International Conference on Computer Systems and Technologies - *CompSysTech'2003*

Selective Stack Prefetch Method

Radi Romansky, Yordan Lazarov

***Abstract**: In this paper is described novel approach for prefetching of stack references. In proposed method the decision which blocks to be prefetched is based on the context of current procedure. The results clearly point that we can effectively prefetch the stack memory references, we have around 50.82% of miss rate reduction compared to the standard N block lookahead (OBL) methods as meanwhile the accuracy is on average 16.62%.*

***Key words**: prefetching, stack data.*

1. Introduction

The basic prefetch mechanism is known as One Block Lookahead (OBL)[5]. In this method when there is access to block N in cache is started prefetch of N+1 block. Later in [2] is described method that extends the idea of OBL with adding small associative buffer where are prefetched N blocks instead of just one. The published results point one good improvement versus the OBL method.

The common in all these methods is the fact that they do not distinguish the different type of the data they try to prefetch and thus they can not exploit the specific characteristics of them, like the clear spatial locality of stack data.

In [3] is studied stack cache that employed method of tracking invocations/returns of procedures. The method of precise tracking of procedure frame allows the requests for prefetching to be issued early enough to hide more of the missed data latency. The showed results and conclusions point that even there is good hits ratio, around 99,80% for 4KB stack cache, also there are some drawbacks. As the average usage of the prefetched blocks are relatively small, only 52% and with proposed decoupled cache architecture there is problem with low utilization of both caches, data and stack.

The results from [3] led us further to try to overcome pointed problems. First we chose to be used unified cache for L1 global and stack data, but we preserved and employed the same mechanism for tracking the procedure's frame as proposed in [4].

And second we chose mechanism similar to classic OBL method for prefetching stack data, but instead of prefetch one or N blocks starting from missed address, we use the context of the procedure to identify where to begin prefetch from.

2. Selective Stack Prefetch Method

2.1. Goals

Our goal is to be prefetch exactly on time the slice from the stack assigned to current procedure at the time of miss on stack data without evicting the usable blocks from L1 data cache.

2.2. Limitations and simplifications

In context of this work we will use term as 'procedure context' that describes the information that is relatively constant during the execution of a procedure, i.e. the address of control branch and the size of procedure's frame.

We put constraint that during execution of the procedure it has access only to its frame and also that at the moment of call/return the two pointers BP and SP are filled in advance with correct values. This is achieved with help of compiler.

2.3. A case of Selective Stack Prefetching (SSP)

We employ the simple possible mechanism for prefetching, OBL[2,5], but instead of prefetching one constant number of blocks starting from missed address we use the stored information for context of current procedure to prefetch exactly portion of procedure frame.

In this way it is minimized the traffic to L2 cache in case the size of procedure's frame is less than the size of prefetch buffers. And also is improved the accuracy of prefetched bocks versus OBL method, when we have misses to data that are not at the beginning of the procedure's frame.

On next figure 2.1 is showed one example implementation of proposed selective stack prefetch method(SSP) and its integration.

