Exploration of the Spatial Locality on Emerging Applications and the Consequences for Cache Performance

Martin Kämpe and Fredrik Dahlgren*

Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
mkampe@ce.chalmers.se

*Ericsson Mobile Communications
Scheelevagen 15, SE-221 83 Lund, Sweden
fredrik.dahlgren@ecs.ericsson.se

Abstract

The performance gap between processors and memory is increasing, making the cache hit-rate paramount for performance. Studies show room for improvement, especially in data caches. The cache effectiveness is dictated by software locality, hence the software behavior directs the cache performance. This paper presents a framework for studying spatial locality. It focus on the characteristics of the spatial locality in terms of closeness in time and space, to get the amount of accessed sequential data and the potential for cache hits. By using the framework we gain knowledge for improving the cache performance.

Our experiment consists of a program driven simulator and 11 important applications. We show a large performance potential in the data cache with up to 75% less miss rate, exploiting spatial locality. In order to investigate where potential bottlenecks are located we make a simple implementation of a scheme to exploit this spatial locality.

1 Introduction

Over the last few years the performance gain of processors has had a higher pace than the decrease in memory latency. Adding a cache reduces this gap, but programs still spend a large amount of time stalling on data cache misses [2]. The effectiveness of a cache depends on the locality in executed software. Therefore, by analyzing the amount and characteristics of memory reference locality, we will not only be able to explain the reason for the large amount of cache misses, but also find directions in development of more effective memory hierarchies.

To analyze the program behavior with respect to cache performance, a framework for measuring locality is needed. Such a framework is presented here: *Immediate Spatial* (IS) locality, a subset of *spatial* locality.

The framework is based on SimICS [8], a program-driven simulator. The framework has been evaluated with 11 applications; DSS [14], OLTP [15], SPEC95 [12], multimedia [9] [17], and a 3D-game [7]. All executions, except SPEC95, include Linux, since the operating system influence cache performance [10].

Our results indicate a large potential to reduce the miss rate in a data cache. Between 50% and 90% of all blocks participate in an IS sequence. Interestingly, most of the IS sequences are generated by multiple instructions, hence small loops are not the major source for the exploitable spatial locality. We show a huge difference between the characteristics of different applications, and among the user code, kernel, and system libraries. This indicates useless prefetches for hardware-based prefetching techniques.

Although data is treated differently at different parts of the execution, the IS behavior of the data that are accessed by the same instruction is relatively consistent. The miss rate could ideally be reduced by up to 75% for a 16KB cache, if this behavior is exploited. In an attempt to exploit our findings, we explored the gains from a novel device that makes use of IS for prefetching. The first attempt was able to reduce the miss rate by up to 23% with good accuracy (i.e. few useless prefetches). We also explore the incompleteness of our prefetching algorithm.

This paper is divided into the following sections: Section 2 describes spatial locality. We also present our framework and introduce the concept of Immediate Spatial Locality. In Section 3 we describe the experimental methodology, while Section 4 presents our experimental results. Discussion and related work is presented in Section 5. The paper is concluded in Section 6.

2 Spatial locality

A large part of the execution time in a program is spent on data stalls, sometimes over 70% [4]. This problem is present in both uni- and multiprocessor systems, but data stall time tends to increase with the number of processors in the system [2]. The stall time is related to the miss rate in the cache. A simple solution to reduce the miss rate is to increase the block size [1]. However, increasing the block size may introduce other problems, such as cache pollution [13], since structure of spatial locality varies among data.

In this section, we start in Section 2.1 by establishing the critical parameters for techniques measuring spatial locality, and by introducing the concept of *Immediate Spatial Locality* (IS). In Section 2.2, we describe the technique for measuring IS with our experimental infrastructure.

2.1 Measuring spatial locality

A common definition of spatial locality is: "items whose addresses are near one another tend to be referenced close together in time" [6]. In addition to providing a good abstract description of spatial locality, this definition also points out the two most important parameters for any technique measuring spatial locality of a program: the closeness in space ("whose addresses are near one another") and the closeness in time ("close together in time").

Memory accesses have spatial locality if the criteria in terms of closeness in space and time are fulfilled. Since the criteria can be defined in virtually any way, it is important to specify the purpose of the metric. Our goal is to understand the potential for data cache miss rate reductions by exploiting spatial locality.

2.1.1 Closeness in space. Our focus is to study an ordinary cache, therefore the closeness in space is defined as the size of one block. Allowing a larger distance between addresses, has two consequences. First, if consecutive data is brought into the cache, all loaded blocks might not be accessed. Second, if the technique should only fetch those blocks that are accessed, it must be capable of identifying the presence of spatial locality, as well as the block distribution of such a pattern. Even though our study is limited to consecutive blocks, techniques have been proposed for non consecutive patterns in specific cases, such as stride accesses [3].

2.1.2 Closeness in time. One way of determining the amount of spatial locality at block level, where closeness in space is defined as consecutive blocks, would be to execute the whole program and for each access detect consecutive block accesses. The problem of such approach is that the measured amount of locality will not reflect the ability of a cache to hold that block until it is requested.

