used earlier and invoke the corresponding recovery mechanism.

5.1. Spatial Footprint Predictor

The simple definitions of active and inactive sectors from the previous section need to be redefined. This is because the lines within a sector are treated independently. As before, all sectors in virtual memory are initially inactive. A cache miss in an inactive sector causes that sector to become active. A sector is deactivated again if the AST entry is evicted because of conflict. A sector is also deactivated on a miss to a line which is marked used in the footprint. This happens when a line in an active sector was fetched, evicted and needs to be fetched again.

The miss rate is fairly sensitive to the sector deactivation policy because it determines the duration during which footprints are recorded. If the deactivation policy is too lazy, the cache incurs an extra miss for every reference in the active sectors and causes multiple footprints to get merged into one. If the deactivation policy is too aggressive, the footprints get fragmented into multiple footprints resulting in poor prediction accuracy. Our deactivation policy was chosen because of its simplicity and better performance on reasonable SHT table sizes. A more aggressive deactivation policy, which in addition to the above conditions also deactivates a sector when the nominating line is evicted, results in better performance (about 5% better on average) when using an infinite SHT but yields worse performance (about 5% worse on average) for reasonable size tables.

When the SFP predictor is invoked, the corresponding entry in the SHT is used to predict the footprint and fetch the corresponding lines. When the SHT recovery mechanism is invoked, only the line that caused the miss is fetched. Since the number of lines mispredicted by the SFP is fairly low, just bringing in one line is good enough.

5.2. Default Predictor

A default predictor is needed for the case when the spatial footprint predictor fails to make a prediction for lack of spatial footprint history. We use a simple global predictor as our default predictor. The predictor observes the spatial footprints generated by the running application and, depending on the locality observed, predicts the number of adjacent lines that should fetched on a miss. The default predictor essentially choses between "line sizes" of 32, 64 and 128 by fetching 4, 8 or 16 aligned block of lines. It maintains a cost meter for each of the line sizes. The cost meters are initialized to zero and updated each time a recorded spatial footprint is moved into the SHT. For each of the three meters, it uses the footprint to com-

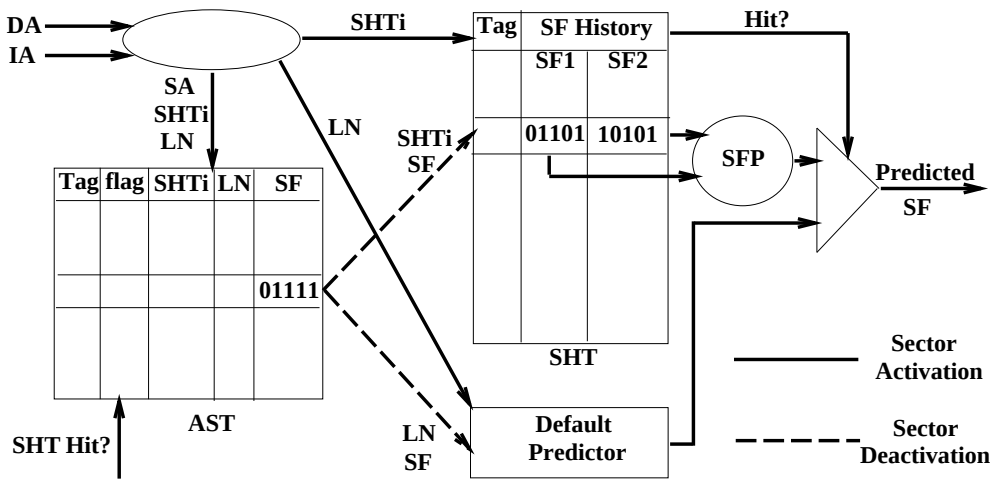


Figure 3. On sector activation, an AST entry is allocated and the predicted spatial footprint is used to fetch lines. On sector deactivation, the recorded footprint is used to update the two predictors.

pute the number of lines it would have missed and the number of extra lines it would have fetched had the corresponding line size been chosen. These two values are combined using a cost function and the meters updated using each of the corresponding values. We used the cost function $Cost(missed, extra) = 2 * missed + extra$.

