Hardware Prefetchers for Emerging Parallel Applications

Biswabandan Panda
Computer Architecture and Systems Lab
CSE Department, IIT Madras, India
biswa@cse.iitm.ac.in

Shankar Balachandran
Computer Architecture and Systems Lab
CSE Department, IIT Madras, India
shankar@cse.iitm.ac.in

ABSTRACT

Hardware prefetching has been studied in the past for multi-programmed workloads as well as GPUs. Efficient hardware prefetchers like stream-based or GHB-based ones work well for multiprogrammed workloads because different programs get mapped to different cores and are run independently. Parallel applications, however, pose a different set of challenges. Multiple threads of a parallel application share data with each other which brings in coherency issues. Also, local prefetchers do not understand the irregular spread of misses across threads. In this paper, we propose a hardware prefetching framework for L1 D-Cache that targets parallel applications. We show how to make efficient prefetch requests to the L2 cache by studying and classifying the patterns of L1 misses across all the threads. Our preliminary results show an improvement of 7% in execution time on an average on the PARSEC benchmark suite.

Categories and Subject Descriptors

B.3 [**Memory Structures**]: Cache memories

Keywords

Prefetching

1. DESIGN AND IMPLEMENTATION

We categorize miss patterns into three categories. A demand miss on a cache line from a particular thread is a (i) *Global Stride* (GS) miss if the line is at a stride S from another line that saw a demand miss from another thread; (ii) *Local Stride* (LS) miss if the line is at a stride S from another line that was missed from the same thread; (iii) *Producer_Consumer Shared* (PCS) miss if the miss to the same line happened in the other threads. The observation window is restricted to the last 32K cycles for this classification.

To alleviate the GS and PCS misses, we add two prefetchers called the GS prefetcher and the PCS prefetcher beside the L2 cache. LS misses are handled by the local prefetcher.

Prefetching Framework: *GS* prefetcher uses a table indexed by tag-bits. Fig. 1 shows the structure of the GS table. Misses *miss1* and *miss2* originating from two different threads *tid1* and *tid2* are entered into the GS table. A miss from a thread other than *tid2* is called *miss3*. If the strides between subsequent pairs in these misses are the same (say S), a particular entry is trained.

PACT'12, September 19–23, 2012, Minneapolis, Minnesota, USA.
ACM 978-1-4503-1182-3/12/09.

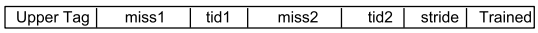

Figure 1: GS Table

Subsequent misses to a trained entry results in a prefetch request to $miss\_address + S$. The response from the L2 cache is sent to a hardware structure called the *filler* (which is private to each core). In the immediate future, if a demand miss occurs to the prefetched address, the data is transferred to L1 D-cache and deallocated from the respective *fillers*. PCS prefetcher maintains saturating counters, named PCS_{ij}, for every pair of threads. The PCS_{ij} counters are incremented when the PCS prefetcher observes trained entries in GS table with stride zero and they are decremented when the entries are evicted. For a demand miss from thread i, the PCS prefetcher issues prefetch responses as a *selective-broadcast* message to the fillers of those threads for which the PCS_{i*} counters have exceeded a threshold value.

Fig. 2 shows the overall design of our prefetching framework for a 2 core system which consists of *GS*, *PCS* and *LS* prefetchers.

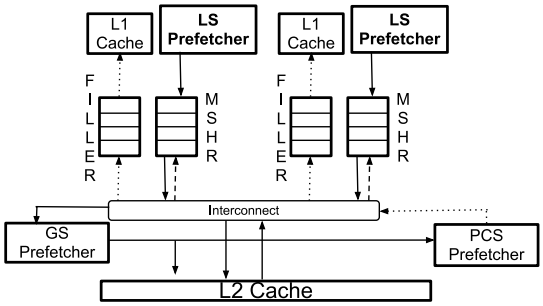

Figure 2: Prefetcher Framework

Preliminary Results: We use gem5 FS simulator to implement *GS* and *PCS* with L1 D-cache of size 32KB. We use stream-based prefetcher as the *LS* prefetcher. *GS* prefetcher uses a table of 256 entries. PCS prefetcher uses 3 bit PCS_{ij} counters per thread. We use PARSEC benchmarks with *sim-medium* as input set to evaluate our framework. Our framework outperforms the feedback-prefetcher [1] in terms of execution time by 4.2% and 7.1% on an average for 2-core and 4-core systems. The accuracy of *GS* and *PCS* is above 70% on an average. We believe that a collaborative framework between *GS*, *PCS* and *LS* prefetchers will further improve the performance.

[1] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt."Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers", in HPCA-13, 2007, pp. 63-74.