MLP-Aware Dynamic Cache Partitioning

Miquel Moreto[1], Francisco J. Cazorla[1], Alex Ramirez[1,2], Mateo Valero[1,2]

[1]Universitat Politècnica de Catalunya, Departament d'Arquitectura de Computadors, {mmoreto,aramirez,mateo}@ac.upc.edu

[2]Barcelona Supercomputing Center – Centro Nacional de Supercomputación. francisco.cazorla@bsc.es

1 Poster Abstract

The limitation imposed by instruction-level parallelism (ILP) has motivated the use of thread-level parallelism (TLP) as a common strategy for improving processor performance. TLP paradigms such as simultaneous multithreading (SMT), chip multiprocessing (CMP) and combinations of both offer the opportunity to obtain higher throughputs. However, they also have to face the challenge of sharing resources of the architecture. Simply avoiding any resource control can lead to undesired situations where one thread is monopolizing all the resources and harming the other threads. Some studies deal with the resource sharing problem in SMTs at core level resources like issue queues, registers, etc. In CMPs, resource sharing is lower than in SMT, focusing in the cache hierarchy.

Some applications present low reuse of their data and pollute caches with data streams, such as multimedia, communications or streaming applications, or have many compulsory misses that cannot be solved by assigning more cache space to the application. Traditional eviction policies such as Least Recently Used (LRU), pseudo LRU or random are demand-driven, that is, they tend to give more space to the application that has more accesses to the cache hierarchy. As a consequence, some threads can suffer a severe degradation in performance. Previous work has tried to solve this problem by using static and dynamic partitioning algorithms that monitor the L2 cache accesses and decide a partition for a fixed amount of cycles in order to maximize throughput [4, 5] or fairness [2]. Basically, these proposals predict the number of misses per application for each possible cache partition. Then, they use the cache partition that leads to the minimum number of misses for the next interval. We denote this algorithm *MinMisses*.

A common characteristic of all these proposals is that they treat all L2 misses equally. However, it has been shown that L2 misses affect performance differently depending on how clustered they are. As was said in [1], an isolated L2 miss has approximately the same misspenalty than a cluster of L2 misses, as they can be served in parallel if they all fit in the reorder buffer. As a consequence, an isolated miss has a higher impact on performance than a miss in a burst of misses as the memory latency is shared by all clustered misses. Based on this fact, we propose a new dynamic cache partitioning algorithm that gives a cost to each L2 access according to its impact in final performance. We detect isolated and clustered misses and assign a higher cost to isolated misses. Then, our algorithm determines the partition that minimizes the total cost for all threads, which is used in the next interval. We denote this algorithm *MLP*.

Acknowledgments.This work has been supported by the Ministry of Education of Spain under contract TIN-2004-07739-C02-01 and grant AP-2005-3318, the HiPEAC European Network of Excellence, and the SARC European Project (Contract number 27648).

Our results prove that differentiating between clustered and isolated L2 misses leads to dynamic cache partitions with higher performance than previous proposals.

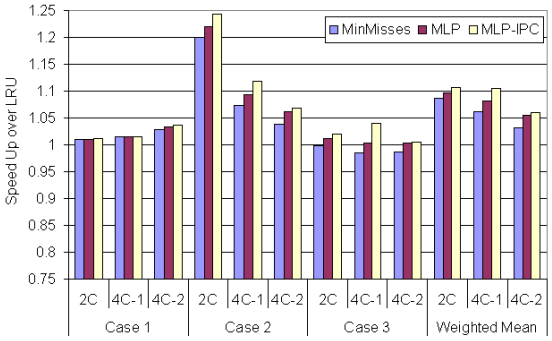

Figure 1. Average throughput speed up over LRU

Figure 1 shows the average throughput speed up that our proposals obtain over LRU. *MLP-IPC* is a variant of the *MLP* algorithm that gives priority to applications with high IPC. We also compare our results with *MinMisses* [4, 5]. We use configurations with two cores (*2C*) and four cores with a shared L2 cache of 1MB and 16-way associativity (*4C-1*) or 2MB and 32-way associativity (*4C-2*).

We followed the methodology introduced in [3] to classify workloads of benchmarks in the SPEC CPU 2000. In that way, we have one situation where our policies show similar results to traditional LRU (Case 1), one situation where significant speed ups are obtained (Case 2) and a third situation where *MinMisses* obtains performance losses while our proposals attain performance benefits (Case 3). We obtain improvements over LRU up to 63.9% (10.6% on average) and over previous proposals up to 15.4% (4.1% on average) in a four-core architecture. Our results also show that our proposals do not obtain these improvements at the expense of fairness. *MLP* and *MLP-IPC* outperform LRU by 5.4% and 3.8% in fairness (measured as the harmonic mean of relative IPCs), while *MinMisses* attains a 4.1% improvement over LRU.

References

[1] T. S. Karkhanis and J. E. Smith. A first-order superscalar processor model. In *ISCA*, 2004.

[2] S. Kim, D. Chandra, and Y. Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *PACT*, 2004.

[3] M. Moreto, F. J. Cazorla, A. Ramirez, and M. Valero. Explaining dynamic cache partitioning speed ups. *IEEE CAL*, 2007.

[4] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO*, 2006.

[5] G. E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *HPCA*, 2002.

1089-795X/07 $25.00 © 2007 IEEE
DOI 10.1109/PACT.2007.32