When the default predictor is invoked, the number of lines associated with the meter displaying the lowest cost is chosen. An aligned group of contiguous lines consisting of the number of lines chosen is fetched. Exactly the same thing is done when the recovery mechanism for the default predictor is invoked. This effectively mimics the behavior of traditional caches with the same line size as the one picked by the default predictor. Due to the coarse granularity of the default predictor, the number of lines chosen by it rarely changes during the execution of an application.

5.3. Experimental Evaluation

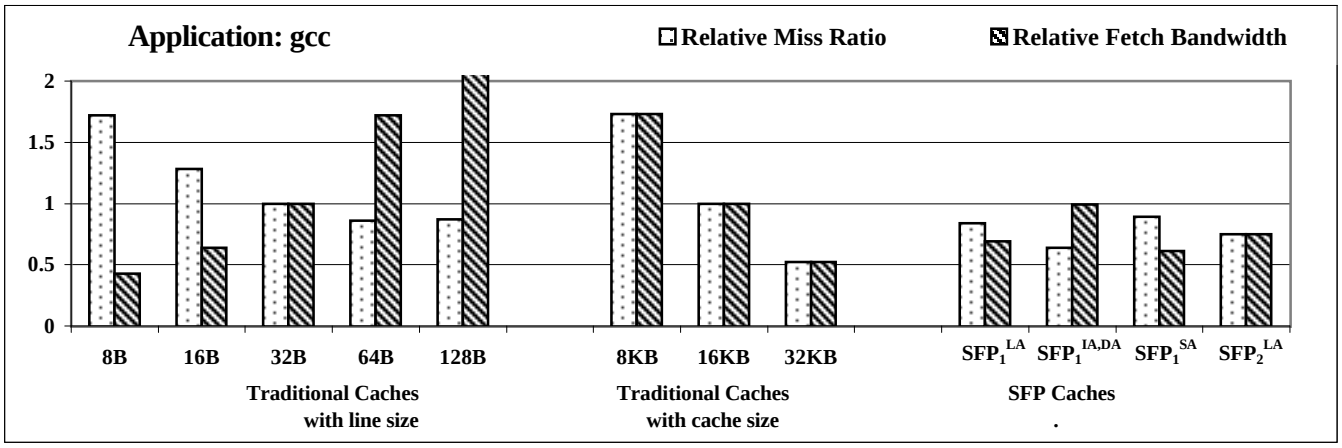
We use a cache simulator to look at four of the predictors (SFP_1^{LA}, $SFP_1^{IA,DA}$, SFP_1^{SA} and SFP_2^{LA}). All predictors use 32 bits to index the SHT. So we use 20 bits of data address together with 12 bits of the instruction address in the implementation of $SFP_1^{IA,DA}$. Since some instructions access a lot of different data locations, the data address bits are used as the lower order bits in the SHT index. This reduces the conflict significantly when using a reasonable sized SHT (Section 6).

Figure 4 presents the relative miss ratio and fetch bandwidth of traditional caches with varying line and cache sizes along with the corresponding numbers for caches using the various SFP predictors. The first set of numbers is for traditional 16 KBytes caches with line size varying from 8 to 128 (the sector size in SFP caches). The second set of numbers correspond to traditional caches using 32 byte line but with cache sizes of 8 KBytes, 16 KBytes and 32 KBytes. The last set of numbers correspond to 16 KByte SFP caches with 8 byte lines and 16 lines per sector.

The first thing to note is that the best line size is application dependent. Comparing the SFP caches with caches of varying line size shows that most of the SFP caches achieve close to (and often better than) the best miss rate for each application. The fetch bandwidth in each case is also fairly close to the lowest value which occurs when using 8 byte lines. In traditional caches, the best miss ratio and the lowest fetch bandwidth rarely occurs at the same line size. However, the SFP mechanism successfully tracks both.

Comparing the SFP caches with the 32 KB cache shows that the miss rates are comparable and the fetch bandwidth is often better for the SFP caches. In a few cases (e.g. specweb), the best SFP cache has a lower miss rate too.

