# A Prefetch Taxonomy

Viji Srinivasan, Edward S Davidson, Gary S Tyson
e-mail: sviji@eecs.umich.edu, davidson@eecs.umich.edu, tyson@eecs.umich.edu

**Abstract**

*The difference in processor and main memory cycle time has necessitated the use of aggressive prefetching techniques to reduce or hide main memory access latency. However, prefetching can significantly increase memory bandwidth and unsuccessful prefetches may even pollute the primary cache. Although the current metrics,* coverage *and* accuracy*, do provide an impression of the overall quality of a prefetching algorithm, they are simplistic and do not unambiguously and precisely explain the performance of a prefetching algorithm. In this paper, we introduce a new and complete taxonomy for classifying prefetches, accounting for the difference in traffic and misses generated by each prefetch. Our classification of individual prefetches is very useful in identifying the specific strengths and weaknesses of an existing prefetch implementation and we demonstrate it by applying our taxonomy to two implementable prefetching techniques. Based on the histogram of prefetches by category we developed a static filter to reduce/eliminate polluting prefetches of any given prefetching technique.*

## 1 Introduction

As processor speeds have increased over the past few decades, the gap between memory access latency and the processor cycle time has steadily increased. Use of multi-level cache hierarchies has been a traditional solution to bridging this gap. However, with superscalar processors now executing multiple instructions per cycle, even greater demand is placed on the cache system. Prefetching is a technique that reduces memory access latency by fetching lines into cache before a demand reference. In theory, we could prefetch each missing cache line so that it arrives in cache just before its next reference. However, in reality aggressive prefetching techniques are speculative; they must predict the prefetch addresses. Hence the following issues arise:

- Some addresses are not accurately predicted limiting the effectiveness of speculative prefetching techniques.
- Even when the prediction is accurate and the prefetch is issued early enough to cover the nominal latency, the full memory latency may not be hidden due to additional delays caused by limited available memory bandwidth.

- Prefetched lines, whether correctly predicted or not, may be issued too early and pollute the cache by replacing more desirable lines.

Effective prefetching depends on achieving good miss coverage with su cient accuracy to avoid saturating the memory bus with useless prefetches and polluting the cache. raditionally and have been used as metrics to evaluate the effectiveness of Prefetch is the fraction of misses that are eliminated by the prefet Prefetch estimates the quality of the prefetching algorithm as a fracti useful. o measure coverage and accuracy, prefetches are broadly classi ed into two categories : and . ood prefetches are referenced by the application befo bad prefetches are replaced before they are referenced. If a conventional cache has misses without using any prefetching technique and has good prefetches and $B$ bad prefetches with some prefetching algorithm, we can measure prefetch coverage and accuracy as follows:

$$\text{Coverage} = \quad /\text{M}, \tag{1}$$

$$\text{Accuracy} = \quad /(\ ). \quad + \text{B} \tag{2}$$

Although and algorithm, they are simplistic and do not unambiguously and precisely explain the performance of a prefetching algorithm. Using the simple example in Figure 1 we show that maximizing good prefetches, or even getting reasonable coverage with high accuracy is not the true objective of a prefetching technique.

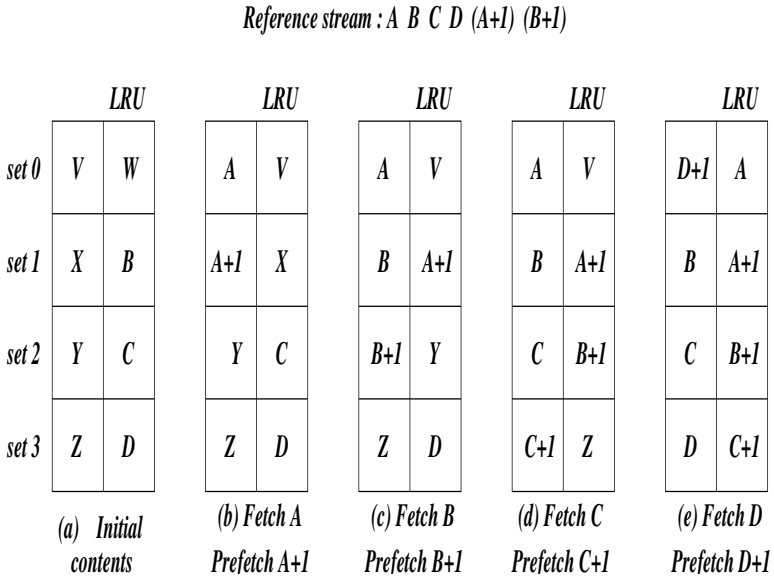


Figure 1: Cache access outcomes with NSP

he cache used in this example has sets with 2-way associativity and uses an policy. he initial contents of the cache are given in 1(a). Figure he reference stream consists of the unique line addresses, , ,1), and ($B$ + 1( , which map to sets 1, 2,1, , and 2 respectively. he total misses for this cache without any prefetching technique is (missing lines : + ,1 and ( $B$(+ 1) . ) consider the cache access outcomes using the SP) technique 1 . Figure 1(b) shows the cache contents after the miss to line is resolved. In SP the next sequential line ( 1 is ) prefetched along with . ine ( 1 replaces the $B$ of set 1. U line Figure 1(c,d,e) show the contents of the cache after the references to $B$, and , respectively, are resolved. f the prefetches, 2 are herefore, the coverage is . SP does eliminate 2 of the misses, it introduces new misses. herefore the total misses with SP is increased from N to . Furthermore, SP increases the total tra c from to cache lines. his example illust a prefetch technique, even with . coverage and ratio and more than double the memory tra c.

Prefetch side-effects on the misses and tra c are thus critical in designing an e cient prefetching technique. Coverage and accuracy, as seen above, do not provide a direct indication of performance, and can be very misleading metrics. Furthermore, even direct measurements of the total change in misses and total tra c do not provide much insight on the underlying causes of that net change and how the performance of a prefetching technique may be improved. here is a need for a more re ned classi cation of individual prefetches that gives more insights into how a given prefetching technique wor s.

