# Selective Fill Data Cache

Rice University – ELEC525 Final Report
Anuj Dharia, Paul Rodriguez, Ryan Verret

**Abstract – Here we present an architecture for improving data cache miss rate. Our enhancement seeks to capture greater temporal locality than a standard cache by more efficiently using the area available. Frequently used data ought to remain in the cache, while infrequently used data ought not to be admitted into the cache. Policing entry into the cache leaves more room for useful data.**

**The Selective Fill Data Cache prevents rarely used data from entering the cache. The Cache Fill Policy keeps record of data blocks that exhibit little temporal locality. These data blocks must bypass the L1 cache before reaching the CPU. It is in the bypass path where we cache these values in a bypass buffer in order to decrease the penalty for mis-predictions.**

***Index Terms*** **– Selective Fill Data Cache, SFDC, bypass buffer, Cache Fill Policy Table, CFPT, data cache**

## I. INTRODUCTION

The gap between the performance of a superscalar processor and its memory subsystem is quickly increasing. For many years computer architects have focused their efforts on improving processor architecture and have created enhancements such as speculative and out-of-order execution. Architects have also increased the issue width to improve performance. It is important to realize that applications can only fully and effectively utilize these processor enhancements if the underlying memory subsystem can feed the processor with both instructions and data quickly enough.

Researchers are constantly striving to mitigate the effects of long memory latencies. Many modern processors implement techniques such as, "lockup-free caches, cache-conscious load scheduling, hardware and software prefetching, stream buffers, speculative loads and execution, multithreading, data value prediction, and instruction reuse [1]" to help reduce this latency.

*a. Motivation*

Expanding issue widths, aggressive software and hardware prefetching techniques, speculative execution, and a host of other modern techniques create a greater demand for data, straining caches more than previous generations of processors and creating more conflict and capacity misses. These cache misses are becoming more and more costly because of the hundreds of cycles they take to satisfy. Additionally, as superscalar clock frequencies continue to increase, data caches must adapt. Small, fast caches continue to shrink in size to maintain their single cycle latency, while larger, slower caches cause longer access latencies.

As access times take more cycles and caches grow smaller, their efficient

use becomes more important. New techniques are necessary in order to take full advantage of the entire cache and ensure that cycles are not wasted as a result of inefficient use of the cache.

*b. Hypothesis*

Increasing the line size or augmenting the associativity of a cache does not capture adequate spatial and temporal locality. If frequently used data is given priority in a cache while infrequently used data is prevented from filling it, the effectiveness of a cache can be increased. Data consistently evicted from the cache before a subsequent access ought not to enter if it is to evict useful data. This scheme, collectively known as a Selective Fill Data Cache (SFDC), should keep more frequently used data in the cache and reduce conflict and capacity misses caused by useful data being evicted by infrequently used data. Our implementation proposes two modifications to existing processors, both of which do no delay the critical path. The first is a Cache Fill Policy Table (CFPT) that records data which does not need to be cached. The other is a Bypass Buffer that is used to cache data that is not allowed in the data cache.

*c. Preliminary Findings*

Prior to the implementation of the SFDC, we closely examined cache miss and hit rates over different reuse distances for the 175.vpr SPEC2000 integer benchmark in Figure 1. For reuse distances from 1 to 10, the first two bars on the graph, the 8KB direct mapped cache sufficiently captures the temporal locality of the data. As reuse distances grow, however, the miss rates grow substantially. Initially, we projected the SFDC to eliminate half the misses for reuse distances of up to 10,000, the first five bars on the graph.