The reduction in miss rates for the SFPs come from two sources—exploiting spatial locality at the sector granularity and lower pollution. The miss ratios for the various SFP caches relative to each other match with the results in Section 4. $SFP_1^{IA,DA}$ and SFP_2^{LA} have similar miss ratios followed by SFP_1^{LA}. SFP_1^{SA} is the least effective. The fetch bandwidth has three components—the useful data brought in, the pollution due to the default predictor and the pollution due to the SFP predictor. The contribution of the useful data fetched to the bandwidth should be approximately the same for all the predictors. The difference between the fetch bandwidth for SFP_1^{LA} and $SFP_1^{IA,DA}$ is due to the greater reliance of $SFP_1^{IA,DA}$ on the default predictor while the difference between the fetch bandwidth for SFP_1^{LA} and SFP_2^{LA} is due to larger pollution by the SFP_2^{LA} predictor.



Relative Miss Ratio	16KB Cache with Line Size					32B Line with Cache Size			SFP_1^{LA}	$SFP_1^{IA,DA}$	SFP_1^{SA}	SFP_2^{LA}
	8 B	16 B	32 B	64 B	128 B	8 KB	16 KB	32 KB				
draw	1.37	1.14	1.00	0.86	0.74	1.03	1.00	0.93	0.82	0.81	0.86	0.79
gcc	1.72	1.28	1.00	0.86	0.87	1.73	1.00	0.52	0.84	0.64	0.89	0.75
go	0.85	0.85	1.00	1.48	2.32	1.94	1.00	0.62	0.75	0.76	0.77	0.74
pc_db	1.55	1.20	1.00	0.91	0.90	1.50	1.00	0.64	0.71	0.64	0.77	0.67
pres1	1.64	1.19	1.00	0.94	0.90	1.44	1.00	0.67	0.68	0.66	0.70	0.66
pres2	2.26	1.46	1.00	0.77	0.68	1.33	1.00	0.79	0.75	0.72	0.79	0.73
specweb	1.42	1.30	1.00	0.87	0.81	1.12	1.00	0.92	0.91	0.75	0.94	0.84
sprdsht	1.39	1.14	1.00	1.00	1.14	1.75	1.00	0.58	0.71	0.67	0.75	0.67
tpcc	1.56	1.24	1.00	0.91	0.87	1.41	1.00	0.62	0.77	0.70	0.85	0.69
tpcc_long	2.19	1.46	1.00	0.80	0.69	1.22	1.00	0.81	0.76	0.72	0.77	0.70
wdproc1	1.85	1.27	1.00	0.88	0.81	1.48	1.00	0.64	0.63	0.59	0.70	0.60
wdproc2	1.50	1.17	1.00	1.00	1.09	1.87	1.00	0.53	0.72	0.69	0.75	0.68
Mean	1.61	1.23	1.00	0.94	0.98	1.48	1.00	0.69	0.75	0.70	0.80	0.71

Relative Fetch Bandwidth	16KB Cache with Line Size					32B Line with Cache Size			SFP_1^{LA}	$SFP_1^{IA,DA}$	SFP_1^{SA}	SFP_2^{LA}
	8 B	16 B	32 B	64 B	128 B	8 KB	16 KB	32 KB				
draw	0.34	0.57	1.00	1.73	2.96	1.04	1.00	0.93	0.39	0.39	0.39	0.41
gcc	0.43	0.64	1.00	1.72	3.47	1.73	1.00	0.52	0.69	0.99	0.61	0.75
go	0.21	0.42	1.00	2.96	9.30	1.94	1.00	0.62	0.38	0.56	0.33	0.46
pc_db	0.40	0.61	1.00	1.82	3.58	1.50	1.00	0.64	0.55	0.58	0.52	0.59
pres1	0.41	0.60	1.00	1.88	3.58	1.44	1.00	0.66	0.50	0.52	0.49	0.52
pres2	0.57	0.73	1.00	1.53	2.71	1.33	1.00	0.79	0.72	0.75	0.70	0.74
specweb	0.36	0.65	1.00	1.75	3.24	1.12	1.00	0.92	0.43	0.40	0.44	0.47
sprdsht	0.35	0.57	1.00	2.00	4.57	1.75	1.00	0.59	0.45	0.47	0.44	0.49
tpcc	0.39	0.62	1.00	1.82	3.47	1.40	1.00	0.62	0.50	0.50	0.50	0.53
tpcc_long	0.55	0.73	1.00	1.59	2.75	1.22	1.00	0.82	0.66	0.65	0.66	0.68
wdproc1	0.47	0.63	1.00	1.76	3.26	1.48	1.00	0.64	0.66	0.73	0.60	0.68
wdproc2	0.38	0.58	1.00	2.00	4.35	1.86	1.00	0.53	0.51	0.55	0.49	0.54
Mean	0.40	0.61	1.00	1.88	3.94	1.49	1.00	0.69	0.54	0.59	0.52	0.57