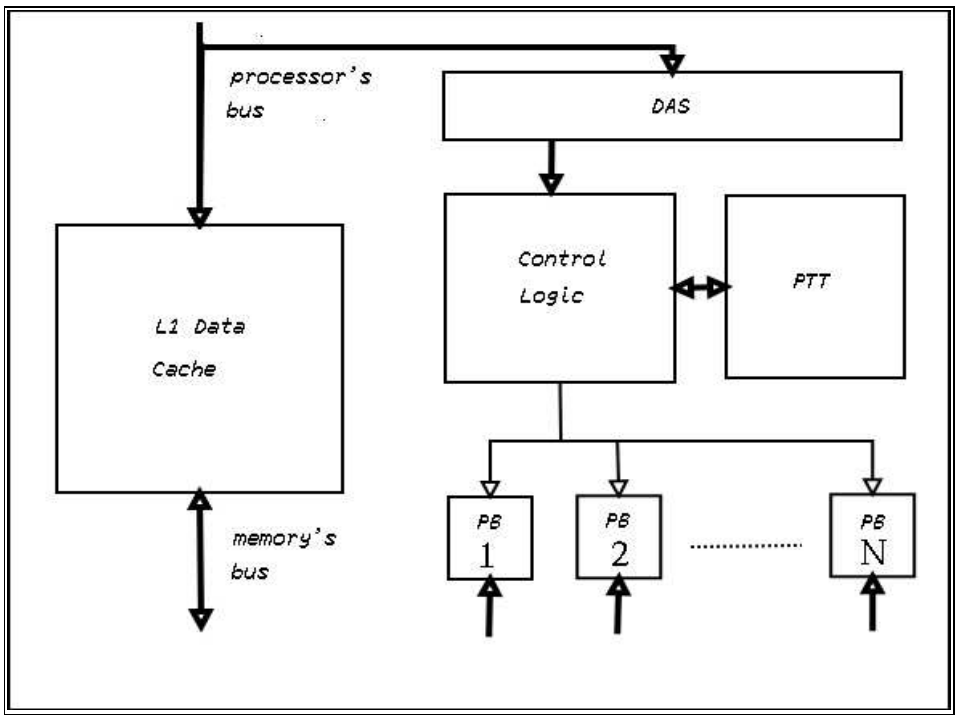

Figure 2.1

It contents cache, scheme for tracking procedure transitions (PTT), control logic, a set of prefetch buffers (PB) and scheme for differentiating the memory reference stream on stack one and global one.

The differentiation address stream scheme (DAS) is used for classification of the memory reference stream on stack one and global one. It uses the address of top of the stack(TOS) to identify the type of references, as if looked address is bigger than TOS it is accepted that it is stack reference. If it is smaller the reference is classified as reference to global data. This simple method for classification is possible because we put constraint that one procedure can make references to data residing only on its procedure's frame.

The procedure tracking table (PTT) is organized as two arrays, procedure context table (PCT) and procedure probability table (PPT), both are with FIFO organizations, as is proposed in [4]. The prefetch buffers (PB) are with FIFO organization.

2.4. Algorithm of work

The SSP method works in following way, when there is reference to memory the address is sent to L1 data cache for processing. In case of hit, data is returned to processor when the action is read, or the data is stored in cache in case of write request. In case of miss, the referenced data is brought from L2 data cache and are performed necessary actions of its preserving in L1 data cache. Simultaneously to processing of memory reference in data cache it is also sent to DAS for classification on type of reference, stack or global.

In case of miss in L1 data cache and if memory reference is classified as reference to stack the address of data is sent to control logic of SSP for further processing. First the address of current procedure context is looked in PTT, if there is hit the stored information for current procedure context is used in decision which blocks to be prefetched. As if the missed address is in context of procedure frame and the address is in range of first N blocks of procedure's frame, where N is the size of prefetch buffers(PB) in blocks, then the first N blocks from current procedure frame are prefetched. When the missed address is outside the first N blocks of procedure's frame, then N blocks from missed address are prefetched. In the case of miss in PTT it is prefetched first N blocks beginning from the missed address.

To be achieved better results is used PTT for prediction of next possible procedure context and if such transition can be found the first N blocks of its frame are prefetched.

When in decoding stage of processor's pipeline is encountered instruction for invocation of new procedure, the address of this branch together with address of SP and BP are sent to PTT for processing where the probability of this transition is updated and the context of procedure is preserved.

3. Testing and Simulations

We use SimpleScalar framework[1] for simulating the proposed SSP. We use the benchmarks from SPECint95 set as test applications. It is simulated the first 50M instructions of benchmarks. In the base configuration the L1 cache is with size of 128 bytes and block size of 32 bytes. Also the chosen cache configuration is with write-back, fetch on write policy and LRU replacement strategy.

