Predicting and Prefetching TLB Entries from Irregular Access Streams
by Kev Kitchens

Advisor: Dr. Calvin Lin
December 8, 2013
Abstract

This thesis describes the design and evaluation the Irregular Translation Buffer (ITB), which is a prefetcher for translation lookaside buffer (TLB) entries that is based on the Irregular Stream Buffer (ISB) introduced by Jain and Lin [1]. The main idea to use the ISB’s improvements in prefetching cache lines from irregular access streams and use them to prefetch entries from irregular access streams for the translation lookaside buffer (TLB). The ISB is particularly good at tracking irregular access streams, which are sequences of temporally correlated memory reads. The ITB attempts to translate the ISB’s strengths from the realm of prefetching cache lines to that of prefetching TLB entries.

I evaluate the ITB using the MARSS full x86 system simulator and several dual-core SPECint and SPECfp 2006 benchmarks to evaluate the ITB. I was unable to show that the ITB causes these workloads to exhibit a speedup due to a lack of prefetch candidates that are not already found in the TLB. Based in this result, I discuss possible workloads that could be used to better evaluate the ITB.

1. Introduction

In much the same way that memory caches attempt to hide the long latencies incurred in accessing main memory, the translation lookaside buffer (TLB) was developed to hide the latencies of translating memory addresses from the virtual address space to the physical address space by caching these translations. This caching is particularly important since translating a single address requires multiple accesses to main memory, and the translation process must finish before the data at any virtual address can be accessed, regardless of whether the data itself is cached or not.

Because good TLB performance is so crucial for good memory performance, two primary approaches have been used to reduce the latency and the number of TLB misses. First, the TLB has been enlarged to store more entries and to allow the use of superpages, memory page larger in size than a typical memory page, that expand the translational reach of a single TLB entry. On an x86 system,
regular pages are 4KB in size, and superpages are 2MB in size. Unfortunately, these changes cannot reduce TLB misses when the size of a workload’s data set exceeds the reach of the TLB, which is the maximum amount of physical memory that can be mapped by all of the entries in the TLB. Second, caching schemes and prefetchers have been proposed that would allow the elision of accesses that would otherwise miss the TLB. Unfortunately, the caching schemes also do not reduce TLB misses, and existing caching schemes have a hard time efficiently tracking irregular TLB access streams, which are sequences of temporally correlated accesses that are not spatially correlated.

Increasing the hit rate of the TLB is crucial for many workloads, particularly server applications that tend to make a lot of memory accesses. Even if the cache prefetcher does a good job of improving the cache hit rate to make cache misses rare, a TLB with a high miss rate will act as a bottleneck, since all accesses to the cache can only happen after the address translation process has either hit in the TLB or conducted an expensive and high latency page table walk. Modern data sets are growing rapidly, so it is crucial that the TLB have a good prefetcher to reduce the miss rate as much as possible since these data sets are frequently much larger than the reach of the TLB. Unfortunately, current TLB prefetchers have major flaws in tracking irregular TLB access streams efficiently that reduce their ability to increase the hit rate of the TLB.

I have designed a prefetcher called the Irregular Translation Buffer (ITB) that improves upon existing prefetchers when dealing with PC-localized, irregular TLB access streams. PC-localization is a technique to improve prefetcher performance that segregates streams based on the program counter of the instruction issuing the temporally correlated memory reads in the stream. This system is heavily based on the Irregular Stream Buffer (ISB) prefetcher designed by Jain and Lin [1]. Like the ISB, the ITB is designed to handle irregular access streams well and fix many of the issues present in current systems targeting irregular access streams. The ITB takes the ISB’s key contribution of the structural
address space that allows temporally correlated addresses to be spatially correlated to make it easier to efficiently track and prefetch addresses from irregular access streams and applies it to the realm of TLB prefetching. Instead of using this innovation to prefetch cache lines by predicting physical addresses, the ITB uses it to prefetch TLB entries by predicting virtual addresses.