In this paper we introduce a new, accurate and complete taxonomy, called the Prefetch ra c and Miss axonomy), for lassifying prefetches and precisely accounting for the difference in tra c and misses generated by each prefetch. M quantifies a prefetch in terms of the difference in misses and the net tra c increase that are due to the outcome of both the prefetch and the line replaced to accommodate the prefetch. P M s classi cation of prefetches has proven to be very useful in identifying the speci c strengths and wea nesses of an existing prefetch implementation. Furthermore, the taxonomy helps answer the following questions:

- How can a prefetch technique eliminate those prefetches that pollute the cache
- How can a prefetch technique reduce useless prefetches and thus relieve the pressure on the available memory bandwidth
- hat insights does the prefetch classi cation provide for improving the performance of the prefetching technique

he P is based on simultaneous simulation of a conventional cache with no prefetching and another cache that implements some prefetching technique. Using this model we analy ze

each prefetch and assign it a -1, , 1 or net change in misses and a cost of 1 or , 2 lines of additional tra c that it causes relative to the conventional cache. he completeness of P M assures that the extra tra c and the difference in misses based on P M plus the tra c a misses of the conventional cache always exactly matches the total tra c and misses of the cache with the prefetch technique. P M therefore provides new and precise metrics for evaluating a given prefetching technique. Furthermore, by separately tabulating the strengths and wea nesses of a prefetch technique, P M provides insights for improving it.

he rest of this paper is organi zed as follows. Section 2 describes the prefetch tra c miss taxonomy. Section P uses to measure the effectiveness of two common prefetching algorithms, and M insights derived from the taxonomy to improve the performance of these prefetching techniques.

e conclude in section .

## r c tc ind r ono

Prefetch tra c and Miss P ) categorizes each prefetch according to a 2 tors, and M whether it remains in the cache until its next reference or is replaced before then, as well as the outcome of the next reference to the line evicted to accommodate this prefetch.

Classifying prefetches using P M fundamentally requires the simultaneous simulation of two identically con gured caches, a conventional cache without any prefetching, and with some prefetching technique of interest, goal is to study the effect of prefetching . As our on the cache performance, we prefetch directly into the cache and do not consider using a separate prefetch buffer 1 . here is no restriction on the si ze, associativity, or line si ze of these two caches, except that they are the same.

In this paper, we refer to lines in each cache as A line that is prefetched into the is called a referenced; after being referenced it is called a either cache are called 1-bit tag is maintained in the . A regular and prefetched lines.

As our focus is on the number of cache lines fetched from the next level of memory, we assume that the caches are write-through and no-write-allocate, and the tra c to the next level of memory due to stores is ignored. U replacement policy is used. Each set of the cache is implemented in the simulation as an stac . All new lines entering the cache via a miss or a prefetch are pushed

onto the top of the U stac (the most recently used position); lines leave the set from the bottom of the stac . If there are lines in a set, the top of the U stac is refe the bottom of the stac is referred to as position prefetch is squashed if the line is already . A present in the cache.

he references to lines in the can be broadly associated with a prefetch (i.e., references to a prefetched line or to a line that was replaced to accommodate a prefetch), and those that are not. P M accounts for all extra misses and tra associated with both categories of references.

Suppose line is prefetched into and re possible next events for line : a hit, i.e., it remains in the replacement, i.e., is replaced before its next reference. In the line is either a hit or a miss.

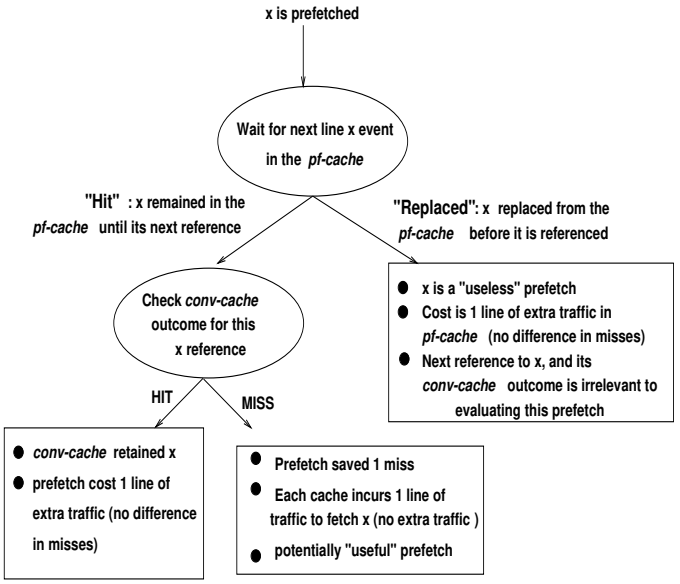


Figure 2: Events in and

Figure 2 shows all the possible events for line in the 1 summari zes the extra tra c and misses associated with each combination.

hen line is a hit in the , the

- 1 cache: line of extra tra is a hit in both prefetch .

- : ine is a useful prefetch. However, the usefulness of this prefetch depends on what happens next to the replaced line . Note that when the incur we represent it as -1 extra misses.

| *vents for x in* | | | |
|---|---|---|---|
| *pf* | *conu cache* | *cache* | **ra c** |
| | | | |
| | | | |
| re | care | ace | |

**tra ss s**

able Cost of events in

hen is replaced f before being referenced, present or absent in the t care . Hence that only we refer to th possible combination when is replaced from

- : he next reference to in the prefetch or by a demand miss. A second prefetch of (after the replacement of the rst prefetched line, but before the next reference to ) is simply classi ed independently of the rst prefetch. herefore, the only cost associated with the replaced 1 cache line prefetch is of additional tra c in the .

Similarly, let us consider all the possible outcomes of line (the line that the prefetch replaced in the ). In the , line is either a miss or is p In the e., it remains in the cache until its next reference, i. is replaced before its next reference. Figure shows all the possible outcomes for line in the and the combination.

