Spatial Memory Streaming with Rotated Patterns

Michael Ferdman, Stephen Somogyi and Babak Falsafi†
http://www.ece.cmu.edu/~stems

Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
{mferdman,ssomogyi,babak}@ece.cmu.edu

†Parallel Systems Architecture Lab (PARSA)
Ecole Polytechnique Federale de Lausanne

Abstract

We submit for evaluation an implementation of Spatial Memory Streaming (SMS), originally presented at ISCA 2006 by Somogyi, et al. Research has shown that memory accesses often exhibit repetitive layouts that span large memory regions (e.g., several kB), and that these accesses recur in patterns that are predictable through code-based correlation. SMS is a practical on-chip hardware technique that identifies code-correlated spatial access patterns and streams predicted blocks to the primary cache ahead of demand misses.

SMS records a bit vector of cache blocks accessed within a spatial region of memory. The bit vectors are stored, indexed by the program counter (PC) and the spatial region offset of the "trigger" access, the first access in the spatial region. When a trigger access recurs, SMS uses the previously recorded bit vector to predict the remaining accesses within the spatial region that surrounds the newly encountered access.

In this submission, we employ an extension of the original SMS design. We use the observation that the same spatial patterns repeat regardless of the offset within the spatial region. We therefore use "rotation" to record the spatial regions independently of region offset, and recompute the rotation at the time of each prediction. While the original SMS design stores multiple bit vectors for every trigger access PC, rotated offset-independent patterns require storing only one bit vector, substantially reducing the predictor's storage requirements.

1. Introduction

In choosing a cache block size, system designers balance spatial locality with many other elements such as storage utilization and memory/processor pin bandwidth. Typically, the optimal cache block size sacrifices opportunity to exploit spatial locality for dense data structures to avoid excessive bandwidth overheads for sparse data structures. For simple data structures, such as arrays, spatial relationships can be exploited through simple prefetching schemes, e.g. stride prefetching [3].

Applications utilize data structures with repetitive layouts and access patterns. As the applications traverse their data sets, recurring patterns emerge in the relative offsets of accessed data. These accesses are frequently non-contiguous and do not follow a constant stride (e.g., binary search in a B-tree). Because sparse patterns may span large regions (e.g., an operating system page), the term *spatial correlation* rather than spatial locality is used to describe the relationship among accesses.

Past research has shown that spatial correlation can be predicted in hardware by correlating patterns with the code and/or data address that initiates the pattern [1,2]. *Spatial Memory Streaming (SMS)* [4] extracts spatially-correlated access patterns at run-time and predicts future accesses using these patterns. SMS then streams the predicted cache blocks into the processor's primary cache as rapidly as allowed by available resources and bandwidth, thereby increasing memory level parallelism and hiding lower-level cache and off-chip access latencies.

2. Definitions

We define the terms used in the rest of this paper:

- *spatial region:* A logical, fixed-size portion of the system's address space, consisting of multiple consecutive cache blocks.
- *spatial region generation:* The time interval over which SMS records accesses within a spatial region.
- *trigger access*: First access in a spatial region generation.
- *spatial pattern:* A bit vector representing the set of blocks in a region accessed during a spatial region generation.
- *spatial region offset*: The distance of an address from the start of the spatial region, measured in cache blocks.

The precise interval over which a spatial region generation is defined can significantly impact the accuracy and coverage of spatial patterns [2]. We choose the interval from the trigger access until any block accessed during the generation is evicted from the processor's primary cache. A subsequent access to any block in the