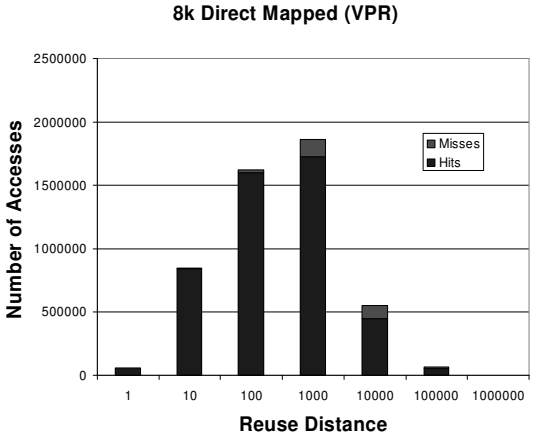


**Figure 1:** Reuse Distance for 8KB Direct Mapped Cache

The remainder of the paper is organized as follows. Section II takes a closer look at the architectural specifics of the proposed SFDC. Section III specifies the experimental methodology for the simulations. Section IV examines the results of these simulations. Section V presents conclusions, and Section VI suggests possibilities for future work.

## II. ARCHITECTURE

The architectural changes made to implement the SFDC are not overly complicated. Figure 2 shows the basic block diagram of our cache implementation, where changes include minor modifications to the L1 data cache, a direct mapped CFPT of tags and counters, and a two-way set associative bypass buffer. These alterations allow the cache to operate with a more sophisticated behavior in an attempt to reduce miss rate.

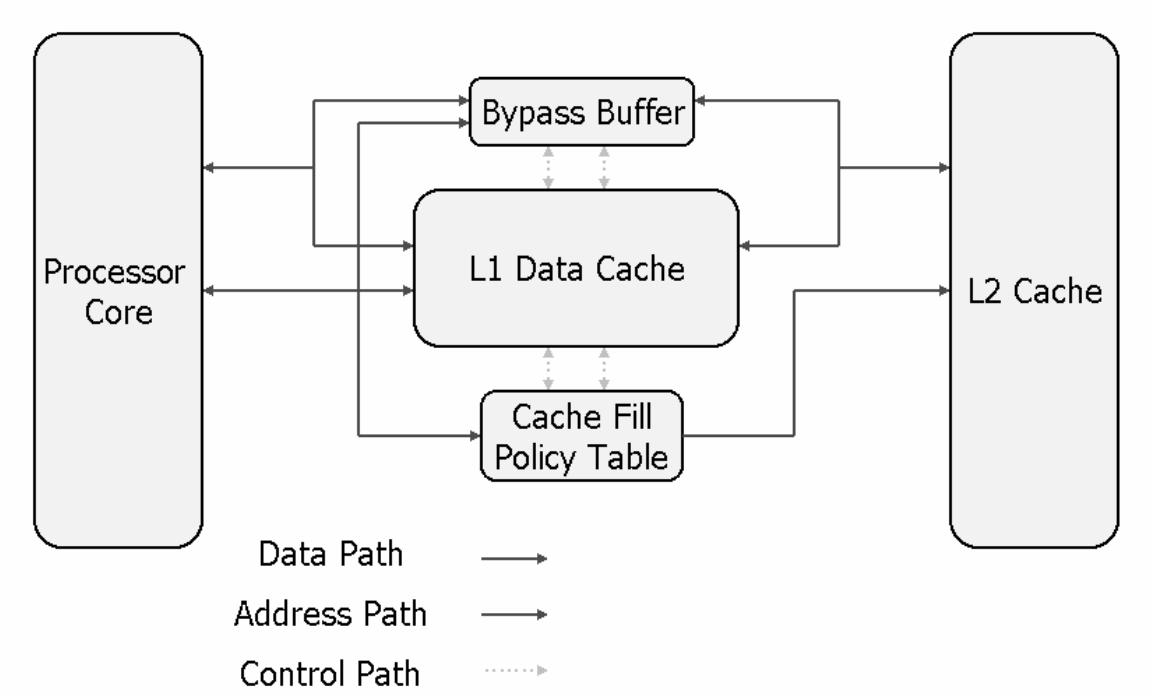


**Figure 2:** Selective Fill Data Cache Architecture

*a. L1 Data Cache*

The only necessary modification to the L1 data cache is the addition of a "used" bit to each block. When a new block enters the cache, its used bit is cleared. If the block is ever accessed again for a read or write, the used bit is set. This is a variation of the "dirty" bit and should not require increased design or verification time.