I evaluate the ITB using some of the SPEC 2006 benchmarks and a full-system x86 simulator that includes the ITB in order to determine how well it improves memory performance by increasing the TLB’s hit rate. Unfortunately, I find that the results of the simulations do not allow me to draw a firm conclusion about how the ITB affects memory performance. However, I discuss what I believe is the reason the results are inconclusive and suggest alternative benchmarks that could be run in the simulator to produce results that would allow me to evaluate the ITB.

This is the organization of my thesis: Section 2 place my work in the context of prior work on improving TLB performance. Section 3 discusses the ISB itself in detail to provide background for understanding the ITB. Section 4 describes the changes I made to the ISB to make it into the ITB. Section 5 evaluates my work, then I conclude.

2. Related Work

To provide the proper context for my work, I will now discuss the designs and drawbacks of other approaches to improving TLB performance, first focusing on improvements to the TLB itself, and then concentrating on prior work in TLB prefetching.

Superpages

Current page tables are implemented as a radix tree as show in Figure 1, where each level of the tree uses some portion of the address as an index to an entry that gives the address of the next level in the tree. The final level of the tree contains the actual addresses of the memory pages. Superpages were created to allow the process of walking this tree to terminate early by enlarging the size of a memory
page and placing the entry containing its address higher up the tree as show in Figure 2. However, this approach can be difficult to utilize in practice because it requires the operating to do expensive memory defragmentation procedures that can lead to a lot of copying to consolidate data together into chunks the size of super pages [2-3]. Also, it is generally very difficult for the operating system to dynamically identify large, properly aligned chunks of memory that can be consolidated into a superpage, so most uses of superpages require a programmer to explicitly signal the operating system to use them.

**Lookup Caching**

Since many entries in the higher levels of the page table structures are used repeatedly in the process of looking up TLB entries, Barr, et al. [4] store higher level entries in high speed caches on the MMU itself to avoid accessing the slower processor caches during a page table walk. This approach does have have the ability to shorten the time it takes to service a TLB miss, but it ultimately does not have a prefetcher’s power to reduce the number of TLB misses. Also, the cache only stores some parts of the page table walks, so it still requires the TLB to access main memory to retrieve an entry.

**Victim Caching**

Memory accesses often exhibit temporal locality, so Kandiraju and Sivasubramaniam [5] use a victim cache to store recently evicted TLB entries reside. This cache is checked as a last resort when
accesses miss the TLB altogether. This approach suffers the same limitations as the lookup caching
approach, mainly that it has no fundamental predictive power. Also, this approach fails for streams that
exhibit some temporal locality but with reuse distances long enough that the entries are evicted from the
victim cache prior to their reuse.

**Stride Prefetching**

As is the case of traditional cache prefetching, TLB entries can be prefetched using stride-based
approaches. Several approaches have been used to predict future TLB accesses with delta correlations,
which are fixed differences between consecutive addresses in a stream. One such example is the
Arbitrary Stride Prefetcher originally introduced by Chen and Baer [6] and adapted for TLB prefetching
by Kandiraju and Sivasubramaniam [5]. However, stride prefetchers fundamentally cannot predict TLB
accesses from memory streams that do not fit a regular delta correlation pattern.

**Irregular Prefetching**

In an attempt to handle access streams that are irregular spacially but temporally correlated,
Kandiraju and Sivasubramaniam [5] created the Markov Prefetcher (MP) where TLB access streams are
stored in a table structure to allow prefetch candidates to be acquired. However, this table structure
stores data inefficiently, with shorter streams leaving parts of the allocated block unused and long
streams requiring redundant information to be stored. Nesbit and Smith’s Global History Buffer (GHB)
[7] attempts to restructure the storage of this history to reduce the amount of on-chip storage needed to
track irregular access streams, but it is unfeasible to use with PC-localization since its predictions
requires multiple, serial table lookups that can even require accessing the prefetcher’s off-chip storage in
main memory.
3. Background