Figure 4. Miss ratio and Fetch Bandwidth for (i) 16 KB caches with line sizes of 8, 16, 32, 64 and 128 (ii) 8 KB, 16 KB and 32 KB cache with 32 byte line and (iii) various 16 KB caches using the SFP mechanism. All numbers are normalized to the 16 KB cache with 32 byte line. The bar graph presents just the gcc results.

Note: *The tag array in Figure 5 is direct mapped and not 4-way set associative.* 9

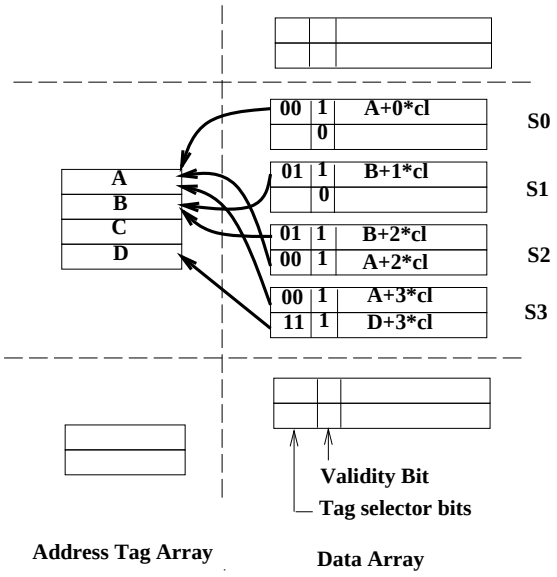


Figure 5. A Decoupled sectored cache where the tag array is four-way and the data array is two-way associative. There are 4 lines to a sector while the size of a cache line is cl.

6. Practical Considerations

In this Section, we study the effect of using reasonable sized tag array and tables. We do this in two stages and examine the performance degradation at each stage.

6.1. Limiting the SHT Size

We use a 1024 entry, four-way set associative SHT in our implementation. The SHT is looked up on every cache access so that the spatial footprint is available on a miss. No extra cycles are added to the miss penalty.

Once an SHT entry is used, it will no longer be used until it is updated with a newly recorded footprint. When the predictor is using a history of one, the SHT entry does not have anything useful once it is used. So it is invalidated. If this is not done, the useless entry will be the last one to be evicted because it is the least recently used entry.

6.2. Using Smaller Tag Array

Sectored caches have been used to deal with the large tag array required by caches with small lines. In a sectored cache, a single address tag is associated with a sector consisting of several cache lines, while validity tags are associated with each of the cache lines. However, this results in bad performance due to poor utilization of the cache resources. Seznec [15] proposed decoupled sectored caches as a way to reconcile low tag implementation cost with low miss ratio. In a decoupled sectored cache, instead of the static association between the tag and data, the association is determined dynamically at allocation time. The address tag location associated with a cache line is dynamically chosen among several possible tag locations. Figure 5 shows a decoupled sectored cache where the tag array is four-way set associative and the data array is two-way set associative. Each sector has 4 lines. This means that a tag *A* can be associated with up to 4 lines—one in each of *S0*, *S1*, *S2* and *S3*. Similarly, each line *S0* can have its tag in one of four location—*A*, *B*, *C* and *D*). This mapping is dynamically maintained by using tag selector bits for every line to specify the location of the tag. It also maintains the validity information on a per line basis.