*b. Cache Fill Policy Table*

The CFPT is implemented as a direct mapped cache of tags. The number of entries in the fill policy table is the same as the number of sets in the cache, reducing complexity. This configuration gives each set the ability to keep out one block that should not reside in it. Additionally, each entry in the CFPT contains a two bit saturating counter used to profile access patterns.

When a block is evicted from the cache, the CFPT checks the status of the used bit. A set used bit indicates that the processor accessed the block after it entered the cache and the CFPT does nothing. If, however, the block is never accessed, the CFPT compares the tag of that block with the content of the corresponding entry in the table. When these tags differ, the CFPT retains the tag of recently evicted block in the appropriate entry, overwriting whatever was present before. The CFPT also clears the counter associated with that entry. If the tags are the same, the CFPT increments the entry's counter. Once in the table, tags remain there until replacement. This behavior determines the number of sequential occurrences in which a given address block is evicted from its appropriate set without having been used. A comparison of the counter and a threshold value determines whether or not a block with a given tag is allowed to enter the cache.

*c. Bypass Buffer*

For blocks that are kept out of the L1 data cache, a bypass path is provided from the L2 cache. In this path exists a small cache called a bypass buffer. This structure is intended to alleviate some of the costs of mispredicting the usage of a data block. Data blocks remain in the bypass buffer until replacement.

We performed initial tests to determine what sizes and configurations best suit these additional structures. Tests between associative and direct mapped fill policy tables showed that associativity causes the SFDC to perform worse than when using a direct-mapped table. This is likely do to the fact that tags do not leave the table often enough. As later results show, we should have given more consideration to this fact. For the bypass buffer, larger sizes and associativities yield better results, albeit with diminishing returns. Later results elaborate on this and explain why the chosen configuration is two-way set associative, one sixteenth the size of the L1 cache.

## III. EXPERIMENTAL METHODOLOGY

*a. SimpleScalar*

In order to test the efficacy of the proposed SFDC, we implemented it using the SimpleScalar 3.0 tool set, a "suite of powerful computer simulation tools that provide both detailed and high-performance simulation of modern microprocessors [2]". The experiment utilized the latency independent cache simulator and focused solely in improving the effectiveness of the L1 data cache miss rates. Arguably, cache miss rates are the most important metric for determining the efficiency of a cache. A latency intrinsic model of our simulation would be valuable in the future (see Section VI).

We used four different baseline cache configurations as a method for comparison to our enhancements: 8KB direct mapped, 8KB two-way set associative, 16KB direct mapped cache, and 16KB two-way set associative. Further details about these baseline configurations can be found in Table 1. These configurations are exemplary of modern primary level caches.

The SFDC augments each baseline configuration for four additional implantations. Moreover, the SFDC implementations are identical to the baseline implementation with three added improvements. First, the data cache contains one extra used bit in each cache line. Second, a variable sized, two-way set associative bypass buffer was added to the cache. The size of the bypass buffer varies from 1/64 the size of the baseline data cache all the way to the full size of the baseline data cache. For example, a 1/16 size bypass buffer would be 512 bytes for the two 8KB configurations and 1024 bytes for the two 16KB configurations. Finally, the direct mapped CFPT contains as many lines as sets in the baseline cache.

In an effort to compare the effectiveness of the SFDC to another cache enhancement based on communication limitations, a victim cache supplements each of the four baseline configurations. The victim cache is always fully associative, and the

| | |
|---|---|
| L1 Data Cache | 8KB Direct Mapped, 8KB Two-Way Set Associative, 16KB Direct Mapped or 16KB Two-Way Set Associative, all with 32 byte blocks and a LRU replacement policy |
| L2 Data Cache | 256KB Four-Way Set Associative with 64 byte blocks and a LRU replacement policy |
| L1 Instruction Cache | 8KB Direct Mapped with 32 byte blocks and a LRU replacement policy |
| L2 Instruction Cache | 256KB Four-Way Set Associative Cache with 64 byte blocks and a LRU replacement policy |
| Cache Flush | False |
| Instruction TLB | 256KB Four-Way Set Associative with 4KB pages and a LRU replacement policy |
| Data TLB | 512KB Four-Way Set Associative with 4KB pages and a LRU replacement policy |

**Table 1:** Baseline Cache Configuration

size of the victim cache equals the size of the bypass buffer.