Instead, we measure spatial locality based on the principle that blocks at consecutive addresses should be accessed close enough that they are simultaneously present in the cache. This means that the closeness in time is determined by the interference of interjacent accesses. For the metric to better reflect the performance improving potential by exploiting spatial locality, the criteria is based on the cache interferences in a small cache. We denote the criteria of closeness in time as *Immediate Spatial Locality*, IS.

In the next section, we describe the technique used to measure the amount and characteristics of IS Locality.

2.2 Measuring immediate spatial locality

To describe the technique for measuring Immediate Spacial Locality, a concept of a spatial sequence is defined. A *spatial sequence* is the set of accesses mutually spatially local according to the concept of IS locality. According to Section 2.1, two memory accesses are *spatially local* if they are consecutive blocks, and issued close enough that they reside in a detection buffer at the same time. For an access to join a spatial sequence, it must be spatially local to the access that most recently joined the sequence.

In our measurements, we used a 64 block large, 4-way set-associative buffer to detect IS sequences. Observe that the detection buffer is not a proposed hardware component, but rather a means to instrument the simulator for measurements. The default block size is 32 bytes, but other sizes have been studied. The replacement policy is FIFO. To insure that detected IS sequences can be exploited by a cache, the buffer size is much smaller than the cache.

For each access to a block not in the detection buffer, we insert the block and investigate if prior or subsequent blocks are in the detection buffer. If so, the accesses belong to the same spatial sequence. We track all ongoing spatial sequences to measure their length, which addresses they consists of, and the instruction generating the access. Hence we gain statistics for the number and length of spatial sequences, as well as statistics for how blocks and instructions behave with respect to spatial sequences.

In the following sections, we will present our findings. We also describe which consequences they have on the cache behavior and techniques to reduce the miss rate.

3 Experimental environment

We start in Section 3.1 explaining the methodology. The simulation environment used in this study, is described in Section 3.2. Section 3.3 describe the benchmarks.

3.1 Methodology

Our approach to eventually improve miss rate is to first study the amount and type of spatial locality. Hereby, we get an upper limit on how much it is possible to lower the miss rate, as well as characteristics of the spatial locality present in the program. An upper limit gives the ability to tell how well a certain technique harvests the spatial locality and the room for improvement. The characteristics of spatial locality are also important to know before any implementation is made, since such knowledge gives better understanding for how an implementation should work.

3.2 Simulation environment

We have used SimICS [8] to execute our benchmark applications. It is a program driven simulator implementing the Sparc V8 instruction set architecture, and the sun4m system-level architecture including I/O.

For the processor simulator, we developed all memory system simulators necessary for this study. The simulation allows for non-intrusive instrumentation of execution activities, and allows us to trace a memory access back to which instruction on the source code that caused it.

Our study focus on a level-one cache with 16KB, 4-way associativity, and a 32 byte block size, but we also study the effects of different cache and block size.

3.3 Benchmarks

The focus in this study is on the Decision-Support Systems (DSS) and the On-Line Transaction Processing (OLTP) database applications, because of their importance [11]. The benchmarks for DSS and OLTP are TPC-D [14], and

TPC-B [15], respectively. TPC-B has been replaced by TPC-C as a benchmark, but they have similar memory behavior [2]. It is important to point out that we do not attempt to benchmark our system, but rather to study the characteristics of spatial locality. The TPC benchmarks are simulated with the Linux 2.0.35 operating system, and the database engine is MySQL [16]. MySQL is a multi-threaded database management system (DBSM) designed for high performance. The reason for using MySQL and Linux is that we have access to the source code.

For TPC-B, the DBSM executes 400 transactions. We use 40 branches, and the size of the database is approximately 600MB, excluding metadata and log table. In TPC-D, each query is converted to a query plan where the typical operation is a scan. There are two types of scan operations: index- and sequential scan. In index scan only the relevant parts of a database entry are scanned. In sequential scan all the entries are scanned. Therefore, more spatial locality is expected in sequential. We include two queries from the TPC-D benchmark, Q3 and Q6, where Q3 uses index scan and Q6 uses sequential scan.

We include other applications in our study to be more general in conclusions, and to relate our findings from database workload to other programs. Five SPEC95 [12] benchmarks is included, two integer (Compress and GO) and three floating point (Fppp, Mgrid, and Tomcatv). We also include a 3D game Quake [7], an Mpeg decoding tool [9], and a raytracing program, Raytrace, SPLASH-2 [17]. The benchmark characteristics are displayed in Table 1.

Benchmark	In data	Number of data accesses	Missrate in a 16KB, 4 way data cache	Comment
Go	Ref. data	20964M	4.5%	Plays the game Go against itself.
Mpeg	lemon.mpg	72M	0.8%	Decoding of a mpeg picture.
Raytrace	car.env	258M	4.1%	Raytraces a car, SPLASH-2.
Quake	timedemo	1031M	2.8%	3D game demo.
Compress	Ref. data	19090M	7.2%	Compress large text files.
Fppp	Ref. data	89895M	3.5%	Performs multi-electron derivatives.
Mgrid	Ref. data	1656M	3.6%	Calculation of a 3D potential field.
Tomcatv	Ref. data	57423M	6.4%	Two-dimensional coordinate system.
TPCB	600MB	296M	2.1%	OLTP benchmark.
TPCD, Q3	200MB	70M	3.6%	DSS benchmark, indexed scan.
TPCD, Q6	200MB	343M	1.8%	DSS benchmark, sequential scan.