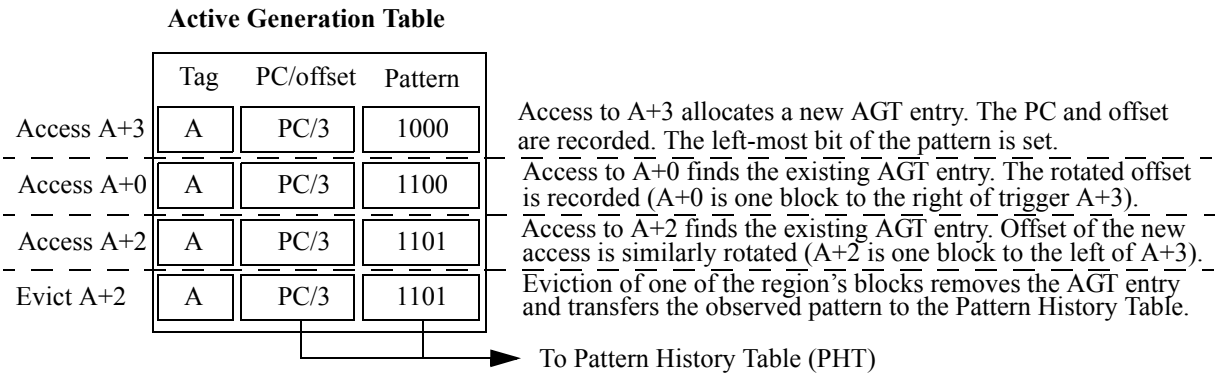

FIGURE 1. Active Generation Table. The figure illustrates the actions taken over the course of one spatial region generation.

region is the trigger access for a new generation. This definition ensures that the set of blocks accessed during a generation was simultaneously present in the cache.

3. Design

3.1. Observing Spatial Patterns

SMS learns spatial patterns by recording which blocks are accessed over the course of a spatial region generation in the active generation table (AGT). When a spatial region generation begins, SMS allocates an entry in the AGT. As cache blocks are accessed, SMS updates the recorded pattern in the AGT. At the end of a generation (eviction/invalidation of any block accessed during the generation), the AGT transfers the spatial pattern to the history table and the AGT entry is freed.

Spatial patterns are recorded in the AGT. Entries in the AFT are tagged by the *spatial region tag*, the high order bits of the region base address. Each entry stores the PC and spatial region offset of the trigger access, and a spatial-pattern bit vector indicating which blocks have been accessed during the generation. Unlike the original SMS proposal, we do not employ a filter table in the AGT.

The detailed operation of the AGT is depicted in Figure 1. Each L1 access searches the AGT table. If no match is found, this access is the trigger access for a new spatial region generation and a new entry is allocated (step 1 in Figure 1). If a matching entry is found, the spatial-pattern bit corresponding to the accessed block is set (step 2). Additional accesses to the region set corresponding bits in the pattern (step 3). Unlike the original SMS proposal, we apply rotation to the new bit, rotating the new bit to the left by the offset of the trigger access. The recorded pattern is always stored such that the trigger access is the left-most bit of the pattern. Consequently, the bit does not actually have to be stored, because all rotated patterns will have the left-most bit set.

Spatial region generations end with an eviction (step 4). The AGT is searched for the corresponding spatial region tag at every eviction. A matching entry is transferred to the pattern history table (PHT).

3.2. Identifying Recurring Spatial Patterns

Upon a trigger access, SMS predicts the blocks within the region that are spatially correlated and therefore likely to be accessed. Spatial correlation often arises because a data structure traversal recurs or has a regular structure (multiple accesses to the same structure allocated at multiple locations in memory). In this case, the spatial pattern correlates to the code (program counter values) executing the data structure traversal. Prior research indicates that PC-based indexing provides low storage and a fundamentally powerful indexing mechanism because it can eliminate cold misses.

When a code sequence repeats the same access pattern over a large data set, the PC-correlated spatial patterns learned at the start of the access sequence will provide accurate predictions for data that have not been previously visited. In this submission, we further extend this idea with the concept of vector rotation. Whereas the original SMS design stores a pattern for each PC+offset combination, this proposal stores only one pattern per PC, leveraging the observation that in most cases the PC+offset patterns are simple rotations of each other, centered around the spatial region offset of the trigger access.

