

Run-time Adaptive Cache Management

Teresa L. Johnson Daniel A. Connors Wen-mei W. Hwu
Center for Reliable and High-Performance Computing
University of Illinois
Urbana-Champaign, IL 61801
{tjohnson,dconnors,hwu}@crhc.uiuc.edu

1 Introduction

As the microprocessor industry struggles to deliver higher performance superscalar and VLIW processors, memory access penalties have become a major issue. The growing disparity between processor and memory performance will make cache misses increasingly expensive. Additionally, data caches are not always used efficiently, resulting in large numbers of data cache misses. In numeric programs there are several known compiler techniques for optimizing data cache performance. However, integer (non-numeric) programs often have irregular access patterns that are more difficult for the compiler to optimize.

As memory latencies increase, the importance of cache performance improvements at each level of the memory hierarchy will continue to grow. Also, as the available chip area grows, it makes sense to spend more resources to allow intelligent control over the cache management, in order to adapt the caching decisions to the dynamic accessing behavior. In the past, cache management techniques such as cache bypassing were implemented manually at the instruction programming level. Additionally, spatial locality was often exploited via large block sizes and other fixed amounts of hardware prefetching. Our goal is to develop a framework for adaptive and automatic control of cache management techniques.

The objective of our research is to improve cache effectiveness in order to deal with long memory latencies, utilizing run-time adaptive cache management techniques, optimizing both performance and cost of implementation. Specifically, we are aiming to increase data cache effectiveness for integer programs. We propose a microarchitecture scheme where the hardware determines data placement based on dynamic referencing behavior. This scheme is fully compatible with existing Instruction Set Architectures.

Initial studies show that run-time adaptive cache management can significantly improve the overall performance of integer applications [1][2]. The improvements are due to increased cache hit rates and reduced cache miss handling latencies. However, there is still a large amount of potential improvement available in the cache hit ratios.

Due to space constraints we are unable to discuss related work here, but an overview can be found elsewhere [1][2].

2 Overview

There are several key components in intelligent cache management schemes:

- *Hardware monitors* to track run-time memory access patterns. These monitors will track the frequency of accesses to particular regions of memory, as well as the temporal and spatial locality in those regions. Monitors may also collect cache line utilization information on a per-line basis.
- *Adaptive caching decisions* to proactively control the movement and placement of data in the hierarchy based on the data usage characteristics detected by the hardware monitors. This includes selective cache bypassing and spatial locality optimizations, which have shown great potential in our previous studies, as well as other techniques such as intelligent cache remapping.

3 Hardware Monitors

The hardware monitors track cache and memory activity. Benchmark analysis suggests that tracking memory regions, rather than load instructions, will better capture access patterns. For example, in the compress benchmark a single load instruction will often access data with dramatically different usage patterns, even during small time intervals. The amount of information generated, as well as the accuracy of the statistics, will depend on the granularity at which we track the memory regions. Initial studies suggested that 1K-byte regions of memory, called macroblocks, will balance this tradeoff. It may be beneficial to track individual cache blocks while they are in cache, and somehow absorb this information into the macroblock statistics on replacement.

Currently, we implement these hardware monitors as a separate table in hardware called a Memory Address Table (MAT). The current design of the MAT structure is a counter array that keeps track of the access frequency of the macroblocks, ideally containing one entry per macroblock accessed. This design has been expanded to include another counter per macroblock which indicates the presence of spatial locality. We will be examining extensions of this design to detect temporal locality and other usage patterns. We will also examine alternative designs which

reduce the hardware cost, including integration with the TLB and page tables.

Another important issue is the retention of information across context switches and multiple invocations of the same program. Several questions that will need to be answered include whether such retention results in high payoff, and if so, whether there are clever ways to achieve it with little or no operating system intervention.

4 Caching Decisions

The hardware monitors described in Section 3 can then be used to guide caching decisions, usually triggered by cache misses. Three such types of decisions are cache bypassing, spatial locality optimizations and cache remapping.