On a cache reference, the tag verification is slightly more complicated than in traditional caches and, therefore, takes longer. However, since the data array access is not affected, a way-predictor [11] can be employed to optimistically use the data without waiting for the tag matching to complete [15]. This technique is already used by some processors to achieve low cache hit time in set-associative caches. So, using a decoupled sectored cache should have little impact on the cache access latency.

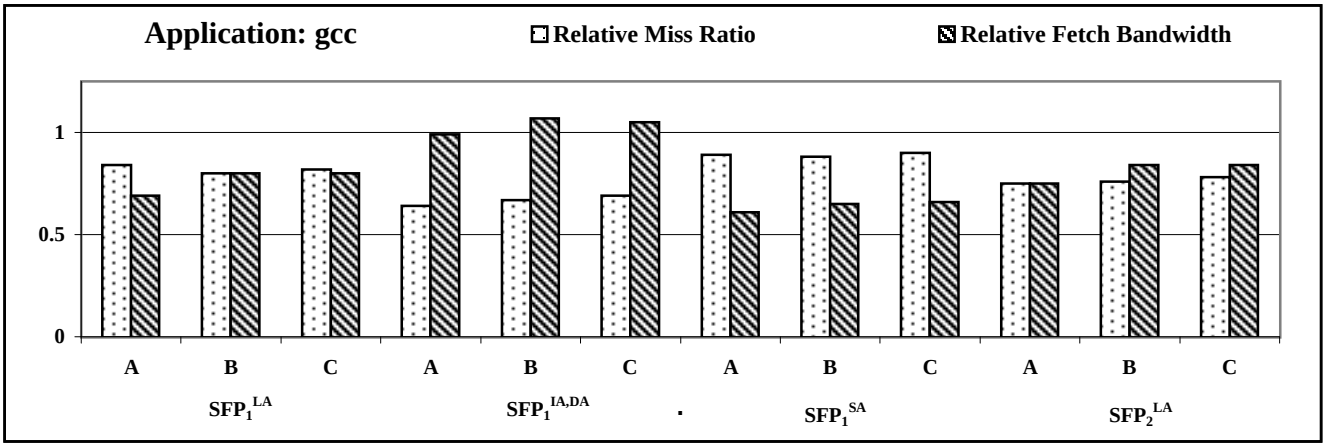
Decoupled sectored caches were originally proposed to reduce the size of tag array for L2 cache controllers. For them to be effective, several lines per sector have to be present in the cache so that multiple lines can share the same tag. This property holds in our case because a sector has, on average, more than 4 lines in the cache. We use a decoupled sectored cache with four-way set-associative data and tag array. The data array has 2048 entries while the tag array has 512 entries so that the ratio of data to tag entries is 4:1. This is the same number of tags as in the base configuration.

6.3. Reducing the AST Size

Since the decoupled cache is tagged at the sector granularity, we can combine the AST with the tag array for the cache. There are two benefits to this approach. First, separate tags are not required for the AST. Second, since there is a tag associated with every sector that has a line in the cache, there is always a free AST entry available when needed.

6.4. Experimental Evaluation

Figure 6 presents the miss ratios and fetch bandwidth when using reasonable size tables. Column A shows the numbers obtained when using infinite size tables (Section 5). Column B presents the numbers obtained when using 1024 entry SHT. Column C presents the numbers when using a 512 entry AST integrated with a decoupled sectored cache in addition to a 1024 entry SHT.