To understand the ITB, I now discuss the ISB’s design, since the ITB is heavily based on the ISB. This description is inspired by Jain and Lin’s ISB paper [1]. At its core, the ISB attempts to take the complex problem of prefetching from irregular access streams and transform it into the simple problem of prefetching from a regular access stream. To accomplish this, the ISB introduces the notion of a structural address space. In this space, structural addresses map to physical addresses, somewhat similar to the way virtual addresses map to physical addresses in the page tables. In the ISB, temporally correlated physical addresses from an access stream are mapped to consecutive structural addresses. Figure 3 shows how an irregular access stream in the physical address space maps to a regular stream in the structural address space. This mapping allows temporally correlated addresses from an access stream to be spatially correlated in the structural address space, facilitating easy prediction.

**TLB Synchronization**

To reduce the storage needs of the ISB, only metadata for cache lines in pages mapped by entries present in the TLB is stored on-chip by the ISB. Since an access to a cache line in a page mapped by an entry not in the TLB would trigger a page walk, the ISB will synchronize it’s on chip metadata with a larger off-chip storage in main memory while the TLB miss is serviced. It will retrieve the metadata for the page whose entry is being retrieved and flush the metadata for the page whose entry is being replaced by the inserted entry. Since the page walk process is slow, the synchronization latency can be
effectively hidden within it. By hiding the synchronization latency and placing it off the critical path, the on-chip metadata can be smaller than in other irregular access stream prefetchers without sacrificing any performance.

**Training Unit**

The training unit tracks the PCs of load instructions it observes from the overall memory access stream and the last loaded physical address from the load instruction at each PC. It also assigns consecutive structural addresses to temporally correlated addresses in each tracked access stream.

**Address Mapping Caches (AMCs)**

The mappings between the structural and physical address spaces are maintained by a pair of on-chip caches. The Physical-to-Structural AMC (PS-AMC) is indexed by physical addresses and stores the mappings from the physical address space to the virtual address space, and the Structural-to-Physical AMC (SP-AMC) is indexed by structural addresses and stores the mappings from the structural address space to the physical address space.

To reduce the on-chip storage necessary to hold these mappings, the top 42 bits of the physical address are replaced by a 7-bit index in the TLB. This is possible since only pages with an entry in the TLB have their ISB metadata stored on-chip. This index is concatenated with the 6-bit offset into the page that specifies an individual 64-byte cache line. In a similar fashion, structural page indices can be reduced to 7 bits and concatenated with a 6-bit offset into each structural page. These reduced mappings are stored in an on-chip CAM and updated during TLB synchronization. The 13-bit addresses are used internally by the ISB, being expanded upon when a full address is needed for TLB synchronization or when a prefetch is issued.

The AMCs are set-associative with 32-byte lines, and each line maps 16 consecutive addresses. These two-bit mappings contain the aforementioned 13-bit reduced address, a valid bit, and a 2-bit
confidence counter used for training. 8K of AMC entries are needed to fully cover the pages mapped by the entries in a 128 entry TLB.

**Stream Predictor**

The stream predictor tracks and manages streams within the structural address space. It stores information about each stream, including the first structural address in the stream, the length of the stream, and the current prefetch look-ahead. It makes predictions like a traditional stream buffer, also allowing different prefetch degrees and look-ahead distances to be configured [8].

**Training**

When a pair of temporally correlated physical addresses (A, B) is observed in the access stream by the training unit, it uses existing metadata to assign them to consecutive virtual addresses. The training unit queries the PS-AMC to see if A and B already have structural addresses assigned. If they already have assigned consecutive structural addresses, B’s confidence counter is incremented; if the assigned addresses are not consecutive, B’s confidence counter is decremented. If B’s confidence reaches 0, it is reassigned to the structural address consecutive to A’s structural address. However, if A did not have a structural address previously assigned, A and B are assigned consecutive structural addresses and the AMCs are updated accordingly. Figure 4 shows an example of the training process. To allow consecutive structural addresses to be easily assigned, the ISB allocated blocks of structural addresses in chunks of 256, using a 64-bit counter to track and assign unallocated structural addresses.
**Prediction**

When a previously observed physical address is passed to the ISB, it first looks up the corresponding structural address in the PS-AMC. Then, the next $k$ consecutive structural addresses are predicted, where $k$ is the configured degree of prefetch. Then, the predicted structural addresses are looked up in parallel in the SP-AMC to get their corresponding physical addresses. These physical addresses are then prefetched. Figure 5 shows an example of the prediction process. This prediction process is simple and efficient since it is essentially a simple stream prediction.