4.1 Cache Bypassing

Extensive studies of MAT-driven data cache bypassing, show that significant speedups can be achieved by selectively bypassing data with low access frequencies [1]. This bypassing is only performed on conflicts with much more frequently accessed data, as detected by the MAT access frequency counters. The mechanism to detect this situation involves accessing the corresponding macroblock entry in the MAT on a cache access, and comparison to the macroblock entry corresponding to the cache entry that would be replaced. The second MAT lookup and associated comparison are only necessary on a cache miss, and can be performed on the following cycle.

We have examined several performance issues of the bypassing scheme, including the effects of temporal and spatial locality in infrequently accessed data. To allow exploitation of temporal locality we place all bypassed data in a bypass buffer, which will then be accessed similar to a victim cache, however data will never be moved from the bypass buffer into the main cache as in victim caching schemes. We will also examine alternative MAT monitors that detect temporal locality and allow further refinement of the bypassing decision or guide selective bypass caching.

4.2 Spatial Locality Optimizations

Spatial locality optimizations must be able to detect and adapt to the varying spatial locality characteristics both within and across applications in order to be effective. We developed a scheme which meets these objectives by detecting the amount of spatial locality in different portions of memory, and making dynamic decisions on the appropriate number of blocks to fetch on a memory access [2]. A Spatial Locality Detection Table (SLDT) facilitates spatial locality detection for data while it is cached. This information is later recorded in the MAT for long-term tracking, and is then used to tune the fetch sizes for each missing access.

Detailed simulations of several applications showed that significant speedups can be achieved by our techniques. The improvements are due to the reduction of conflict

and capacity misses by utilizing small blocks and small fetch sizes when spatial locality is absent, and utilizing the prefetching effect of large fetch sizes when spatial locality exists. In addition, we showed that the speedups achieved by this scheme increase as the memory latency increases.

4.3 Cache Remapping

Preliminary measurements suggest that the data caches are not evenly utilized. The cache blocks replaced on a miss in a direct-mapped cache were rarely among the least recently used blocks in the cache, even after using a MAT to bypass the most infrequently accessed blocks. This suggests that there is heavy contention for a subset of the cache lines, while other lines are under-utilized. In order to improve the hit ratios even further we will investigate dynamic cache remapping guided by the MAT. The hit ratios can be improved by remapping conflicting cached data to less utilized cache locations. Our work will extend existing methods of cache remapping which rely on a static hashing function to choose the remap location, to instead allow the MAT and related hardware to intelligently guide remapping choices.

Because at least one extra cycle is needed to access remapped data, the more infrequently accessed data among the conflicting data should be selected for remap. The MAT can guide these decisions using its access counters. Usage counters associated with each cache line can be used to detect under-utilized locations. Then additional information must be contained in the tag store entries to identify the remap location. While this will add a cycle to each missing access, in the case of a hit to the remap location the large memory latency can be avoided.

5 Future Directions

We will continue to develop intelligent monitor mechanisms to identify run-time memory access patterns. Additionally, we will need to further evaluate the hardware complexity of these monitors, and conduct simulations to tune and validate designs. By taking advantage of the information provided by the monitors, we can further refine these schemes, as well as develop new intelligent cache management algorithms.

Finally, these mechanisms will need to be examined in the context of a multiprocessor system. Techniques such as adaptive cache bypassing and spatial locality optimizations could be utilized to alleviate such problems as false sharing.

References

- [1] T. L. Johnson and W. W. Hwu, "Run-time adaptive cache hierarchy management via reference analysis," in *Proceedings of the 24th International Symposium on Computer Architecture*, pp. 315–326, June 1997.
- [2] T. L. Johnson, M. C. Merten, and W. W. Hwu, "Run-time spatial locality detection and optimization," in *To appear in the Proceedings of the 30th International Symposium on Microarchitecture*, December 1997.