On next figure 3.1 are given the results of simulating the proposed prefetch method in case we employ prefetch scheme with 8 prefetch buffers each with size of 8 blocks. It is not employed the mechanism for prefetching the next possible procedure frame. As it is pointed in [4] about the size of frame and scattered usage of blocks for *gcc* and *li* lead to bad results, we have around 92% reduction of misses for *gcc* and only 85% for *li*. Even with these results, we will see on next figures, that the proposed method SSP outperforms the standard OBL method.

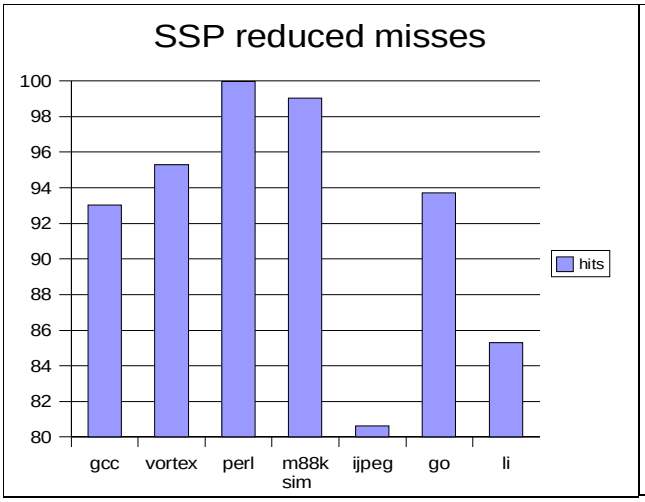

Figure 3.1

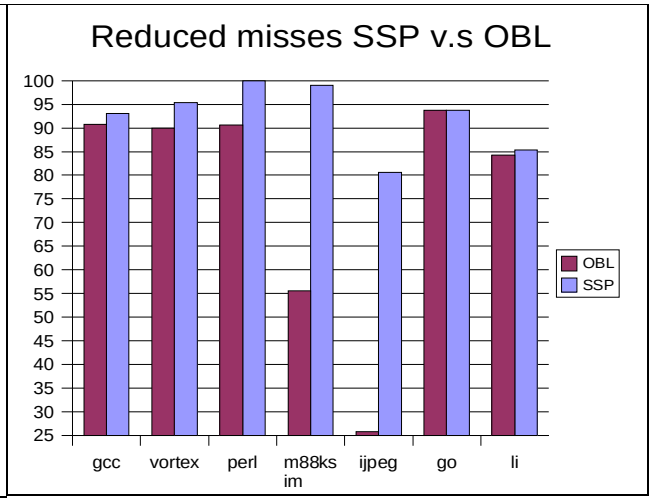

Figure 3.2

For *ijpeg* we have very little reduction, unfortunately this test has very little misses to stack memory only 62, so its results are not clean. For the rest of tested programs we

have one very good reduction of misses to L1 data cache.

On figure 3.2 are given the results for reduced misses for OBL and SSP method. The showed results point that we have one acceptable level of reducing the L1 data cache misses, especially for *m88ksym*, *ijpeg* and *perl.* We can see that except *go* all others tests have good improvement versus standard OBL methods. For *go* we can argue that the misses are the least compared to the others benchmarks, so in our opinion the result for *go* is not representative.