Table 1: Benchmarks

4 Quantifying spatial locality

Our quantitative results on the amount of Immediate Spatial Locality is presented in Section 4.1, while Section 4.2 shows the potential gain based on different approaches. In Section 4.3 we vary the block and cache size. Finally we discuss a technique for how to exploit IS in Section 4.4.

4.1 Amount of immediate spatial locality

In Figure 1, we show the length distribution of spatial sequences in number of blocks. Each bar consists of five layers (from bottom): fraction of all spatial sequences of length one (NO IS), two or three (IS 2), four to seven (IS 4), eight to 15 (IS 8), and finally 16 or longer. As can be seen, a clear majority of all sequences, excluding NO IS, have a length of just two blocks. However, we also see that some applications, FPPP and MGRID have a significant fraction of sequences of lengths between two and eight.

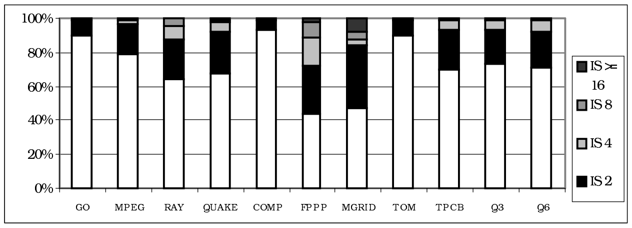


Figure 1: The number of IS sequences of different length in benchmarks.

When looking at how large a fraction of all memory accesses that belong to IS sequences, an important observation is that many accesses are to blocks that have recently been brought into the cache. Since the goal of this study is to explore the cache effectiveness for spatial locality, we do not want the figures to be affected by such immediate reuse of blocks. Therefore, we focus on the accesses to blocks that are not in the small detection buffer. In Figure 2, we show the fraction of all such accesses that belong to spatial sequences, and the length of it.

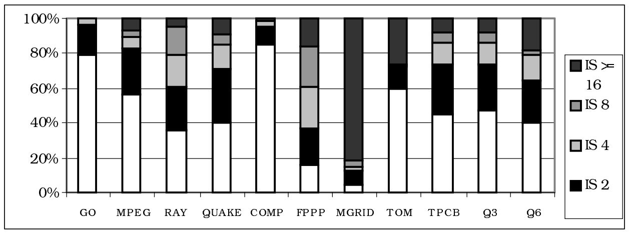


Figure 2: The number of blocks in IS sequences of different lengths.

For Mgrid in Figure 2, we see that the majority of all block misses in the detection buffer belong to IS sequences of length 16 or longer. This indicates a great potential for selective prefetching. By comparing that result with Figure 1, we see that only around 15% of all IS sequences for Mgrid has length of 16 or longer. This clearly shows that many of these sequences are very long. By contrast, Go and Compress are dominated by accesses to non-spatial data. Comparing different benchmarks we see that increasing the block size is not beneficial. For many applications, this would lead to less utilization of the cache. For a small L1 cache, this could be devastating for performance.

There are two principle sources for spatial data sequences: accesses to data structures that are larger than the block size or crossing block boundaries, and accesses to an array or linked data structures. In the former case, the spatial sequence is often generated by different load instruction, while the latter is often generated by the same load instruction within a loop. The distribution of generating instructions for IS sequences is displayed in Figure 3

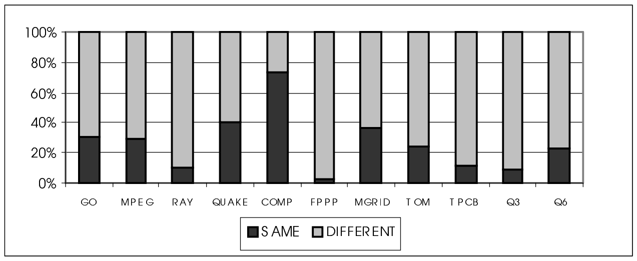


Figure 3: IS sequences generated by the same or different instructions.

All application executions, except SPEC95, included the operating system. In Figure 4, the IS sequences are divided for application, libraries, and operating system, as opposed to the whole execution in Figure 1. These parts has different characteristics when it comes to IS sequence behavior. Not only is the difference between the application, libraries, and OS large for each application, there is no consistent behavior for the library software or the OS across the applications. The OS in TPC-B does not have IS sequences with the size one (NO IS), while the same OS in TPC-D-q3 has a large part of NO IS. Except TPC-D, the library shows little access to non-spatial data. The percent number above each column in Figure 4 shows the amount of all IS sequences that each part account for.