*b. Benchmarks*

Four applications from the SPEC2000 benchmark suite were used to determine the value of the architectural modifications specified in Section 2. All four benchmarks were members of the integer component of the SPEC2000 benchmark suite: 175.vpr (VPR) which is used for FPGA circuit placement and routing, 181.mcf (MCF) which is used for combinatorial optimization, 197.parser (PARSER) which is used for word processing, and 164.gzip (GZIP) which is used for compression.

## IV. RESULTS

*a. Bypass Buffer*

Shown in Figure 3 are the results of tests on differing bypass buffer sizes for the VPR benchmark. The buffer sizes are defined as a fraction of the total L1 data cache size. Small bypass buffer sizes have the most impact on miss rate. At around 1/16 of the L1 data cache size, the size of the bypass buffer is marginally significant, and further increases in size result in diminishing returns. The final implementation of the SFDC makes use of a bypass buffer which is 1/16 the size of the L1 data cache. The bypass buffer uses two-way associativity as it is a good compromise between direct mapped caches and four-way caches.

*c. Reuse Distance*

Figure 4 illustrates miss rates versus data reuse distance for the baseline, SFDC, and victim cache configurations using the VPR benchmark. SFDC outperforms the baseline configuration by eliminating essentially all misses for reuse distances of 11 to 100 and reducing misses for distances of 101 to 1000 and 1001 to 10000 by half. The victim cache, when