Like SMS, our submission is designed to integrate with a traditional cache hierarchy. SMS uses a pattern history table for long-term storage of spatial patterns and to predict the pattern of blocks that will be accessed during each spatial region generation. The implementation of the PHT and the address prediction

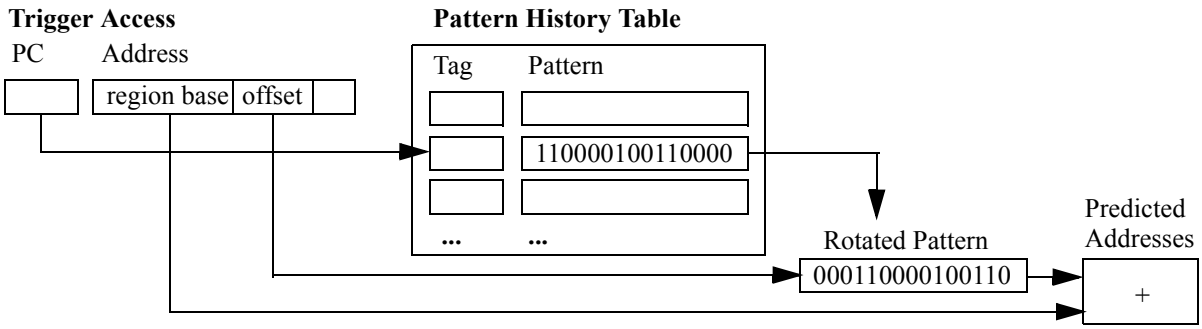

FIGURE 2. Pattern History Table and prediction process. Upon a trigger access that matches in the PHT, the region base address is combined with the rotated pattern to produce the predicted addresses.

process is depicted in Figure 2. The PHT is organized as a set-associative structure similar to a cache. The PHT is accessed using the PC of the trigger access. If an entry is found, the offset of the trigger access is used to rotate the stored pattern into a prediction pattern. The rotated pattern is then combined with the region tag of the trigger access to produce the predicted addresses, one per bit in the predicted pattern.

4. Implementation

4.1. Additional Implementation Details

High-bit in Rotated Implementation. Because the version of the predictor with bit-rotation guarantees that the left-most bit of the pattern is always set, we omit accounting for this bit's storage in the patterns when rotation is used.

MSHRs and Eviction Detection. The SMS implementation ends spatial generations at the time of an eviction from the cache. Because the provided simulator interface did not include a mechanism to notify the predictor of evictions, we created this mechanism by implementing MSHRs and probing the cache for addresses of interest using the available interface.

Imprecise MSHRs. Because the MSHRs we model are limited and only an approximation of a mechanism that keeps track of in-flight memory references (for example, there is an artificial limit of 16 MSHRs), we include an extension of the SMS logic to learn patterns at the time of eviction from the AGT in addition to at the time of eviction from the cache.

Index Hash. To reduce aliasing conflicts in the on-chip AGT and PHT structures, we use a strong hash of all bits of the input address (PC or Region Tag) before indexing or tagging the actual on-chip table.

Prediction Into L2 cache. Although the SMS design predicts L1 cache misses, for larger region sizes prefetching directly into the L1 cache creates too much L1 capacity and bandwidth pressure. For the large-region submission, we therefore use a design that prefetches only up to the L2 cache.

Cache Probe Mechanism. The provided simulator interface includes an unlimited-bandwidth mechanism to probe the cache for presence of a block. It would be beneficial to probe the cache prior to issuing a prefetch to filter our extraneous prefetches and save on "real" processor resources, however, we do not take this approach in our implementation in favor of a realistic hardware design.

4.2. Storage Requirements

All of the predictor storage is accounted for by the AGT and PHT tables. The AGT and PHT structures each have a tag array and data array. The AGT is tagged by the region tag, its size depends on the size of the spatial region of the predictor [27-log(region size)] and the AGT associativity. Each entry in the AGT data array includes a region offset [log(region size)], a pattern bit vector [w/o rotation: region size/64, w/rotation: region size/64-1], and a program counter [14-bit hash]. The PHT is tagged by the program counter. Each PHT data entry consists of the pattern bit vector.