**4. My Solution**

There are three major challenges in adapting the ISB to work on the virtual address space as the ITB. First, I had to deal with the fact that virtual addresses are prefetched at a 4KB page granularity, while physical addresses are prefetched at a finer 64-byte cache line granularity and can be compressed to 13 bits by exploiting the fact that most of the upper bits of each physical address are already stored in the TLB. Next, I had to change the system to account for the fact that the virtual address space changes on every context switch, while the physical address space is permanent. Also, I had to figure out how to modify the ISB to account for the removal of the upper level TLB synchronization mechanism. The remainder of this section explains how I dealt with these three issues.

In the ISB, the addresses can be shrunk down because the TLB already stores physical address information, so the ISB can use an index into the TLB to recreate full physical addresses. Since the ITB
doesn’t have an upper level TLB, I instead opt to store the full page number of each virtual page I track. This requires 36 bits per virtual address instead of the previous 13 bits per physical address. The addressing system remains otherwise unchanged.

Since the virtual address space changes on every context switch, information previously gathered by the ITB may no longer be valid when the virtual address space changes. This is compounded by the fact that the compiler and operating system will often reuse the safe virtual addresses for things like global variable storage and the heap in multiple virtual address spaces for simplicity. To solve this issue, I currently reset the entire ITB when the TLB is reset during a context switch.

Finally, I had to determine a way to deal with the loss of the upper level TLB synchronization mechanism in deciding what metadata gets stored. I decided to simply allow insertions into the various metadata structures to overwrite older ones. Since the AMCs are designed as standard set-associative caches, old mappings between the virtual and physical address spaces will be evicted as the streams they are a part of fall out of disuse and the metadata becomes stale. I also doubled the size of the AMCs to reduce the chances that actives streams will conflict with each other. Since the whole ITB is reset after every context switch, this increase in size should allow the ITB to function well without a TLB to synchronize against.

5. Evaluation

I will now discuss how I evaluated my work and the results of this evaluation.

Methodology

I used MARSSx86, a cycle accurate full-system infrastructure that accurately models the entire CPU and memory subsystem [9], to simulate a dual-core, 64-bit, x86 system. I modified it to add the ITB prefetcher system, utilizing the existing page-table walking code to perform the prefetches. I added
a 64-entry table to track in progress prefetches and to place limits on how many prefetches can run in parallel, since a real system can only allow a finite number of prefetches to occur in parallel.

To assist with my evaluation, I also added a mechanism to track several statistics about the TLB prefetches. Various issues can cause a prefetch candidate not to be issued: it might already be in the TLB, the prefetch table might be full, or the candidate might already have been issued for prefetch. Also, not every issued prefetch completes successfully since a prefetch that causes an exception is considered to be invalid and is killed by the simulator. So, the simulator tracks how many prefetches are killed and how many are successfully inserted into the TLB. Finally, the simulator tracks how many of the successful prefetches get hit in the TLB before being evicted. This allows me to see how accurate a prefetcher’s predictions are.

I also modified the existing TLB to make it more realistic. The previous TLB organization used a fully associative design, which is generally unrealistic since it does not scale up well, requiring more chip area, power, and time to perform operations as the size increases. I modified it to be 4-way set associative since that is a common design for TLBs found in retail processors. I chose to leave the TLB size at 128 entries as was used in evaluating the ISB. I also implemented the ASP prefetcher mentioned in the Related Work section to have another TLB prefetcher to compare against.