Relative Miss Ratio	SFP_1^{LA}			$SFP_1^{IA,DA}$			SFP_1^{SA}			SFP_2^{LA}		
	A	B	C	A	B	C	A	B	C	A	B	C
draw	0.82	0.90	0.92	0.81	0.89	0.91	0.86	0.94	0.97	0.79	0.91	0.93
gcc	0.84	0.80	0.82	0.64	0.67	0.69	0.89	0.88	0.90	0.75	0.76	0.78
go	0.75	0.79	1.05	0.76	0.80	1.05	0.77	0.77	1.06	0.74	0.79	1.04
pc_db	0.71	0.74	0.78	0.64	0.70	0.74	0.77	0.76	0.81	0.67	0.74	0.77
pres1	0.68	0.72	0.82	0.66	0.72	0.82	0.70	0.72	0.82	0.66	0.72	0.82
pres2	0.75	0.73	0.74	0.72	0.71	0.71	0.79	0.79	0.80	0.73	0.72	0.73
specweb	0.91	0.96	0.98	0.75	0.93	0.94	0.94	0.97	0.98	0.84	0.97	0.98
sprdsht	0.71	0.76	0.81	0.67	0.76	0.80	0.75	0.76	0.81	0.67	0.76	0.80
tpcc	0.77	0.83	0.86	0.70	0.81	0.83	0.85	0.85	0.89	0.69	0.82	0.83
tpcc_long	0.76	0.91	0.93	0.72	0.91	0.92	0.77	0.92	0.93	0.70	0.91	0.92
wdproc1	0.63	0.66	0.67	0.59	0.65	0.66	0.70	0.69	0.71	0.60	0.65	0.66
wdproc2	0.72	0.75	0.80	0.69	0.75	0.80	0.75	0.75	0.80	0.68	0.75	0.80
Mean	0.75	0.80	0.85	0.70	0.78	0.82	0.80	0.82	0.87	0.71	0.79	0.84

Relative Fetch Bandwidth	SFP_1^{LA}			$SFP_1^{IA,DA}$			SFP_1^{SA}			SFP_2^{LA}		
	A	B	C	A	B	C	A	B	C	A	B	C
draw	0.39	0.56	0.57	0.39	0.57	0.58	0.39	0.55	0.57	0.41	0.63	0.65
gcc	0.69	0.80	0.80	0.99	1.07	1.05	0.61	0.65	0.66	0.75	0.84	0.84
go	0.38	0.52	0.63	0.56	0.70	0.89	0.33	0.33	0.45	0.46	0.54	0.66
pc_db	0.55	0.66	0.68	0.58	0.72	0.73	0.52	0.58	0.61	0.59	0.68	0.70
pres1	0.50	0.56	0.60	0.52	0.60	0.63	0.49	0.53	0.57	0.52	0.57	0.61
pres2	0.72	0.90	0.91	0.75	1.02	1.03	0.70	0.77	0.79	0.74	0.94	0.95
specweb	0.43	0.76	0.77	0.40	0.80	0.81	0.44	0.71	0.72	0.47	0.79	0.80
sprdsht	0.45	0.53	0.57	0.47	0.59	0.64	0.44	0.47	0.52	0.49	0.55	0.59
tpcc	0.50	0.57	0.60	0.50	0.65	0.68	0.50	0.52	0.56	0.53	0.60	0.63
tpcc_long	0.66	0.80	0.81	0.65	0.84	0.85	0.66	0.76	0.77	0.68	0.81	0.82
wdproc1	0.66	0.86	0.85	0.73	0.96	0.97	0.60	0.75	0.76	0.68	0.87	0.87
wdproc2	0.51	0.59	0.57	0.55	0.66	0.62	0.49	0.53	0.53	0.54	0.61	0.59
Mean	0.54	0.68	0.70	0.59	0.76	0.79	0.52	0.60	0.63	0.57	0.70	0.73

Figure 6. Miss ratio and Fetch Bandwidth for SFP caches with (A) Infinite tables and per line tags (B) 1024 entry SHT and per line tags (C) 1024 entry SHT and decoupled sectored cache integrated with the AST. All numbers are normalized to the base configuration which uses a 16 KB cache with 32 byte lines. The bar graph presents just the gcc results.

Comparing columns A and B, we see that the biggest change in miss ratios is for $SFP_1^{IA,DA}$ and SFP_2^{LA}. For $SFP_1^{IA,DA}$, this is because it uses the most distinct entries in the SHT (Section 4) and thereby suffers a higher miss ratio in the SHT. In the case of SFP_2^{LA}, the use of a small SHT results in the loss of most of the benefit of using additional history. So it performs like SFP_1^{LA} cache. The corresponding behavior can be observed in the fetch bandwidth.