We submit three versions of the predictor:

- 512B region, 4x16 AGT, 256x8 PHT, w/o rotation.
  AGT tags = 64*(18-2) = 1Kb
  AGT data = 64*(9+8+14) = 1.9375Kb
  PHT tags = 2048*(14-8) = 12Kb
  PHT data = 2048*8 = 16Kb
  Total = 30.9375Kb
- 512B region, 4x16 AGT, 256x8 PHT, w/rotation.
  AGT tags = 64*(18-2) = 1Kb
  AGT data = 64*(9+7+14) = 1.875Kb
  PHT tags = 2048*(14-8) = 12Kb
  PHT data = 2048*7 = 14Kb
  Total = 28.875Kb
- 8KB region, 8x16 AGT, 8x16 PHT, w/rotation, into L2.
  AGT tags = 128*(18-3) = 1.875Kb
  AGT data = 128*(9+63+14) = 10.75Kb
  PHT tags = 128*(14-3) = 1.375Kb
  PHT data = 128*127 = 15.875Kb
  Total = 29.875Kb

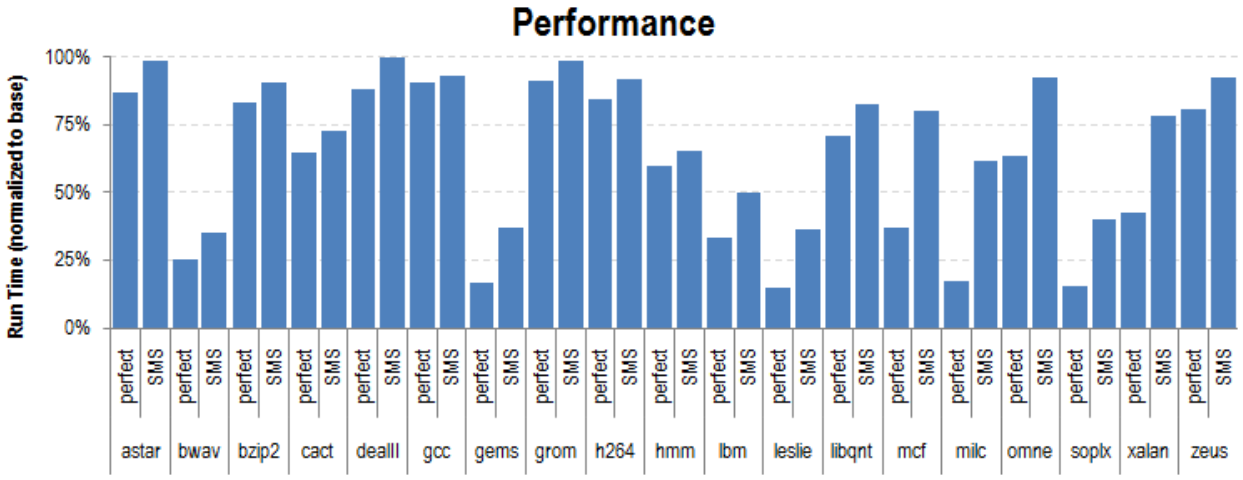

FIGURE 3. SMS Performance. Run time of SpecCPU 2006 applications normalized to the application run time on the base system. *Perfect* represents a configuration with zero L2 and memory latencies. *SMS* represents a configuration equipped with SMS using 512B regions without the rotation optimization.

5. Evaluation

5.1. Methodology

We present results of running the SpecCPU 2006 suite, compiled with Intel Compiler (icc) version 9.1. For each benchmark, the first reference input was used to collect a 100-million instruction trace after skipping 40 billion instructions from the start of execution.

To assess performance, benchmark traces were fed through the DPC timing simulator. The simulator models a 4-wide 15-stage pipeline, with a maximum of two load and one store operation issued on every cycle. A 128-entry instruction window and a perfect branch predictor are simulated. All non-memory instructions execute in one-cycle.

The memory model consists of a 2-level cache hierarchy with a 32KB 8-way set-associative L1 and 2MB 16-way set-associative L2. L2 cache hits are serviced in 20 cycles. Memory accesses are serviced in 200 cycles. To simulate memory bandwidth, queues are used to limit L2 requests to at most one per cycle and memory requests to at most one per ten cycles.