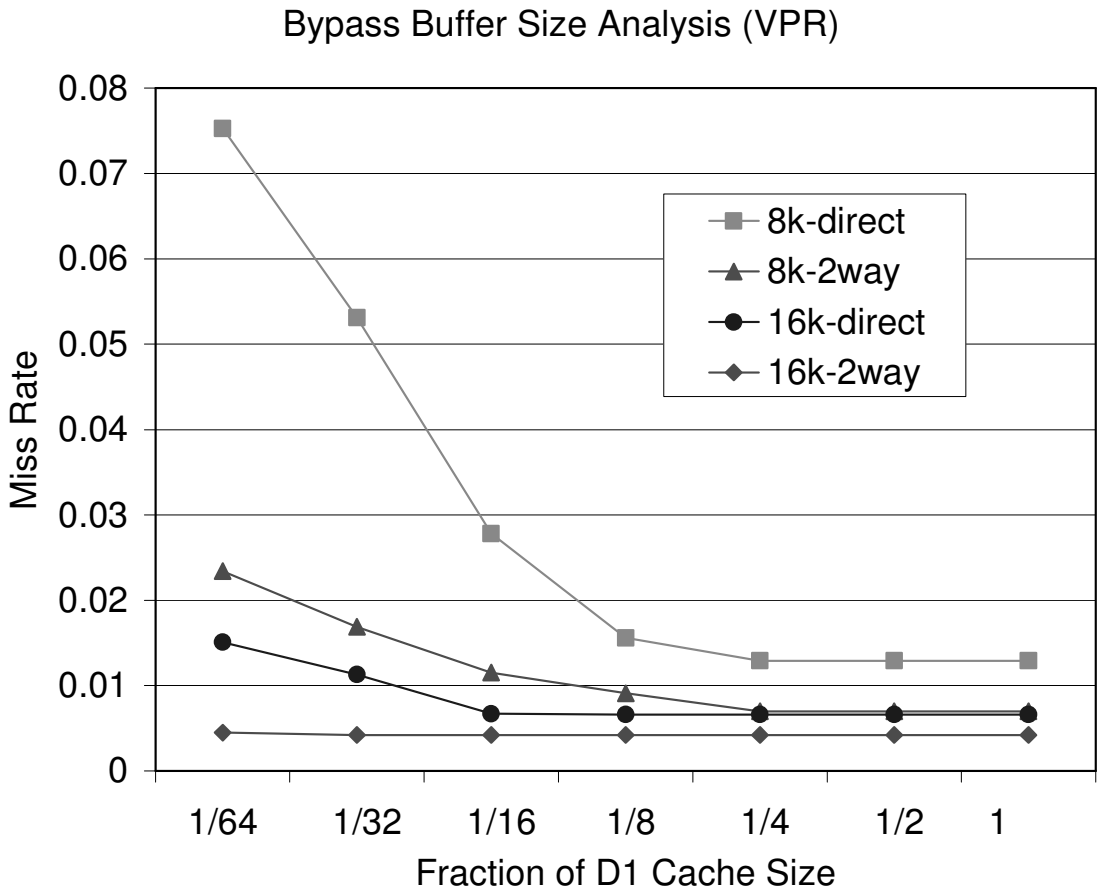


**Figure 3:** Bypass Buffer Size versus Miss Rate

compared against the SFDC, also eliminates misses for reuse distances of 11 to 100, further decreases misses for distances of 101 to 1000 and eliminates fewer misses for distances of 1001 to 10000.

The victim cache is essentially a last-in first-out queue which forwards evicted data blocks to the processor. The last-in first-out nature of the cache eliminates misses with shorter reuse distances. Due to the small number of spaces available in the victim cache, misses with longer reuse distances cannot be prevented as effectively. The requested data will have already been replaced in the cache by the time the next miss occurs. The SFDC implementation is not subject to this problem because many of those same elements which were placed into the victim cache and later evicted are never forced to leave the cache. This allows useful elements to remain in the cache for a longer time then they would in a traditional cache.

*c. Benchmark Results*

Figure 5 shows the resulting data cache miss rates for the four benchmarks chosen. Each graph shows the miss rates for 8KB and 16KB direct mapped caches, as well as 8KB and 16KB two-way set associative caches.

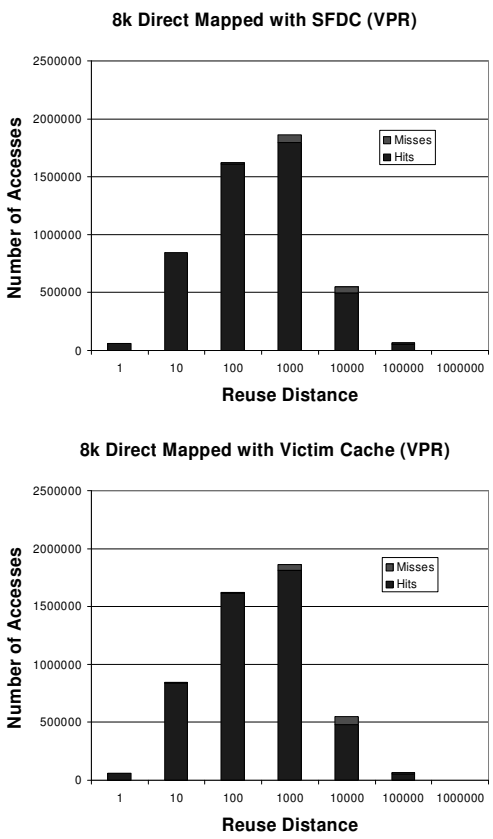


**Figure 4:** Reuse Distance versus Hit/Miss Counts for SFDC and Victim Cache

The VPR benchmark shows a significant miss rate decrease from the baseline for the SFDC in all cases. However, in both of the 16KB caches the victim cache outperforms the SFDC. The resulting miss rate drops in all cases are due to the fact that the data in the VPR benchmark is highly cacheable with small reuse distances.

MCF is a benchmark which is not particularly cacheable. As a result the SFDC implementation only manages to reduce the miss rates from the baseline, though in all cases decreases are recorded. The SFDC outperforms the victim cache for both of the two-way implementations.