For ease of explanation we discuss the possible outcomes of in the possible outcomes of in the . hen line possibilities arise:

- : ine is a mis replacement of in .
- : If being replaced to accommodate ), the outcome of this prefetch falls into one of the cases in 1. So we must not attribute the cost of that prefetch to this prefetch.

hen is replaced from before bei

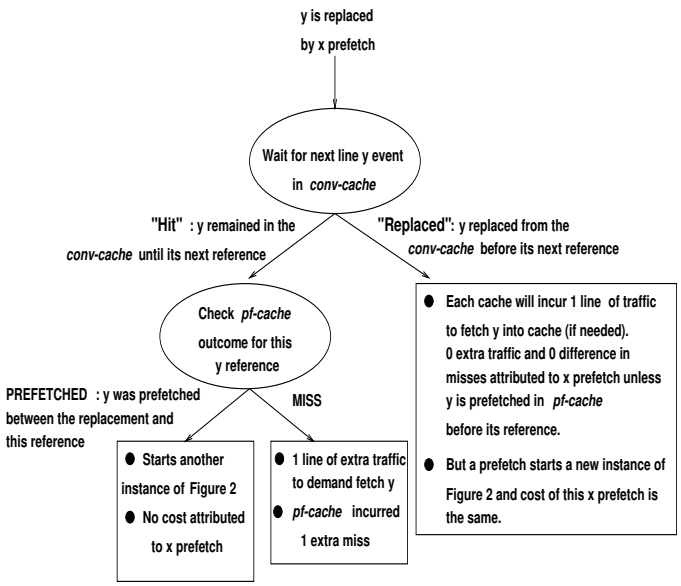


Figure Events in and

| *vents for y in* | | | **tra** |
|---|---|---|---|
| *pf* | *conche* | *cache* **ra** **c** | **ss** **s** |
| | | | |
| c re | e | | |
| care | re | | ace |

able 2: Cost of events in

in the is satis ed by a prefetch or de
Hence the only possible combination is:

- the :
  cost attributed to the prefetch is based on a demand fetch miss of in the prefetched bac into the be correctly calculated and attributed change in cost will to that prefetch.

| | as pf s cache | | conc cache | | s | tc | tra |
|---|---|---|---|---|---|---|---|
| | x prefetched replaced | | prefetched replaced | | ra | c | ss |
| | t | | ss t | t | | | |
| | t | r | t | tc t | | | |
| | t | | t | tr car | | | ac |
| | t | | ss | ss t | | | |
| | t | r | | tc ss t | | | |
| | t | | | tss ar | | | ac |
| | r | ac | ss | t | t car | | |
| | r | ac | | tc t | t car | | |
| | r | ac | | tr car | t car | | ac |

able : Cost in tra c and miss for the

| | |
|---|---|
| | |
| | , 6 |
| | , 3, 4, 8, 9 |
| | , 7 |

able : hree categories of prefetched derived from

o quantify the extra tra c and misses incurred due to a prefetch we combine the cases in 1 b with the cases in able 2. combined cases are presented in able . hese are further grouped into categories based on the amount of extra tra c and misses generated and presented in able .

From able we see that cases and save a miss and yet cause no additional are the only (1 additional line of tra ) c without saving any cache miss. Finally, lines of additional tra c and causing an additional miss.

In section 2.2 the possible outcomes of prefetched lines and the lines they replaced were described and the associated extra tra c and misses were derived. But in an associative cache (associativity 1) those may not be the only altered outcomes due to prefetching; the re-ordering of the U stac may induce side-effects on some of the regular lines that remain in the stac . In this U section, we derive the extra tra c and misses associated with these lines which are not directly associated with a prefetch.

able lists the possible combinations of the outcomes in the a line that is not associated with a prefetch.

| | | |
|---|---|---|
| | | |
| Hit | Hit | |
| Miss | Miss | |
| Miss | Hit | |
| Hit | Miss | |

able Possible outcomes in both caches for lines not associated with a prefetch

For outcomes of line at the or misses relative to the . If only these tw prefetches, then the case pairs of able ully accounted for the cost in tra c and misses of any prefetching technique. e now show that although the easily be detected and their costs are then incorporated in the taxonomy. Finally we show that the outcomes cannot occur if is a regular line.

First consider . his case is possible only when prior to this events occurred at some time since the last reference to ) : be(came the U line in the and was then replaced to accommodate a regular line; and (ii) was retained in the this reference. his reference to would additionally result in a miss and 1 cache line of additional tra c then he following simple example illustrates an occurrence of this case.

In this example the and the these caches. Initially, as shown in Figure a), suppose the contents of the set are the same in both caches (regular lines and , where line). is the Now if the prefetching technique initiates a prefetch for line , we obtain the cache state shown in )Figure he (b and replaces ; the tra c and misses due to the prefetch can be accounted for able . Note

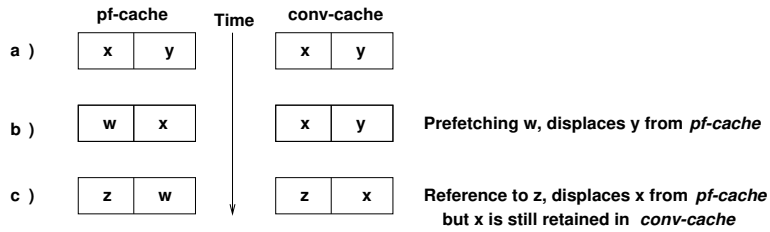


Figure Prefetch Side-effect example

that the contents of deferences do not change.
line ; we have a miss in both the and the
are shown in Figure ). Finally,(suppose that there is now a reference to , which is a regular
line. he outcome is a miss in the 1 cache line and a hit in the
of additional tra c1 and extra miss in the relative
algorithm would not have prefetched line and would have instead prefetched line . he
case therefore an indirect consequence of poor speculative prefetching. e refer to this case, case
1 , as a 1 is detected by noting when prefetching. An occurre

- ifferent bloc s are replaced on a demand miss in both caches. For example, the demand miss to in Figure and
- the bloc replaced in the Figure ) remains in the ( reference.