5.2. Results

We evaluate the effectiveness of SMS in improving processor performance by comparing our prefetcher against a *perfect* predictor. The perfect predictor system is configured with zero-latency links between the L1 and L2 caches and between the L2 and off-chip memory. The perfect configuration approximates a system where the latency of L2 hits and memory accesses is equivalent to prefetched hits, while still enforcing memory bandwidth constraints.

The results of our evaluation are presented in Figure 3. Run time of each application is shown, normalized to the run time of the application on the base system. We omit results for benchmarks *calculix, games, gobmk, namd, perlbench, povray, sjeng,* and *tonto* because these benchmarks show minimal sensitivity to prefetching (the perfect prefetcher configuration has 5% or less reduction in run time).

We observe that many of the SpecCPU 2006 applications can derive substantial benefit from memory system optimization. Some benchmarks (*bwav, gems, leslie, milc, splx*) show over 75% run-time reduction with a perfect prefetcher configuration. In many cases, the SMS system approaches the performance of the perfect prefetcher, reducing run time by over 60%, or more than 2.75x speedup over the base system.

In a number of benchmarks, most notably *mcf* and *xalan*, the performance observed in the perfect configuration is significantly better than with SMS. The SMS prefetcher is able to achieve substantial performance improvements on these applications; however, due to the large number and high frequency of misses, SMS is not able to prefetch many of the misses in time to fully hide their access latency from the processor core. Additionally, these applications are bandwidth hungry, and some opportunity is lost as a result of bandwidth that is consumed by extraneous prefetches (i.e., predictions are made by SMS but the prefetched blocks are never accessed by the processor).

Finally, we note that Figure 3 reports results for SMS without the rotation optimization described in Section 3. The rotation optimization substantially reduces the number of patterns that need to be stored in the PHT, at a slight cost in performance (less than 1%

for most benchmarks) for applications whose PHT storage requirements are met without the rotation optimization. Additionally, increasing the spatial region size captures farther-reaching spatial correlation, but this comes at a significant performance loss for applications where only fine-grained spatial correlation is present (e.g., small data structures or array accesses with fine-grained elements). Unlike the commercial workloads which are targeted by our research group and for which SMS was originally constructed, the SpecCPU 2006 workloads do not exhibit far-reaching spatial correlation and do not require storage for a large number of spatial patterns. The 2K-entry PHT which fits into the 32Kbit budget provided by the DPC rules is sufficient to capture all the patterns needed by SMS for the SpecCPU 2006 benchmarks, thus rendering the rotation optimization and increased spatial region size ineffective—and in some cases detrimental—compared with the baseline SMS design without rotation and 512B regions.

6. Conclusion

In this paper, we described our implementation of Spatial Memory Streaming for DPC-1, including a novel mechanism of pattern bit-vector rotation to reduce SMS storage requirements. We presented results which, for the first time, evaluated SMS on SpecCPU 2006 benchmarks, showing SMS to be a highly cost-effective prefetcher for this class of workloads.

Acknowledgements

The authors would like to thank Andreas Moshovos for his contributions to and involvement with the Spatial Memory Streaming work. Additionally, we would like to thank Chi F. Chen and Andreas Moshovos for their prior work and insight on spatial prediction. We would like to thank the DPC organizers and reviewers for their efforts and comments, and also to offer great thanks to Alaa R. Alameldeen for his assistance in collecting the evaluation results presented in this paper.

References

[1] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos. Accurate and complexity-effective spatial pattern prediction. In *Proceedings of the Tenth Symposium on High-Performance Computer Architecture*, Feb. 2004.

[2] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *Proceedings of the 25th International Symposium on Computer Architecture*, June 1998.

[3] T. Sherwood, S. Sair, and B. Calder. Predictor-directed stream buffers. In *Proceedings of the 33rd International Symposium on Microarchitecture,* Dec. 2000.

[4] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Spatial Memory Streaming. In *Proceedings of the 33d International Symposium on Computer Architecture*, June 2006.