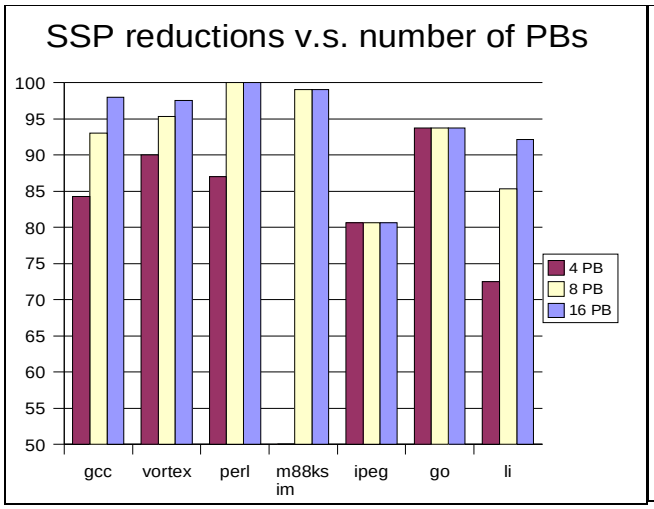

Figure 3.3

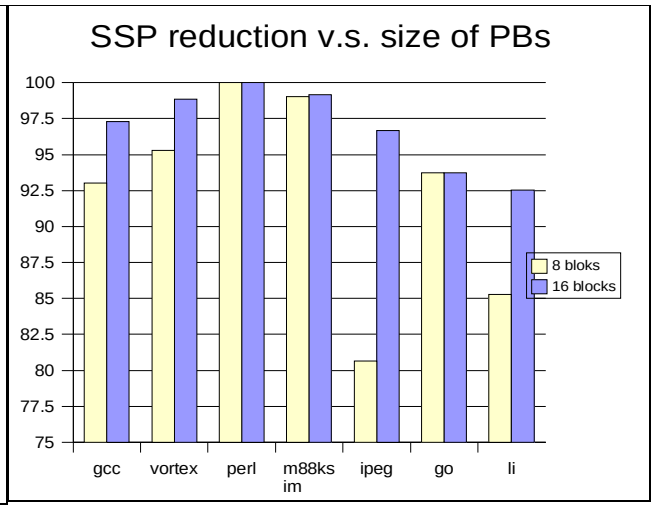

Figure 3.4

On figure 3.3 are given the variations of the reduced misses for SSP in case of changing the number of prefetch buffers(PB). As can be expected more prefetch buffers give us better reduction of L1 stack misses. Exception is only *go* and *ijpeg* benchmarks, where we do not have any improvement.

On figure 3.4 is given the average improvement in case we change the size of prefetch buffers. In base configurations it is 8 blocks. Again we have improvement for *gcc*, *vortex* and *li* and no for the rest of tests. If we return to results published in [4] we can see that for these tests we have references that are outside the first 8 blocks of procedure's frame, thus the prefetch buffers with deeper size give us better coverage of missed stack references.