Finally, we analy ze the case when the outcome of line is 1,
which depends on 1 and 2 shows that this case cannot occur if is a regular line. An
outcome is thus possible only when is a prefetched line in

In the all the lines of set
contain some prefetched lines. Since both caches have the same associativity, each set of the
has at most the same number of regular lines as the

) pf-c
)

Note that whenever line is referenced, it moves to stac position in both caches
remains or becomes a regular line in the . his co
remains to show that the lemma is satis ed until the next reference to . e do this by sh

that every type of cache event either preserves ) ) or causes to no longer be a regular line in the (which satis es the lemma until the a regular line in the ). et be the associativity of reference . Consider any non-reference interval of and let be the initial portion of that interval during which is a line in the .

(a)

(b) = : In this case, the position of line , while it remains unchanged prior to this event, ) ) after this event. (

(c) = Prior to this reference, let ) and ) ( : be the position of in the and the either of the caches, its position is de ned to be (the associativity itself) following possibilities arise:

1. ) ( ), ( ) ) his increases the position of line in both caches by 1, and hence ) ) is preserved after this reference to ( . (Note that and or ) could be ) . (
2. ) ), ( ) ( ) Since ) (), line ( was referenced ( ( earlier in . However, ) implies that ( was replaced and prefetched in the Since is not referenced during reference. must be a p line which contradicts the assumption that is a regular line.

. ) ( ), ) ( ) his does not alter the position of in either cache, and hence ) ) is preserved after this reference to ( .

. ) ( ), ) ( ) Since ) ), line (was not referenced ( ( earlier in . Considering also that ), it must be that was prefetched earlier in . hus is a prefetched line in the ) increases by 1 and ) remains unchanged. However, this event occurs only after a corresponding occurrence during of event above, in which ) increases by ( 1 and ) remains unchanged. herefore, this case can only arise if ). Hence even though ) is now increased by 1, ( ) ( ) is preserved after this ( reference to .

■

: Suppose line is a hit in the and prior to this reference of , the stac position of line in the in the ). From emmas 1 and 2, this (which is, by de if nition, is a regular line. Hence the proof. ■

heorem 1 shows that the outcome has not occur for regula outcome 1 ) and (the case pairs of able , constitute a complete set of cases for the taxonomy. P he M case histogram the s provides a new and accurate basis for comparing prefetch techniques. able summari zes the cost in tra 1 c and cases. misses for the

| | | | |
|---|---|---|---|
| | | | |
| | , | | -1 |
| | 2, , | 1 , | , |
| | 1, | 2 | 1 |
| | 1 | 1 | 1 |

able Prefetch 1 axonomy

From able we observe that, except for the 1 case, the extra tra c is always one more than the extra misses. Since the 1 cases of P M completely account for all the extra tra c and the extra misses of a prefetch technique, equations and are satis ed.

Misses = ( Misses

ra = c ra + c useless prefetches + 2 prefetches + t )

Since each occurrence of the only remaining case 1 , causes 1 additional line of tra c and extra miss relative to number of case 1 occurrences. ,

From able we observe that in cases 2, and , the line that was replaced to prefetch is subsequently prefetched into the before its nex be prefetched and let it replace line (referred to). If falls or , then line must have been prefetched bac into the replaced some line, say , forming Since the . the outcome for line in

1, 2, or Since cases 2, , and require at least one more prefetch for completion, they form a part of a chain of at least 2 prefetches. herefore, the total cost associated with a case 2, prefetch should include the cost of all prefetches in the chain. Since cases 2 and are useless prefetches we focus our discussion on useful chains which start with case followed by case 1, 2, or prefetches.

- Case 1 has a cost of 2 blocks of extra traffic and 1 additional miss which makes chain of cases (1) and not favorable.
- Case 2 has a cost of 1 additional block of traffic and does not cost any extra miss, which when linked with case 1 costs a total of of extra traffic 1 miss. Furthermore, we showed above that case 2 must be followed by another prefetch of cases 1, 2 or . hus n linked chain can end with a case 2 prefetch and the linked chain s cost depends Note that a case 2 followed by a case 1 prefetch is clearly not favorable. However, case 2 followed by cases 2 or may or may not be favorable depending on the relative importance of traffic and misses. For example, if case is followed by case 2 prefetches, whi followed by a chain-ending case prefetch, the 1) blocks of extra traffic to save 1 miss. his linked list thus becomes progressively less desirable as grows.
- Case has a cost of 1 additional block of traffic and does not cost any extra miss. Since case replaced a block that is also replaced in the the linked chain.

herefore, the prefetch that is always beneficial is case . An ideal prefetching algorithm maximize the occurrence of case prefetches. It should never generate prefetches belonging to cases, , . epending on the length of the resultant chains, the prefetching technique may or may not benefit from case prefetches.

## nt ri or r

In section 2 we discussed our classification and demonstrated that PM accounts for the direct and indirect costs of all the prefetches in the program. e apply PM to variants of two common prefetching techniques and derive some insights into how to improve their performance.

Many software and hardware methods have been proposed to predict addresses and issue prefetches. Software prefetching techniques 11, 12, 1 1 , 1, derive hints from global program analysis

and insert explicit prefetch instructions only when they are deemed li ely to be useful. But existing software-based selection methods are limited to the ind of data access patterns (constants or strides) that are easiest to recogni ze at compile time. n the other hand, hardware prefetching techniques 1, , 1 , 1 1 , 1 1 , , rely on speculation about future memory-access patterns based on an observed run-time pattern history.

o illustrate the use of M we chose two well- nown hardware prefetching algorithm among the above techniques.

- Next sequential prefetching S(P), N1 , is a simple prefetching technique in which a prefetch for line ( 1) is issued 4 (if not already present in the cache) whenever bloc is accesse o control the number of useless prefetches, a variant called tagged prefetching has been proposed. In this A tag bit associated with each cache line is set when the line is brought to cache by a prefetch access. A prefetch to the next sequential line is triggered whenever there is a cache miss or when there is a hit to a bloc whose tag bit is set (prefetched). At the time initiating the prefetch, the tag bit is reset. In essence, this technique exploits an application s spatial locality and uses the data addresses to predict the next sequential address as the prefetch address. ur experiments (omitted due to space constraints) using SN with and without tags indicate that the tagged variant generally has higher accuracy, but lower coverage than SN without tags. he selective prefetching afforded by the tags does help reduce the useless and the polluting prefetches, but the overall miss rate is similar for both techniques. In this paper we use only the tagged variant of NSP.