Comparing columns B and C, we see that the change in the miss ratios corresponds to the amount of spatial locality exhibited by the application. The bulk of the difference can be attributed to the fewer tags used by the decoupled sectored cache. On one hand, in applications like *go* which have very little locality, the fewer tags available result in about 20% of the data array not being used. On the other hand, in applications like *pres2* and *wdproc1* which exhibit a lot of spatial locality, the change in the miss ratio is marginal. Also, the change in the miss ratio is fairly independent of the predictor used. As before, the corresponding behavior can be observed in the fetch bandwidth.

Overall, the $SFP_1^{IA,DA}$ achieves the best miss ratios. However, practical considerations might make SFP_1^{LA} the better choice. First, the data address is easily available in the data caches while additional datapath might be needed to make the instruction address available. Second, they can be implemented at a lower cost.

6.5. Cost of the System

The decoupled sectored cache can be designed without a significant increase in L1 access latency (Section 6.2). Since the SFP mechanism does not add any operations in the critical path of an L1 access, the entire implementation should have little impact on the L1 access latency. In addition, our mechanism does not affect the miss penalty. So the reduction in the miss rates and bandwidth from the Section 6.4 should translate into better cache performance.

Most latency tolerance mechanisms do so at the expense of increased bandwidth requirements [4]. However, the SFP mechanism achieves a lower miss rate and reduces the bandwidth requirement at the same time. The L1-L2 bus protocol will need only minor modifications to benefit from the lower bandwidth. Current bus protocols support critical word first where the referenced word is transferred before the remaining words in the cache line. They can be modified so that the critical word is transferred first followed by the specific words within the sector requested by the L1 cache. Since all the words in a sector lie within the same cache line in the L2 cache, this should be a simple change.

The amount additional space needed to implement the predictor is about 8.75 KBytes. 512 bytes (2 bits per line) are need to maintain the dynamic mapping between the tags and data in the decoupled sectored cache. 5 KBytes (1024 entries * (3 bytes for tags + 2 bytes for the footprint)) are used for the SHT. The remaining 3.25 KBytes (512 entries * (4 bytes for SHT index + 2 bytes for the footprint + 4 bits for the nominating line number)) are used by the AST. Not all of the 2 KBytes needed for the SHT index in the AST has to be stored explicitly because the sector address can be generated from its index and the tag in the AST. Two points are worth noting here. First, none of the additional memory lies in the critical path of the cache operation. Second, the optimizations described in Section 7 should reduce the space requirements significantly.

7. Future Work

The best SFP implementation using reasonable resources resulted in about an 18% reduction in miss ratio on average. Although, this is a significant improvement, the biggest reduction in miss ratios using infinite tables (Section 5) is around 35%. Achieving this will require better utilization of the limited SHT entries.

Having an entire class of predictors which differ in coverage and accuracy allows the design of hybrid predictors. When the more accurate predictor fails to make a prediction, a less accurate predictor can be used instead. Clever design of SHT can allow multiple predictors to share the SHT. For instance, this can be done fairly easily in the case of SFP_1^{LA} and SFP_1^{SA}.

One potential problem with SFP prediction is that regular access patterns while accessing large amount of data can result in thrashing in the SHT causing the default predictor to be used extensively. This can be avoided by using the SFP in combination with a coarse-grain predictor. The locality detection mechanism (AST) can also be used in the implementation of the coarse-grain predictor.

The memory overhead of the SFP mechanism can be reduced further by compressing the footprints in the SHT. Preliminary results show that only a small number of distinct footprints are used. In addition, the size of the tag array for the SHT can be reduced by using partial tags. In some case, like the SFP_1^{SA}, it might be entirely eliminated by merging the SHT with the L2 tag array.

Using decoupled-sectored caches allows cache designers to pick different associativities for the data and tag arrays. Also, the LRU replacement policy might not be the best policy for the tag array because evicting a tag entry usually results in multiple lines getting evicted from the data array.

Finally, the SFP mechanism might be applicable to other layers of the memory hierarchy including L2.

8. Conclusions

Today's caches benefit from some of the spatial locality exhibited by applications. However, they exploit only a small fraction of the spatial locality available. Run-time mechanisms that observe and adapt to the application behavior are more effective alternatives.