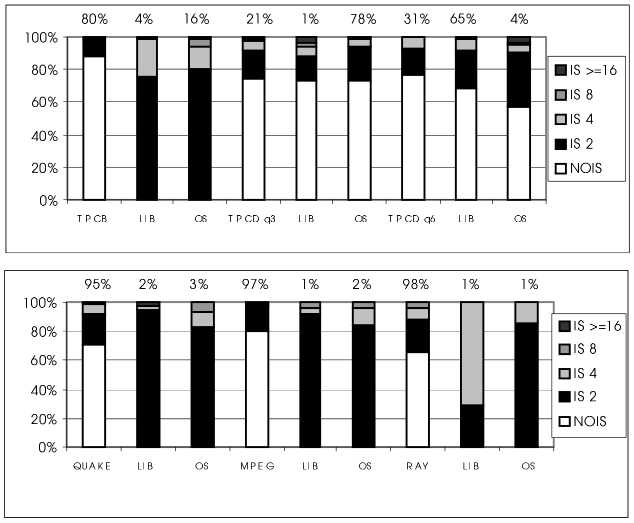


Figure 4: IS length behavior for all benchmarks accesses divided for application, libraries, and operating system.

Overall, we can see that there is a large amount of IS locality looking at the dynamic accesses during execution. Based on above results, it seem that the block size is a less effective means to exploit the IS locality. In the next section we investigate other methods of exploiting this locality.

4.2 Potential of immediate spatial locality

There are several techniques to exploit the IS locality, such as tagging data blocks as having IS locality of a certain amount, or tagging the instructions that generate the IS sequences. Tagging data can be done at either a block- or a page granularity. While the former is more fine grained and potentially more accurate, the latter implies less overhead and could potentially be implemented as an addition to the virtual-to-physical address translation. To tag instructions has the advantage of small and constant size, since typically the number of instructions is smaller than the number of data blocks and the data set size can vary significantly.

To gain an intuition on the effectiveness of such tagging of data or instructions, we have analyzed how often the same block, page, or instruction is involved with similar locality characteristics. Two metrics are used to determine the best item to tag. The first metric is the deviation from the main IS sequential size. To determine this, each data block, instruction and page will have an output that consists of a histogram of the different sizes a particular instruction, data block, and page will generate. The diagram in Figure 5 displays the difference in deviation between data block, instruction, and page for all references. Although only two diagrams are displayed in Figure 5, the trend is the same for all the benchmarks.

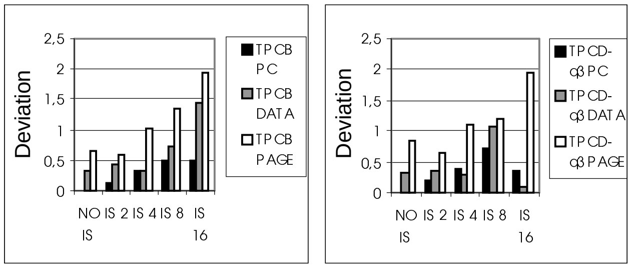


Figure 5: Deviation from main IS length for instructions, data blocks, and pages for different main IS lengths. Only two application are shown, TPCB and TPCD-q3.

Tagging instructions clearly has the smallest deviation compared to data blocks and pages. However, tagging data blocks is not far behind. The second metric is locality among IS sequences length. If the two sizes interleave we would prefetch the wrong number of blocks each time, the best behavior among these IS sequences is to use one size first and then the other size, since we would like to have as few wrong guesses as possible when it comes to the size of the IS sequence. The metric consists of dividing the number of different sizes an IS sequence has with the number of times the IS sequence has changed size plus one. The ideal value is one for this metric, then the size is only changed once for each size category. The result for all the benchmarks can be viewed in Figure 6.

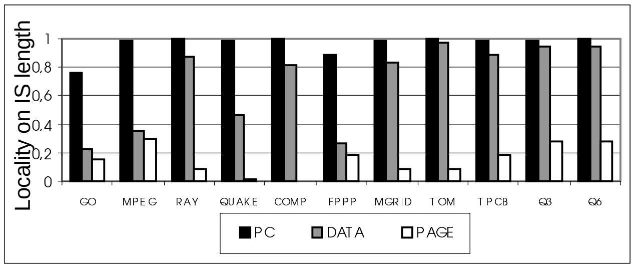


Figure 6: Locality for IS sequences length for all references.

As with the first metric we can conclude that tagging instructions is the best choice, but as for the first metric tagging data blocks is not far behind for some of the benchmarks. However, since the amount of data blocks to tag is larger then the amount of instructions, Figure 7, we conclude that it is best to tag instructions. This is especially true for Compress, where the data bar is so large that it will not fit in the diagram. Observe that we only need to tag the first instruction or data block in each IS sequence.

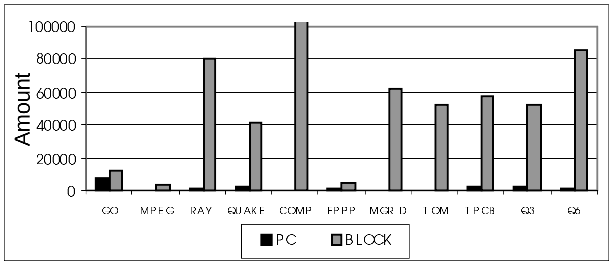
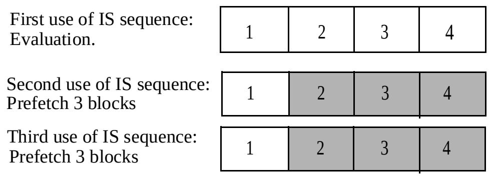


Figure 7: The amount of memory instructions or data blocks to tag for all IS sequences.