- Shadow irectory Prefetching S (P), proposed in 1 , is a hardware based prefetching technique in which a history of the referencing patterns is maintained in a hardware table. For each line in 2, a shadow address is maintained in the 2 cache directory along with the address of the currently resident line. he shadow address refers to the next line accessed in 2 after the currently resident line was last accessed. henever we have an 2 cache miss, we issue a prefetch for the corresponding shadow address in the directory. o reduce the number of useless prefetches, a 1-bit con rmation scheme was proposed. A con rmation bit was added to each shadow entry in 2 cache directory indicating whether that line was referenced ( ) or not referenced ( ) while still in 1 after its last prefetch.

Simple Scalar simulator 2 and its functional cache simulator were used to apply P to M six benchmar rom SPECint and rom SPECfp . he benchmar s were compiled w gcc - option and run instructions with the 1 billion instructions of the results were similar for the different cache con gurations, we present results only for data prefetching using a 1 B, -way associative data cache with 2 byte lines.

| | ar c | | ss rat | | Acc rac | | ra | | ss r | | ss s |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |
| co re | | | | | | | | | | | |
| gcc | | | | | | | | | | | |
| go | | | | | | | | | | | |
| eg | | | | | | | | | | | |
| | | | | | | | | | | | |
| er | | | | | | | | | | | |
| a | | | | | | | | | | | |
| a | | | u | | | | | | | | |
| ur | | | | | | | | | | | |
| | | | | | | | | | | | |
| ave | | | | | | | | | | | |

able : Miss rate, coverage, accuracy, misses and prefetches for SP and SP N

able shows the miss rate, coverage, accuracy, misses, and prefetches for SP and SP N. n average, NSP has higher coverage and lower accuracy than SP. For benchmar s with sequential access patterns (e.g., ), NSP outperforms SP. SP is generally far more selective than NSP as seen from the total tra c +( ) in able SP has, ( only of NSP tra SPECint and for SPECfp benchmar ) implying that most of the NSP tra c is useless. For example, in 1. for NSP, the miss rate is down to fo SP, but the total tra c is up from for SP to. 1 million lines for NSP. hese metrics do not separately quantify the extra tra c and extra misses due to polluting and useless prefetches and hence they do not provide much insights for improving the techniques.

Figure shows the distribution of prefetches into the categories identi ed using our taxonor From Figure we see that has a higher fraction of useful prefetches than NSP (except for ). con rming the lower accuracy for SP. N he additional tra SP c of N is wasted on polluting and useless prefetches (e.g., in the polluting prefetches are 1 up f for SP to 1 for SP N). exhibits a classic example of polluting and useless prefetches

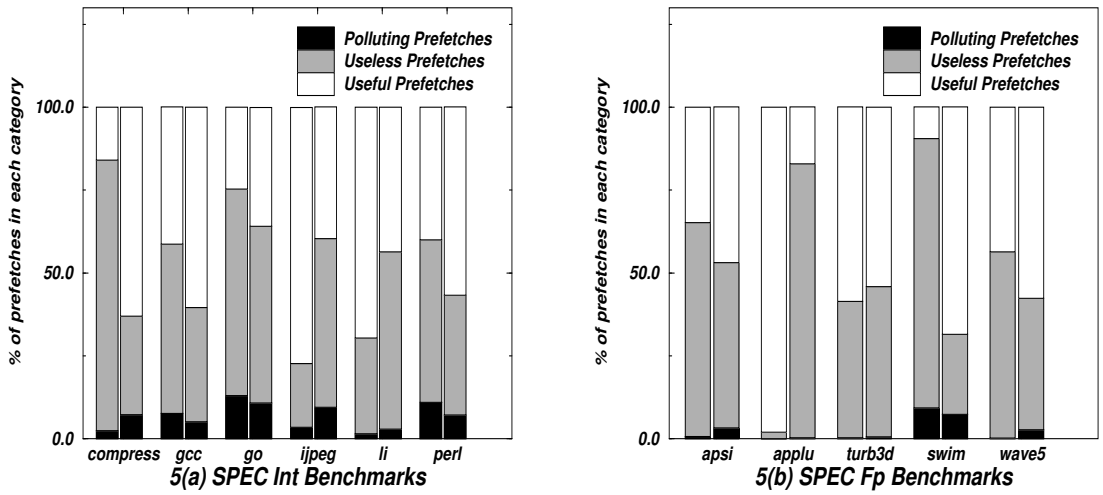


Figure Classi : cation of prefetches for NSP and SP

(left bar = SP, right bar = SP)

( Total prefetches in each scheme normalized to 1)

outweighing the effects of the useful prefetches (useless prefetches are upon 2 for SP to 1 for NSP).

| | ar cr | |
|---|---|---|
| | | |
| co re | | |
| gcc | | |
| go | | |
| eg | | |
| | | |
| er | | |

| tc c | ar r | cts |
|---|---|---|
| | | |
| a | | |
| a | | u |
| ur | | |
| | | |
| ave | | |

tc

able Prefetch Side-Effects (Case 1 )

able presents the number of side-effect cases identi ed in section 2 showing indirectl effect of speculative prefetching on the U stac . 1 occurrences compared to the total prefetches ( . to ) as seen from able prefetch .

From the histogram of the prefetches by category we identi ed the need for a lter to decrease the polluting prefetches. e have proposed such a P lter and use M to evaluate its effectiveness in the next section.

## t tic i t r

e have proposed a tatic Filter (SF) ( which attempts to reduce polluting prefetches so as to minimi ze the impact of the prefetching technique on the bandwidth requirements and at the same time to retain the achieved reduction in miss rate.