**Workloads**

I used ten multi-programmed workloads where each is a combination of two existing benchmarks taken from SPECint2006 and SPECfp2006. These benchmarks ran simultaneously on the dual-core simulated system. I used SimPoint sampling to simulate 1 billion instructions directly following a fast-forward to the beginning of a SimPoint for each of the two benchmarks [10-11]. The results are averaged over five runs to account for differences caused by timer interrupts.
Results

I had originally planned to evaluate the ITB by comparing its performance with a baseline of no TLB prefetcher and the performance of the ASP prefetcher. I planned to examine changes in the TLB hit rate and changes in overall instructions per cycle (IPC) of the benchmarks. Additionally, I was going to examine how accurate the prefetches made by each prefetcher were since a very inaccurate prefetcher can fill the TLB with useless entries that cause useful entries to be evicted.

Unfortunately, I found that my results did not allow me to conduct these evaluations. I simulated the ITB with prefetch degrees of 1, 2, 4, 8, 12, and 16. In every case, I found that every prefetch candidate issued by the ITB was not issued because it was always already in the TLB. In analyzing this situation, I found that this result was caused by the benchmarks having very TLB-friendly access patterns so that the number of TLB misses is minimized. All of the benchmarks have a very high hit rate, >85%.

These results tend to imply that the ITB is very accurate, like the ISB, since the predictions it makes were consistently found in the TLB. A less accurate system would be predicting useless entries that had been evicted from the TLB due to lack of reuse. However, this is currently a conjecture based on extrapolation from these results and the evaluation of the ISB. In the next section, I discuss benchmarks that I believe would allow a full evaluation of the ITB as described previously.

6. Conclusion

In this thesis, I have attempted to show that the principles of the ISB can be generalized to the virtual address space by applying them in the ITB and evaluating this system. Like the ISB before it, the ITB should allow us to predict and prefect TLB entries from irregular access streams without requiring the storage of a prohibitive amount of on-chip metadata. The key idea of using a structural address space to make the irregular access streams appear regular appears to be extremely effective and may
revolutionize the field of prefetching. However, current results do not allow us to fully determine whether the ITB is an effective prefetcher or not.

**Future Work**

Since the SPEC benchmarks are synthetic and were designed almost a decade ago, I believe more modern, realistic server workloads could be used to evaluate the ITB. We have been working with getting server workloads from the CloudSuite benchmarks [13] running in the simulator to allow the ISB, ITB, and other memory management systems to be better evaluated. However, results from this work were unfortunately not available at the time of this writing.

Like the ISB, the ITB could benefit from extension to a two-level design that allows synchronization with large super pages in the structural address space as was proposed by Jain and Lin [1]. It is possible that both the ISB and ITB could synchronize together with this mechanism to create a unified memory prefetcher for both virtual and physical address spaces. Also, this mechanism could allow the ITB to synchronize with off-chip storage, increasing its reach and leading to even better results. The address space identifiers introduced by newer processors to reduce the overhead of context switches could be used as additional information in the overall system, since they allow the operating system to pierce the abstraction of virtual memory to provide more information to the processor about what memory space it is using.

Another possible improvement to the ITB would be to combine it with a stride prefetcher to create a hybrid system where the ITB handles irregular access streams and the stride prefetcher handles regular streams. This is also done by the ISB where the AMPM prefetcher [12] is used for regular streams. It might even be possible to adapt AMPM itself to predict TLB entries in the same way as the ISB was adapted into the ITB.
Acknowledgments

I owe a great deal of gratitude to Calvin Lin, my advisor for this thesis. I would not have been able to conduct my research and write this thesis without his invaluable guidance and original idea to adapt the ISB to prefetch for the TLB. I am also greatly indebted to Akanksha Jain, who mentored me. Her work on designing and coding the ISB allowed my research and this thesis to exist in the first place. I could not have done this work without her help. I would also like to thank Matthew Hicks for his helpful feedback on drafts of this thesis.
References