The results of the PARSER benchmark for the SFDC show a slight miss rate decreases for the 8KB two-way and both 16KB caches. However, for an 8KB direct way configuration the SFDC increases the miss rate past the baseline rate because valuable blocks remain flagged in the CFPT and are not replaced. In this manner, the selective fill policy is too selective and does not allow the L1 cache to be fully utilized.

The GZIP benchmark shows modest gains for all cache configurations. The two-way associative cache implementations provide ample associativity to eliminate nearly all conflict misses. This indicates that GZIP has predominantly compulsory and capacity misses. In this case, the victim cache outperforms the SFDC.

## V. CONCLUSIONS

A SFDC improves the hit rate of standard caches for most benchmarks. It does so by preventing data with little or no temporal locality from entering the cache, allowing more useful data to reside there. The architectural enhancements required to accomplish this are not prohibitively complex, yet there exists the possibility for this implementation to cause a decrease in performance when it incorrectly classifies access patterns.

Using an unmodified cache for comparison does not take into account the additional complexity of the CFPT and bypass buffer. It is for this reason that we compared our results with a victim cache of the same size as the bypass buffer. Memory trace analysis showed that the victim cache and the SFDC reduced

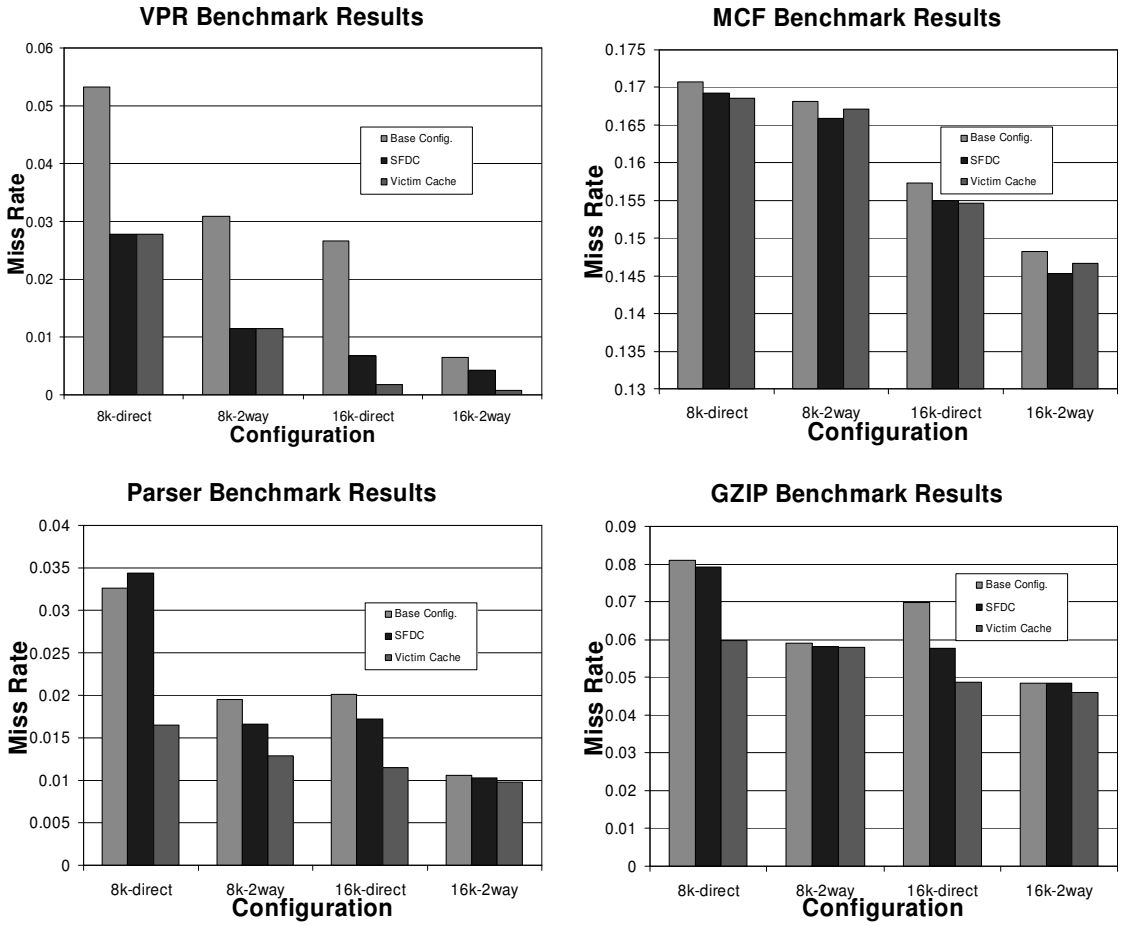


**Figure 5:** Results for VPR, MCF, PARSER, and GZIP

misses for different reuse distances. Simulation of the benchmarks on the two cache configurations showed that these benchmarks have shorter reuse distances, favoring the victim cache.

The results presented illustrate two important details about our implementation of the SFDC. Primarily, that it is a novel architectural improvement which reduces misses for large reuse distances. Additionally, the large dependence on bypass buffer size indicates that there are better metrics than that implemented in the CFPT for filtering the data allowed in the cache.

While the SFDC was topped by a comparably sized victim cache in most cases, this does not mean that the topic should be forgotten altogether. It is a new idea and will require further investigation to obtain the optimum results. More concentrated work in this area should lead to better performing selective fill data caches.

## VI. FUTURE WORK

There is still much to study about the SFDC. A latency intrinsic, cycle accurate simulation and comparison would show how the misses that the SFDC and victim cache eliminate affect overall performance. Results from these

simulations would also suggest whether or not the importance of a given miss correlates with reuse distance.