For a given prefetching technique an access to line is called if the access to may initiate a prefetch of . Most hardware prefetching techniques use data addresses as prefetch triggers (e.g. SP uses an access to line as a trigger to prefetch line ). However, many of these schemes use additional enabling mechanisms to qualify the trigger i.e., to actually perform the prefetch when its trigger occurs is the enable conditio For example, SP does not prefetch line if the access to is not a cache mis SF) uses pro ling to provide an additional enable mechanism that must also be satis ed.

he pro le is used to determine which load instructions tend to generate data references that are useful prefetch triggers, and mar s those loads as SF disables the hardware . prefetch mechanism unless the load ma ing a trigger access is enabled. Since the characteristics of the load instructions of the program tend to be stable across runs, pro ling is an effective way to select enabled loads.

In the pro ling phase of SF, we simulate a conventional cache without prefetching; whenever a line is brought into cache we maintain (or create) an entry in a table (we used an table of U 2 entries for our experiments) that records the current time and the PC of the load instruction that accessed the data. henever there is a demand miss to cache line we chec if at least one its recently brought into cache, 2, and the load instruction, , that accessed to begin tour. he prefetching technique would have prefetched along with at time 2. if that prefetch is 1 of the most recent access to , the line chosen for replacement now (to accommodate ): If 1 the prefetch is deemed 1 it is not. ; if 2

e use a heuristic to select loads to be enabled. For each load instruction we nd the ratio of the number of misses it incurs to the number of If the potentially useful prefetches are more than of the misses we mar as enabled.

In the implementation phase, the baseline technique with SF initiates a prefetch prefetch triggering data access is issued by an enabled load. ur results have sho n that SF achieves

*us*

a signi cant reduction in the tra c requirements of the prefetching techniques while preserving the reduction in miss rate that they achieve without SF. In the context of this paper, SF shows how our metrics can be used to improve the quality of existing prefetch techniques.

e present the miss rate in junction with the total tra c to illustrate the tradeoffs between misses and bandwidth. he thefulness and accuracy of prefetches are assessed via the P M prefetch histogram.

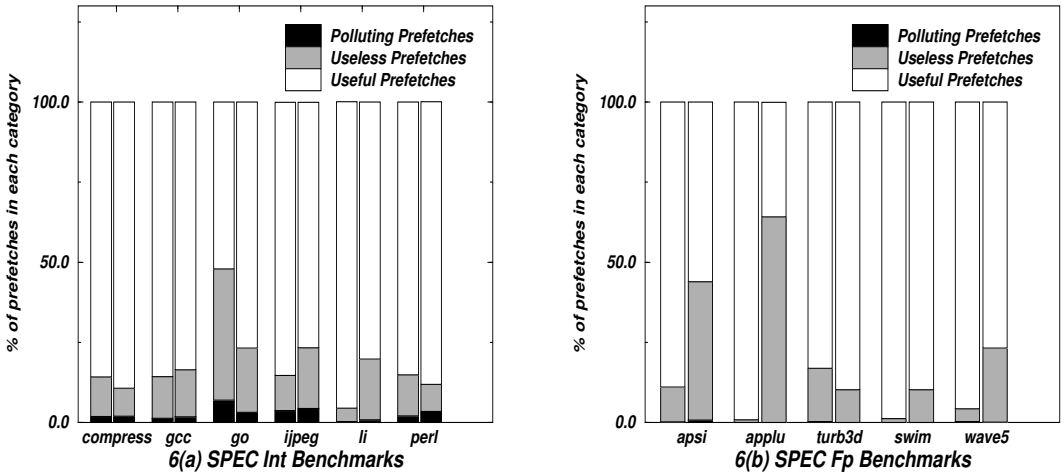


Figure : Classi cation of prefetches in SPN and SF-S P
(left bar = SF- SPN, right bar = SF-S P)
( Total prefetches in each scheme normali zed to )

- Figure shows the improved prefetch histogram when SF is applied to NSP and S P. Furthermore, as the total prefetches issued by SF- SPN and SF-S P differ signi cantly, each is far less than those issued using SPN and S P alone, the improvement in the histogram is even more signi cant.

  For both SPECint and SPECfp benchmar SF- SPN has only about 2 to g., in , the polluting prefetches for SF- SPN vs. 1 (e. for SPN). verall, SF- SPN is dominated by , indicating of SF. However, for some benchmar s li SF may be too selective and decrease coverage.

Similarly, SF-S P has only 2 to SF- SPN, we see that SF-S P, unfortunately, also decreases the total number of miss rate. (as seen in Figure 1). SF s reduction in tra c will not be effective unless SF preserves the decrease in miss ratio achieved by SPN or S P alone.

•

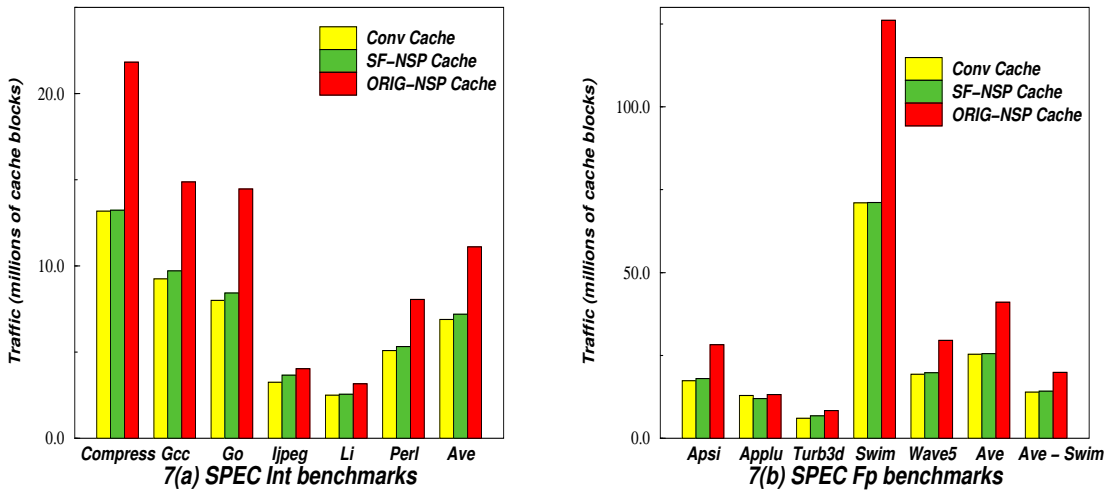


Figure SP : : Na c (millions of cache blocks) B, -way, In all the subsequent gures the leftmost bar is the result for conventional cache, the second bar is for a cache using SF on a baseline technique and the rightmost bar is for a cache using baseline technique alone.