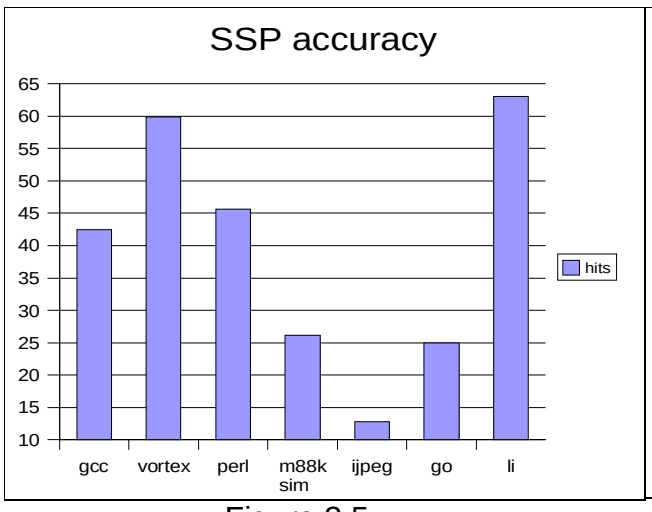

Figure 3.5

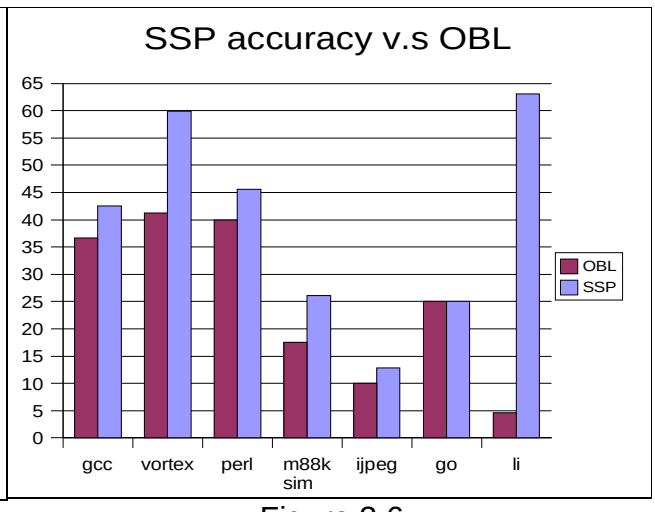

Figure 3.6

On figures 3.5 and 3.6 are given the results of testing the accuracy of SSP and OBL, as the accuracy is defined as relation of numbers of used prefetched blocks to all prefetched blocks. The showed results on figure 3.5 clearly point one very good level of accuracy for four tests, *gcc*, *vortex*, *perl* and *li*. For the rest of tested programs the ratio is on average of 15%-25%. Again for these tests the low rate of misses to stack memory explain the bad accuracy.

In both cases we have very good improvement of accuracy compared to standard OBL methods, on average 16,62%, as this can be seen on figure 3.6.

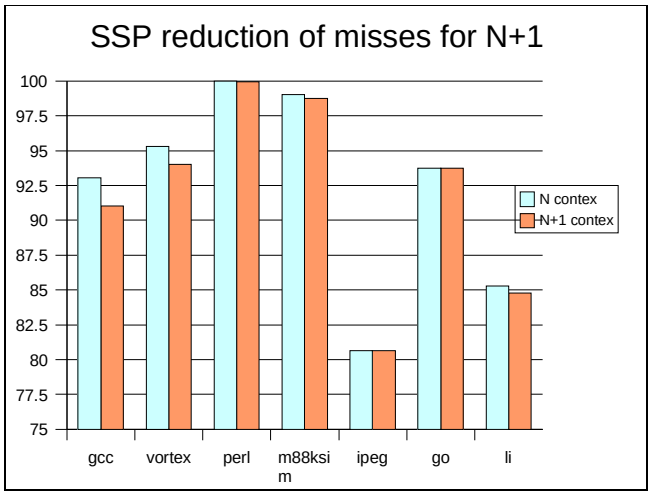

Figure 3.7

On next figure 3.7 are showed the results when is taken in account the prefetch of N+1 procedure, as it is predicted by PTT, versus the case where we prefetch only current procedure.

As we see we have no improvement for all tests. These results clearly point that there is no correlation between misses on different procedure.

4. Conclusion

In this study is proposed a novel approach of prefetching stack data, based on context of current procedure. The showed results point that with proposed method we have very good improvement comparing to standard OBL methods. We have around 50.82% percentage of decrease of stack misses as meanwhile we have very good ratio of accuracy, on average 16.62% for proposed SSP method. The showed results also clearly point that there is not correlation between stack misses on different procedures' frames.

Although the ratio of eliminated stack misses and the accuracy of proposed SSP method is good, further investigations are necessary for minimizing the traffic to L2 cache.

References

1. D. Burger and T. M. Austin. "The SimpleScalar Tool Set, Version 2.0," Computer Sciences Department Technical Report, No. 1342, Univ. of Wisconsin, June 1997.
2. N.P. Jouppi. "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-associative Cache and Prefetch Buffers". In Proc. of 17th Int. Symp. on Comp. Architecture, pp.364-373, May 1990.
3. R. Romansky, Y. Lazarov. "Stack Cache Memory", given for publication.
4. R. Romansky, Y. Lazarov. "A method for tracking procedure's invocations", given for publication.
5. A.J. Smith. "Cache Memories". Computing Surveys, 14(3):473-530, Sept. 1982.

1. Radi Romansky, PhD, Department of Computer Systems, Technical University-Sofia, Phone: +0359 965-25-24,E-mail: rrom@tu-sofia.acad.bg
2. Yordan Lazarov, M.Sc, BTC, Phone: +0359 276-14-25, E-mail: dancho@mbox.infotel.bg.

Contents