An important observation is that sequential sequences tend to be reused. This behavior can be used as an accurate sequential prefetch mechanism. As an example let us imagine we have an IS sequence of size 4 and that IS sequence is appearing 3 times. The first time it appears, it is evaluated and the size is determined. The two following times 3 blocks is prefetched when the first appears, since we know the size. In this example we will reduce the miss rate by 50%. The example is displayed in Figure 8.



First use of IS sequence: Evaluation.
1
2
3
4
Second use of IS sequence: Prefetch 3 blocks
1
2
3
4
Third use of IS sequence: Prefetch 3 blocks
1
2
3
4

Figure 8: Possible miss rate reduction.

The possible miss rate reduction for the benchmarks are calculated as shown in Figure 8. The first time an IS sequence appears, we need to determine its size. The next time the first block of an evaluated IS sequence occurs, we will know the size and therefore we can start to prefetch the rest of the sequence. Since an IS sequence can change size, we have to account for that in our calculation. We assume the worst scenario, if an IS sequence changes size we will count the first occurrence of that size as an evaluation, thus no miss rate reduction is counted.

In an ordinary cache, we miss on the first occurrence of an IS sequence. However, the question is how long that IS sequence remains in the cache once inserted. This is important, since the next time the IS sequence occurs, we can not calculate it as a possible miss rate reduction if it is still present in our cache. Therefore the cache, that we would like to know how much reduction of miss rate is possible, is simulated in parallel to our detection buffer. Once an IS sequence is detected in our detection buffer, we look into our cache to see if it is present there. If it is not, and this is not the first time this IS sequence occurs with this size, we calculate it as a possible miss rate reduction. The calculated results are presented in Figure 9.

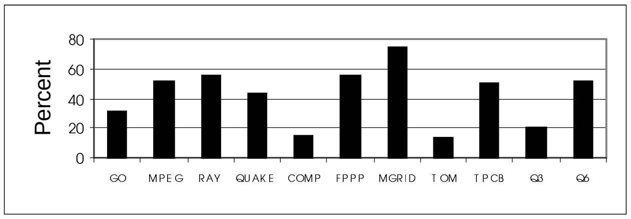


Figure 9: Calculate miss rate reduction using IS.

As shown in the Figure 9, there is a large potential to reduce miss rate for most of the benchmarks. Two out of the three low reduction values can be explained. Compress uses three arrays, where it collects ASCII characters to compress. In every loop a character is collected from each array and a calculation to compress these characters follows. The compression part is so long that the array will be evicted from the cache if put into it. For Q3 in TPC-D the explanation is that the query uses indexed scanning which has a small amount of spatial locality.

4.3 Variation in block and cache size

Our main focus is to study databases, therefore we only display a variation analysis on TPC-B. The parameters varied were the block size and the size of the cache.

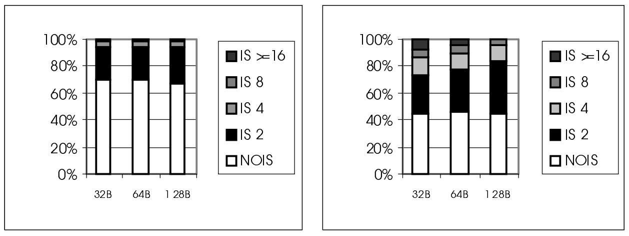


Figure 10: The number of IS sequences (left) and the amount of blocks (right) in the IS sequences for different block sizes.

The distribution of IS sequences for block size 32, 64, and 128 bytes are displayed in Figure 10 (left). 32 bytes is the default size as in Figure 1. The distribution is almost equal for 32 and 64 bytes, but for 128 bytes, the IS distribution has a larger part of IS 2. This is due to that the IS 8 sequences for 32 bytes are transferred to IS 2 in a 128 bytes block size. This effect can also be seen in the right diagram, which displays the amount of blocks in the IS sequences. We can also observe that TPC-B has very few IS sequences that are longer than 256 words, since the 128 byte bar has very few IS sequences larger then 16.

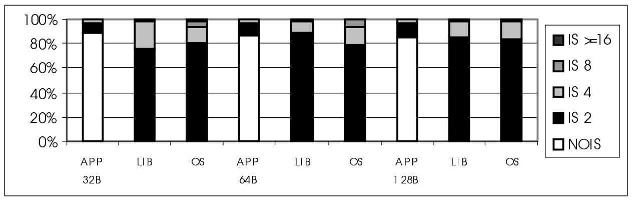


Figure 11: IS length distribution for different block size divided for application, library, and operating system.