Figures ) and (a ) present the total tra c for SPECint and SPECfp benchmar s using NSP as the baseline technique. Using SF we observe an average of reduc for SPECint benchmar s. Although SPECfp benchmar s are nown to cause sequential patterns, SF still achieves an average of 2 reduction in tra c. Here and (excluding in the subsequent gures, the last group of bars shows the average results excluding SP has almost twice as much tra c as SPN his leads to more cache pollution, as we will see when we compare the corresponding miss rates for SPN and SF- SPN in Figure (b).

Figures ) and (a) present the total tra c using S P as the baseline technique. Since S P is more selective than SPN, we do not achieve the same the improvement using SF on S P as on SPN. SF reduces tra c by for 2 SPECint and 1 for SPECfp benchmar s.

•

Figures (a) and (b) show the miss rate for SPECint and SPECfp benchmar s using SF on

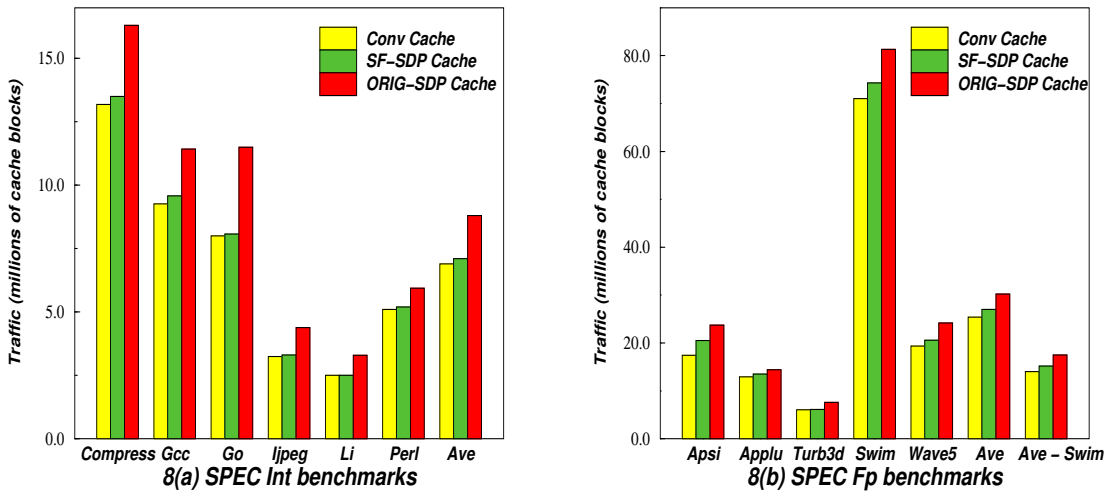


Figure S P : : rmillions of bloc ): Cache: 1 B, -way,

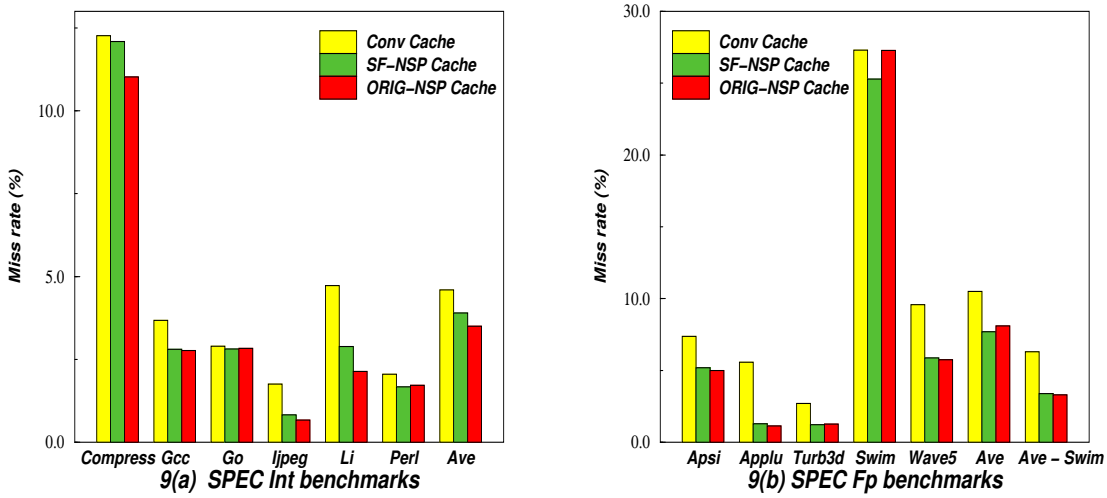


Figure : SPN Miss rate: Cache: B, -way, 2

NSP. SPN achieves an average miss rate reduction of 2 for SPECint and for SPECfp relative to a conventional cache. SF- SPN reduces the miss rate by 2 for SPECint and and SPECfp respectively, indicating that we are not eliminating very many useful prefetches by using SF. From Figures and , average useful SPEC prefetches for from for SPN to 1 SF- for SPN. However, for benchmarks with predominantly sequential access patterns, (e.g. ) SF may be too selective on the other hand for and selectivity is more important, the SPN miss ratio is well preserved. A clear trade-off between misses and traffic is seen for SP a 1 reduction in miss rate is achieved with a increase in traffic for SPN. This may or may not be desirable depending on the available bandwidth.