Elaborating on the fact that victim caches only protect against small reuse distances and that the SFDC does better for larger reuse distances, a worthwhile experiment would simulate both of these optimizations simultaneously on the same cache to determine whether or not they are complementary techniques. It is possible that the performance increase could be more than additive. The victim cache could keep small reuse distance miss patterns out of the CFPT, allowing it to focus on preventing longer reuse distance misses. This could give the improvements of a traditional victim cache plus improvements better than those presented for the SFDC.

An investigation into the cycle times of the victim cache and the selective fill data caches similar to that in [3] would determine whether the assumptions used to compare the victim cache and bypass buffer are valid. As on-chip communication becomes more and more expensive, minimizing area is important, but perhaps not so much as to ignore the affects of a 2-way compare in the bypass buffer versus a 16 or 32-way compare in the victim cache. This data would only serve to shrink the victim cache against which we compare, and possibly show that the SFDC does perform better in more cases.

The increase in miss rate observed on the Parser benchmark for an 8KB direct-mapped cache suggests that some sort of eviction policy from the CFPT should be implemented. Simple replacement is not good enough. One possibility is using an address' access frequency in the bypass buffer to determine whether or not it should be there. If an address begins to frequently hit in the bypass buffer, it should be removed from the fill policy table and allowed into the cache until its behavior becomes detrimental again.

Another possible improvement to the SFDC would be to dynamically determine the threshold behavior used in the CFPT in order to filter bad addresses out of the cache. This could be done either by a hardware controlled performance metric, or by adding a software controlled register to contain the threshold. The hardware approach could keep track of the cache's miss rate and change the threshold if too many misses over a given period. In software, the operating system could vary the threshold each time a program is run until the optimum threshold is found.

The concept of a selective fill data cache is a sound one, and will likely be one of the paths that architects chose as cache sizes decrease and their contents become increasingly important. More advanced cache architectures can always better exploit temporal locality, and we feel that this one shows much promise for the future.

REFERENCES

[1] Doug Burger, James R. Goodman, and Alain Kagi. "Limited Bandwidth to Affect Processor Design." IEEE Micro. November/December 1997.

[2] Simple Scalar LLC: To Serve and Protect. http://www.simplescalar.com.

[3] Wilton, S.J.E.; Jouppi, N.P. "CACTI: an enhanced cache access and cycle time model." Solid-State Circuits, IEEE Journal of. Volume: 31 , Issue: 5. May 1996.