In Figure 11 we can see that the distribution of IS lengths does not shift much for application, library and operating system. Even though the distribution does not change much, the actual number of IS sequences decreases, Figure 12, as the block size increases. With a larger block size, the IS sequences are reclassified. If an IS sequence is classified as IS 2 for 32 byte blocks, it will be NO IS for 64 byte blocks. Since a significant part of the sequences are IS 2 for 32 bytes block, the number of sequences will be reduced as for 64 bytes block. The same is true for the transformation from 64 to 128 bytes.

The tagging characteristics, Figure 6 and Figure 7, will not change with increased block size. Instructions are still the best option for tagging.

The increased block size will reduce the potential for IS locality, however as displayed in Figure 12, there is still a significant potential left. The miss rate reduction for 32 bytes is 50% and drops to 33% for 64 bytes block. For 128 bytes block there is a possible miss rate reduction of 26%. However, a larger block size tend to introduce cache pollution [13].

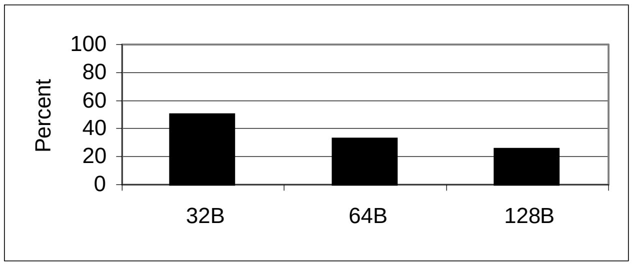


Figure 12: Miss rate reduction for different block sizes.

So far we only studied the impact of block size, but the cache size will also have an impact on the possible miss rate reduction. A larger cache allows a IS sequence to reside a longer time inside it. Therefore the potential of IS locality decreases, since IS sequences close in time will already be in the cache. However, Figure 13 still shows a potential for miss rate reduction in larger caches. Hereby we can conclude that a large amount of the IS sequences is so far apart in time that they can be prefetched.

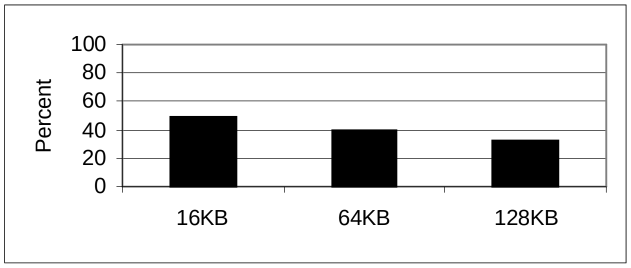


Figure 13: Possible miss rate reduction for different cache sizes.

Even though the main purpose of this paper is to explore the amount and characteristics of locality in emerging applications, the next section presents and analyzes a novel technique that builds on the results shown above in Section 4.1 and Section 4.2.

4.4 A technique that exploits IS locality

In a first attempt to investigate the behavior of our IS approach, a straight forward hardware mechanism is evaluated. This mechanism, called Dynamic Selective Sequential Prefetching (DSSP) and displayed in Figure 14, adds three parts to an ordinary cache. It is, however, important to point out that this hardware mechanism does not have the same behavior as the detection mechanism presented in Section 2.2, but it is exploiting the same IS locality.

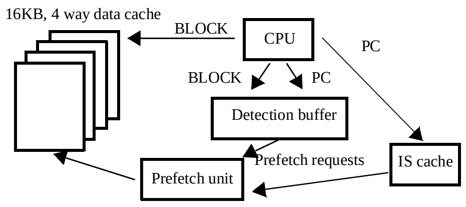


Figure 14: DSSP mechanism for exploiting IS sequences.

The ordinary cache hierarchy is not changed, a block request from the CPU is performed as for ordinary caches. However, during this request, the block address (BLOCK) and the address to the instruction (PC) that causes the data request are transferred to the Detection buffer. The Detection buffer is a fully associative cache with 16 entries and a LRU replacement strategy. The main purpose of the detection buffer is to find the IS sequence and detect the size.

If a miss occurs in the data cache, one of three actions can happen in the Detection buffer. First, if the instruction address is found, the associated size will be transferred to the prefetch unit with the first data block. The second action occurs if the data block address is found to be a successor to any of the detected IS sequence in the Detection buffer, then the size is incremented and the entry is LRU updated in the Detection buffer. The final action occurs in the Detection buffer if none of the above take place. Then the instruction and data block address is placed according to LRU in the Detection buffer with size 1.

The instruction with an associated size that is evicted from the Detection buffer is placed in the IS cache. This cache is a 16 entry fully associative cache and the main purpose of this cache is to save the instruction and size pair until the next time it occurs. The look up in this cache is performed in parallel with the look up in the ordinary data cache and the Detection buffer. If an instruction is found, the size and first data address is transferred to the Prefetch unit. After the instruction is found in the IS cache it is evicted from here, since we want it to be reevaluated. Therefore the same instruction can not be present in both the Detection buffer and the IS cache at the same time.

The reason for having a Prefetch unit is the possibility to flush the cache if there are very long IS sequences present. If, for example, there is an IS sequence of size 1024 data blocks it will flush the cache completely if all is prefetched at once. The Prefetch unit will only prefetch small amounts of an IS sequence at a time to the data cache. The resulting miss rate reduction is displayed in Figure 15, with the calculated result for each benchmark.