The high miss rate (average of ) of SPECfp, vs. for SPECint benchmarks, suggests that cache may be too small to capture the working set of these benchmarks. This makes prefetching more critical for improving performance; SP and SPN have comparable performance. For SP has more misses than SF- SPN due to cache pollution caused by excessive prefetching. This is confirmed by our observation in Figure ) that the SPN:SF- SPN prefetch traffic ratio for about 1 times is very high (

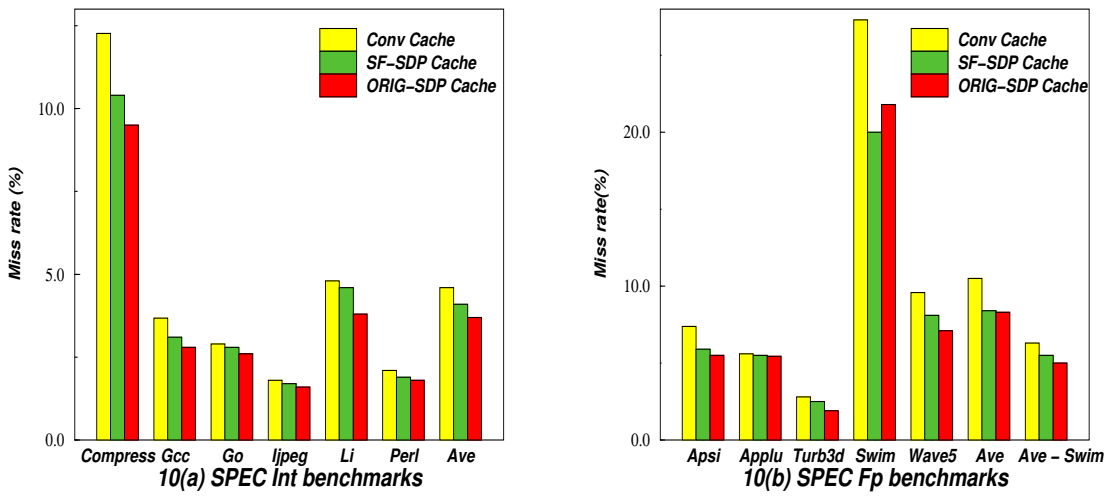


Figure 1 S P : Miss rate: Cache: B, -way, 2

Figures 1 ) and 1 (a ) show the miss ratio using S P as the baseline technique. Neither P nor SF-S P shows a significant improvement in miss rate for these benchmarks. SF-S P has a 1 1 to increase in the miss rate relative to S P. However, S P issues to prefetches than SF-S P in order to achieve this reduction in miss rate. SF-S P performs as well as SF- SPN for SPECint benchmarks, but not as well for SPECfp benchmarks.

ur results using SP and NSP as baseline techniques show that PM did identify the useless and polluting prefetches and that SF did significantly reduce these harmful prefetches while retaining nearly all of the useful prefetches of the baseline techniques. SF does, as claimed, reduce the bandwidth requirement of a baseline technique while preserving its reduction in miss rate.

## onc u ion

PM analyzes each prefetch 1 or 2 blocks of additional tra 1, c and 1 a, -or + net change in misses that it causes relative to a conventional cache. he ey to this classi cati is the simultaneous simulation of two caches a cache with prefetching and a conventional cache. he three jor contributions in this paper are,

- A classi cation that completely describes the possible outcomes that encompass a prefetched line and the line it evicts from the cache.
- A classi cation which further describes the effect of prefetching on the ordering within a cache set (case 1) and the indirect effect this has on the miss rate and tra c.
- An Analysis of the chaining effects among prefetches so as to understand how even a useful prefetch (case 2) may not be bene cial in a broader context.

Using the above classi cation we derived new metrics to evaluate a given prefetching technique by quantifying the additional tra c penalty that each prefetch incurs in order to cache miss and hide the memory access latency.

e have demonstrated the usefulness of PM by applying it to two implementable prefetching techniques (SP and NSP). Based on the prefetch classi cation observed for SP and NSP we developed a static lter to reduce eliminate polluting prefetches. ur results show that SF eliminates nearly all the polluting prefetches and achieves a signi cant reduction in the prefetch tra c of the baseline techniques with only a modest increase in miss rate.

## r nc


e y ec o e or ac a roc or re er *roceedings of the Int l ymposium on icroarchitecture, ecember* ,

urger u e e o ca *Technical oport T* e *of isconsin, un*

aa e a er or e are c y re g *roceedings of the th ympos Architectural upport for rogramming anguages and Operating ystems, April*

ar ey u c a g ae ory y e av or o ere e ear u e c *IB ournal of esearch and evelopment, ol* ,

e aer g ory e e a e cy v ouc g a occ g e *ifth Inac re e ternational onference on Architectural upport for rogramming anguages an October , pp*

e aer ea u ore yro a are aoa re c ar eg e c ro *ceedings of the l st Annu ohos ium omputer Architecture, April*

e a a g u a a cce croarc ec ure or ca roou or er o er e a a re c eg *roceedings of the l ymposiumInt icroarchitec ture, ovember , pp*

u e eoc c e g o r *IB Tech. e isclosure Bull. ,*

o e ru a c g g c orar *roovdings of the re l ympo th Int sium on omputer Architecture, ay , pp*

ou rov reg a e a er e eore o y o a uca oc a ve y e ac c u er *roceedings of the e l ymposiumInt omputer Architecture, a , pp*

er a rc ec ure avor ro o oa a c e re g *roceedings of the Int l ymposium on omputer Architecture, ay ,*

a c u goore ae are c re g o er a e o e ca ve e ro e v *roceedings of the l thymposiumInt icroarchitecture, ovembe , pp*

ry u er re c a ego ve ar rcurare *roceedings of the th Int l onference on Architectural upport for rogramming anguages and Opera , pp*

ro ma oarr o a o ory acce e o cac ea a eror e ve o a er c rogra u *roceedings of the l ofInt ce on upercomputing, a*

ry o a u a e g a o o en gorva or ua gre *ro* e *ceedings of the l th Int onference on Architectural upport for rogramming a ystems, October , pp*

ereo e u ceca e o rearac eg or ye av ag a eco rec ory or a cce e y uoc . . *eb atent ,*

o ovo o e a e c g e re a aor ruc ure *roceed ings of the l th Int onference on Architectural upport for rogramming ang ystems, October , pp*

o o ve u ero c re eg a aor ruc ure *roceedings of the l th Int mposium on omputer Architecture, ay*

e or e e*A* ac *pt. omputing urveys, , pp*

re ove ce re or g revu e