Comparing Cache Injection and Data Prefetching for I/O Data in Chip-Multiprocessors

Edgar A. León and Arthur B. Maccabe
Computer Science Department
University of New Mexico

The memory wall, disparity between processor and memory speeds, adversely affects the performance of applications, particularly those limited by memory bandwidth. Examples of these computations include scientific and vector computations, encryption, signal processing, string processing, image processing, and DNA sequence matching.

Several techniques both in hardware and software have been devised to manage the imbalance between processor and memory speeds including *data caching*, *prefetching*, *software access ordering*, and *hardware-assisted access ordering*. While these techniques have proven to be effective for a variety of workloads, they all incur in memory latency and memory traffic delays when accessing I/O data. *Cache injection* [2, 3], a complimentary technique to caching for I/O data, reduces memory latency and memory bandwidth usage by placing I/O data directly into cache from the I/O bus. While effective its applicability is limited to I/O data.

In this work, we present an evaluation of cache injection and data prefetching for incoming network data in an environment suitable for high-performance applications. We chose data prefetching due to its similarity with cache injection. Cache injection is a producer-driven non-binding technique that reduces memory latency and memory bandwidth usage by placing I/O data directly into cache. Prefetching is a consumer-driven non-binding technique that reduces memory latency by fetching soon to be accessed data into cache. Processor requests for this data are met by the cache.

Many studies have shown that prefetching is an effective technique to reduce memory latency. However, it has several disadvantages when compared to cache injection. First, data may be fetched too late to reduce memory latency. Second, it increases memory bandwidth traffic due to two transactions: (1) transfer of data from the I/O producer to memory, and invalidating cached copies; and (2) fetching data from memory. With cache injection, the second transaction is not necessary. Third, even if prefetching correctly anticipates the access to I/O data, fetch requests can only be served from main memory, incurring in memory latency as well as using memory bandwidth.

We compare these techniques using simulation. We use IBM's PowerPC full-system simulator, Mambo [1], running the K42 research OS. We use a Power5 architectural configuration with a cache injection implementation to the L3 cache. Our evaluation consists of measuring memory bandwidth and execution time of an application in three configurations: (1) base case with no optimizations; (2) prefetching; and (3) cache injection. The application used in this evaluation performs linear traversals of incoming network data followed by a reduction operation. To evaluate cache injection in a suitable environment for high-performance applications, we use a zero-copy, OS-bypass messaging system based on UDP semantics [4].

First, we measure the memory bandwidth used by the application in terms of the number of memory reads issued to the memory controller. The base case and prefetching perform equally as prefetching has to fetch incoming network data from memory. Prefetching anticipates data accesses correctly due to the sequential access pattern used by the application. Cache injection reduces the number of memory reads by up to 96% as all application accesses to incoming network data hit the L3 cache. Second, we measure the execution time of the application in processor cycles. Both cache injection and prefetching outperform the base case as they both reduce the number of cache misses. Prefetching reduces execution time by up to 37% while cache injection by up to 30%. Prefetching performs better because it fetches blocks to the L2 and L1 caches, while our cache injection implementation targets the L3 cache. We expect that injections to the L2 will perform as good as prefetching.

The performance of cache injection is dependent on several factors including timely usage of data, the amount of data, and the application's data usage patterns. To leverage this technique injection policies to determine when and where to inject data are necessary. In this work, the application uses the data shortly after it is injected into the cache. If the application does not use the data promptly, cache injection may create cache pollution taking the application's working set out of the cache. Thus, the performance benefits of this technique rely on a *good injection policy*.

In conclusion, cache injection outperforms prefetching on memory bandwidth and performs comparably on execution time. This work provides a basis for studying injection policies in a high-performance computing environment. Exploration of these policies based on OS, compiler, cache and application information remains as future work.

[1] Patrick Bohrer et al. Mambo – a full system simulator for the PowerPC architecture. *ACM SIGMETRICS Performance Evaluation Review*, 31(4):8–12, March 2004.

[2] Patrick Bohrer et al. Method and apparatus for accelerating Input/Output processing using cache injections, March 2004. US Patent No. US 6,711,650 B1.

[3] Ram Huggahalli et al. Direct cache access for high bandwidth network I/O. In *32nd Annual International Symposium on Com-*

puter Architecture (ISCA 2005), pages 50–59, Madison, WI, June 2005.

[4] Edgar A. León and Michal Ostrowski. An infrastructure for the development of kernel network services. In *20th ACM Symposium on Operating Systems Principles (SOSP'05). Poster Session*, Brighton, United Kingdom, October 2005. ACM SIGOPS.