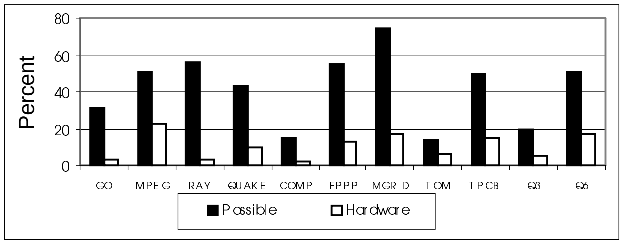


Figure 15: DSSP miss rate reduction in percent compared to the calculated miss rate reduction.

As we can see the actual miss rate reduction is very poor compared to the ideal case. To understand the limitations we analyze the different parts in the hardware implementation for the TPC-B benchmark. The first question to ask is whether the Detection buffer is able to detect all the possible IS sequences. In the diagram, Figure 16, we show the amount (in percent) a certain size of the Detection buffer is able to find.

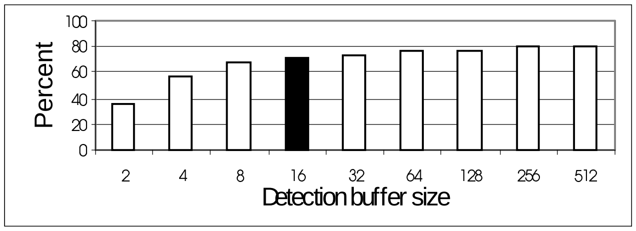


Figure 16: Detected IS sequences in percent of the possible IS sequences.

With a Detection buffer of size 16 it is possible to detect more than 70% of all IS sequences. The conclution is that the Detection buffer is working properly. The next unit to look at is the IS cache and the percentage of number of IS sequences prefetch out of the number of possible ones, Figure 17.

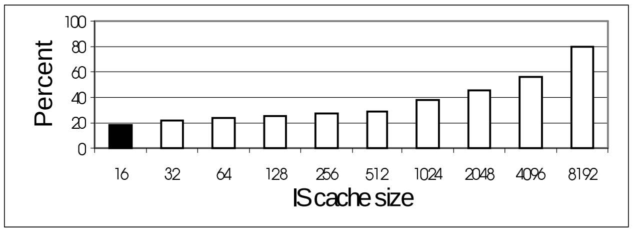


Figure 17: Exploited IS sequences in percent of the possible IS sequences.

The IS cache must be very large to be able to prefetch more than 70% of the possible IS sequences. A size of 8192 entries in a fully associative cache is not practical and therefore we need to know the cause of this, to be able to work around it. First the number of instructions to tag in the TPC-B benchmark is 2584, which are not going to fit in a 16 entry cache. However the largest contribution to poor performance is probably the fact that we detect many IS sequences that are only used once. An IS sequence that is only used once is no use putting into the IS cache, since it will not reappear.

Even though the performance of DSSP is not as high as expected, an important characteristic with this method is that 83% of all prefetch blocks are actually used. Therefore we believe that this method has great potential.

5 Discussion and related work

The major problem for exploiting the characteristics of IS locality shown above are twofold; a mechanism must be provided to detect the behavior (IS of a certain length), and the behavior must be stored in an effective way. The former can be pursued by similar mechanisms as used in our measurements, since we have shown the instructions to be relatively stable in their behavior for IS locality. However, tagging data blocks according to their recent behavior requires additional tags in the cache hierarchy and in the memory. Tagging instructions is potentially less demanding since the number of load/store instructions are far less than the amount of data on most commercial applications. Even though the number of instructions involved in IS sequences are less than for data, they are in the order of several thousands. However, since the results point out a substantial potential for improved performance that can be combined with other prefetching techniques, we are currently looking into mechanisms that will attack the identified obstacles.

In an article by Antonio Gonzales et al. [5] one reason for poor performance in data caches is identified. There is a trade-off between exploiting spatial and temporal locality. A cache block needs to be wide for the spatial locality, but if the references only experience temporal locality a significant pollution of the cache is introduced. A solution to this has been proposed with the split cache [5]. The split cache is a cache where there are two parts, one part for temporal references and one part for spatial references. However, the trade-off problem still exists, it is only transformed to deciding the sizes of the spatial and temporal cache. Let us assume that we use our mechanism in Section 2.2 to decide where to put blocks in the split cache. In Figure 4 we can conclude that the operating system for TPC-B will only be allowed in the spatial part of the split cache. Hence, the OS will experience a much smaller cache than the application.

6 Conclusion

This paper explores the potential of spatial locality in a wide range of applications. To be able to analyze locality we have developed a novel framework for measuring locality. One of the central concepts of our framework is *Immediate Spatial Locality* (IS). IS converges the characteristics of *closeness in time* and *closeness in space* of block accesses, which are the fundamentals of spatial locality that can be exploited by cache memories.

Our experimental infrastructure consists of a program driven simulator and a set of 11 applications, including emerging applications such as OLTP, DSS queries, graphics and a 3-D game. We also include the operating systems for those applications that make heavy use of system code.