Spatial Footprint Predictors can accurately predict the spatial locality at a very fine grain. Both the instruction address and the data address can be used for the prediction. However, the best accuracy is achieved by using a combination of both. We describe a scheme that translates the predictor accuracy into lower miss rates and bandwidth usage without significantly affecting the cache access latencies.

Cache-level simulations on several real-world applications show that the mechanism can effectively exploit spatial locality available to a long cache line with minimum pollution. The miss rates for first-level caches is reduced by 35% on average. For all applications, the miss rate is often better than the best miss rate obtained by using a fixed line size. The miss rates are also comparable to those rates achieved by doubling the cache size. Although a practical implementation of the mechanism loses some of the gains, the improvements in miss rates are still significant (about 18% on average).

It should be possible to refine the schemes described in this paper to achieve greater benefits with smaller cost (Section 7). Also, the Spatial Footprint Predictor might be useful in the other layers of the memory hierarchy as well.

Acknowledgments

We would like to thank Wen-Hann Wang, Nick Wade, Doug Clark, JP Singh, Stefanos Damianakis, and the anonymous referees for their helpful comments. We would also like to thank other members of the Microcomputer Research Lab, Intel, Oregon for their support.

References

[1] 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 International Symposium on Microarchitecture*, pages 139–152, Austin, Texas, December 1–3, 1993.

[2] J.-L. Baer and T.-F. Chen. An effective on-chip preloading chip scheme to reduce data access penalty. In *Proceedings of Supercomputing*, 1991.

[3] K. Bolland and A. Dollas. Predicting and precluding problems with memory latency. In *IEEE Micro*, pages 59–66, August, 1994.

[4] D. Burger, J. R. Goodman, and A. Kägi. Memory bandwidth limitations of future microprocessors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 78–89, Philadelphia, PA, May 22–24, 1996.

[5] W. Y. Chen, A. A. Mahlke, P. P. Chang, and W. W. Hwu. Data access microarchitectures for superscalar processors with compiler-assisted data prefetching. In *Proceedings of Microcomputing 24*, 1991.

[6] A. González, C. Aliagas, and M. Valero. A data cache with multiple caching strategies tuned to different types of locality. In *Proceedings of International Conference on Supercomputing*, pages 338–347, Barcelona, Spain, July, 1995.

[7] T. Johnson, M. Merten, and W. mei W. Hwu. Run-time spatial locality detection and optimization. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 57–64, Research Triangle Park, NC, 1997.

[8] T. L. Johnson and W. mei W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. In *Proceedings of the 24rd Annual International Symposium on Computer Architecture*, pages 315–326, Denver, Colorado, 1997.

[9] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 364–373, Seattle, Washington, May 28–31, 1990.

[10] M. H. Lipasti, W. J. Schmidt, S. R. Kunkel, and R. R. Roediger. SPAID: Software prefetching in pointer- and call-intensive environments. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 231–236, Ann Arbor, Michigan, 1995.

[11] L. Liu. Cache designs with partial address matching. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 128–136, San Jose, California, November 30–December 2, 1994.

[12] C.-K. Luk and T. C. Mowry. Compiler-based prefetching for recursive data structures. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 222–233, Cambridge, MA, October 1–5, 1996.

[13] T. C. Mowry, M. S. Lam, and A. Gupta. Design and evaluation of a compiler algorithm for prefetching. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 62–73, Boston, Massachusetts, October 12–15, 1992.

[14] S. Przybylski. The performance impact of block sizes and fetch strategies. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 160–169, Seattle, Washington, May 28–31, 1990.

[15] A. Seznec. Decoupled sectored caches: Conciliating low tag implementation cost and low miss ratio. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 384–393, Chicago, IL, April, 1994.

[16] A. J. Smith. Cache memories. In *Computing Surveys*, pages 473–530, vol. 14, no. 3, 1982.

[17] A. J. Smith. Line (block) size choice for cpu cache memories. In *IEEE Transactions on Computers*, pages 1063–1075, vol. C-36, 1987.

[18] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun. A modified approach to data cache management. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 93–103, Ann Arbor, Michigan, 1995.