Our experimental results show there is potential to significantly reduce the miss rate in a cache. The fraction of blocks that participate in IS sequences is in general more than 50%, and for some applications more than 90%. Interestingly, for our broad set of applications including commercial application classes, we show that the majority of the IS sequences are each generated by more than one instruction, meaning that small, simple loops are not the dominating source for the spatial locality we can exploit. For those applications that interact substantially with the OS, we have also compared the behavior of references generated by different software components; application code, OS kernel and system libraries. Our results show that, not only do these different parts of the software behave differently during the execution of an application, but also the behavior of each of these software parts varies heavily across different applications. All these results tend to lead to a high degree of useless prefetches for hardware-based prefetching techniques.

However, the results also show that the spatial locality behavior of the data that are generated by the same load or store instruction is relatively consistent. This means that if the data reference of an instruction belongs to an IS sequence of a certain length, there is a high probability that the next time the same instruction request data, that data reference will also belong to an IS sequence of the same length. Taking into account the overhead of detecting such behaviors, we show that the miss rate would be reduced by up to 75% for a 16KB cache had this behavior been fully exploited. In a straightforward attempt to exploit our findings, we explored the gains from a novel device that make use of IS for prefetching. This first attempt was able to reduce the miss rate by up to 23% with few useless prefetches. In addition, we also explore the reasons for the incompleteness of our prefetching algorithm, pointing out which deficiencies to be targeted in future studies.

This paper presents basic knowledge on the nature of spatial locality for many important application classes. The results show that there is indeed substantial performance improvements to gain from exploiting spatial locality. Even though the basic characteristics of the spatial locality differ heavily across applications, there are some static behavior of single instructions that indicate a potential for hardware prefetching mechanisms to reduce the miss rate with few useless prefetches for a range of applications.

Acknowledgments

Martin Kämpe is supported by grants from Swedish Research Council on Engineering Science (TFR) under contract number 211-1997-593 and equipment from Sun Microsystems. We would like to thank the following people at Chalmers University of Technology for help in different ways; Björn Andersson, Jonas Jalminger, Magnus Karlsson, Thomas Lundqvist, Jim Nilsson, Peter Rundberg, and Per Stenström. We would also like to thank Michael Koster and Anders Landin at Sun Microsystems for valuable comments.

References

[1] A. Agarwal, M. Horowitz, and J. Hennessy, "An Analytical Cache Model," *ACM Transactions on Computer Systems,*Volume 7, Number 2, pp. 184-215, May 1989.

[2] L. A. Barroso, K. Gharachorloo and E. Bugnion, "Memory System Characterization of Commercial Workloads," In *Proc of the 25th International Symposium on Computer Architecture,* pp. 3-14, June 1998.

[3] T-F. Chen and J-L. Baer, "Effective Hardware-Based Data Prefetching for High-Performance Processors," *IEEE Transactions on Computers,* Volume 44, Number 5, pp. 609-623, May 1995.

[4] Z. Cvetanovic and D. Bhandarkar, "Characterization of Alpha AXP performance using TP and SPEC workloads," In *Proc. the 21st ISCA,* pp. 60-70, 1994.

[5] A. Gonzales, C. Aliagas and M. Valero, "A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality," In *Proc. from International Conference on SUPERCOMPUTING,* pp. 338-347, July 1995.

[6] J. Hennessy and D. Patterson, "Computer Architecture A Quantitative Approach," Second Edition, Morgan Kaufman Publisher Inc., pp. 38, 1990.

[7] ID software. http://www.idsoftware.com. *Quake,* October 1997.

[8] P. Magnusson, F. Dahlgren, H. Grahn, M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, P. Stenström, and B. Werner, "SimICS/sun4m: A VIRTUAL WORKSTATION," In *Proceedings of USENIX'98,* pp. 119-130, June 1998..

[9] MPEG Organization. http://www.mpeg.org. *MPEG2 decoder,* January 1996.

[10] M. Rosenblum, E Bugnion, S. A. Herrod, E. Witchel, and A. Gupta, "The Impact of Architectural Trends on Operating System Performance," In *Proceedings of SOSP-15,* pp. 285-298, December 1995..

[11] P. Stenström, E. Hagersten, D.J. Lilja, M. Martonosi, and M. Venugopal, "Trends in Shared Memory Multiprocessing," *IEEE Computer,* Volume 30, Number 12, pp. 44-50, December 1997.

[12] Standard Performance Evaluation Corporation. http://www.spec.org. *SPEC 95.* August 1995.

[13] O. Temam and N. Drach, "Software Assistance for Data caches," In *Proc. from HPCA,* pp. 154-163, January 1995.

[14] Transaction Processing Performance Council, "TPC Benchmark D (Decision Support) Standard Specification Revision 1.1, December 1995.

[15] Transaction Processing Performance Council, "TPC Benchmark B (Online Transaction Processing) Standard Specification, 1990.

[16] TcX AB, Detron HB and Monty Program KB, "MySQL v3.22 Reference Manual," September 1998.

[17] S. Cameron Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," In *Proceedings of the 22nd ISCA,* pp. 24-36, June 1995.