Prioritization of Prefetching Traffic into Multicore Networks

Carles Aliagas [*] [**]

Department of Computer Engineering and Mathematics, Universitat Rovira i Virgili
Tarragona, Spain
carles.aliagas@urv.cat

1 Prefetching Overview

Cache holds very few data, so it is compulsory to decide which ones are best suited to be stored in. Their goal, is to increment the probability to find them in cache and avoid the main memory access. We have to decide when–how–where to copy a data into cache, when–how-where to look for data in cache and when-how-where to dismiss data from cache. This means hardware algorithms to *place*, *look–for* and *replace*.

Observing typical program pattern access, we can see a property of *data locality*. That means that after accessing a data it is very probable to access the same data, or a data near that one, in the near future. There are two types of locality: temporal locality (the same data) or spatial locality (data next to the accessed one). Caches designs wants to take profit from this property.

In order to take profit of spatial locality instead of bringing only one word (4, 8 bytes) caches brings a full block (16 to 128 bytes). This is a basic *prefetch* mechanism. This is very useful for scientific programs that have sequential accesses to vector and matrix, but if the block size is increased it can increase pollution.

Caches are initially empty. So at the beginning some accesses will produce a miss (also known as *compulsory miss*). This behaviour leads to a research challenge that tries to reduce the compulsory misses by bringing blocks to cache before the processor ask for them. This is the target of all prefetching mechanism.

Prefetching can be applied at any level of the memory hierarchy and from a software or hardware point of view. At the present moment, most of the processors are multicore/multithread and in this scenario caches can be shared between cores and threads. Several problems arise with prefetching in all those alternatives.

[*] Thanks to: M. Torrents (BSC), R. Martínez (Oracle)
[**] PhD Supervisors: C. Molina (URV), A. González (UPC)

2 Prefetching Challenges

With the idea of providing a good performance in terms of execution time, many prefetching[4] mechanism have been proposed for unicore processors with the main challenge to correctly choose the next useful block and the correct timing. Multicore prefetching[5] has also two main focus of prefetching, one is to bring data as near as possible to the core that would use it (privates caches) and the other is to bring data to the main shared cache of the processor from main memory. The first one will have to deal with the coherence cache protocol and the second one will have to deal with the shared resources of the cores (network, external connections,...). Many[7][8] uses the extra computation of multicores/multithreads to execute the prefetch mechanism but at the expense of not using them for raw computation. Different approaches have to deal with this dilemma and use complex prefetch alternatives that uses a lot of processors resources or simple prefetch alternatives that does not perform as good.

One resource that influence memory latency is the network that connects the cores with the shared cache and the external main memory. A good prefetching algorithm should not produce an increase of traffic in the memory interconnection. An increase in memory traffic entails higher power consumption and a higher degree of contention in the interconnection network. Note that the number of memory requests are in most of the cases going to be higher than in a system without prefetching. This is especially true if we are in a multicore system because it increases on-chip communication since coherence between the L1 caches of the tiled CMP must be ensured. An increase in the congestion of the network will most probably increase the latency of not only prefetching requests but also regular memory operations

Our proposal focus on network congestion in order to dynamically reorder data access to prioritize regular request in detrimental of prefetch request to avoid increments in latency of regular data access.

3 Prioritization into the network: Our proposal

Traditionally, memory systems do not differentiate between prefetch and regular requests. Recently a number of approaches[9][10][11] have appeared that give several priorities to both types of requests depending on the predicted behaviour, since it has been shown that delaying regular requests may degrade performance if prefetch requests are not accurate.

The prefetcher mechanism send requests to the network to bring data from external memory to internal caches or to move data between the internal caches. That requests have to coexist with regular data accesses and travel together within the network subsystem. One challenge is that prefetching does not have to penalize regular accesses but the network subsystem does not have information about the different types of requests.

In our proposal the prefetcher mechanism will add information to its data access in order to mark them as a prefetch requests. Also it will be add information related to the timing of the prefetch. Network routers will use that information to reorder requests in its queues and apply/modify the priority of them. Moreover, it will use dynamic information of congestion in order to apply different policies. regular accesses have maximum priority and prefetch accesses have variable priority depending on its time request.

Another possibility of the mechanism is to discard prefetch accesses when they are arriving after its time request, and also discard some of them when the network utilization is near full capacity.

Several experiments will determine optimal values of priorities and thresholds needed by the mechanism in order to modify priorities and discard some or all the prefetch access.

4 Experimental Framework

The SimpleScalar Tool Set[1]provide simulators ranging from a fast functional simulator to a detailed out-of-order issue processor that supports non-blocking caches, speculative execution, and state-of-the art branch prediction. In this work, we use the sim-outorder to obtain program statistics.

The Standard Performance Evaluation Corporation (SPEC)[2] is an organization founded in 1988 that provides several families of benchmarks to measure the performance of different computer systems. In this work we consider the CPU family, designed to provide performance measurements that can be used to compare compute-intensive workloads on different computer systems. In particular, we consider the following suite of the CPU family: SPEC CPU2006.

The Opnet Modeler Suite[3] provides a suite of protocols and technologies to design, model, and analyze communication networks.

5 Conclusions and Future Work

Prefetching in multicore processors shows new challenge designs that have to deal with sharing the available resources of the processor for demand requests and prefetch requests. We focus on network congestion as a measure to reorder, depriorize and also discard prefetch requests. In this way, tuning the correct values of our mechanism will reduce global average access time of memory accesses.

Acknowledgement. This work has been partially supported by the Spanish MCI and FEDER funds (EU) under contracts TIN2016-75344-R and TIN2016-77836-C2-1-R

References

[1] D.Burger, T.M.Austin and S.Bennet, Evaluating Future Microprocessors: The SimpleScalar Tool Set, CS-TR-96-1308. University of Wisconsin, July 1996.

[2] http://www.spec.org/, "The Standard Performance Evaluation Corporation".

[3] https://www.riverbed.com/, "Opnet Modeler Suite"

[4] N. Oren "A Survey of Prefetching Techniques". TR CS-2000-10, University of the Witwatersrand, 2000.

[5] S.Byna, S.Chen and X-H.Sun Taxonomy of data prefetching for multiprocessors. Journal of Computer Science and Technology, 24(3):405–417, 2009.

[6] Hennessy J, Patterson D. Computer Architecture: A Quantitative Approach. The 4th Edition, Morgan Kaufmann, 2006.

[7] Kim D, Liao S S, Wang P H, et al. Physical experimenta- tion with prefetching helper threads on Intel's hyper-threaded processors. In Proc. the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization, Palo Alto, USA, March 21–24, 2004, p.27.

[8] I. Ganusov and M. Burtscher, "Future Execution: A Hardware Prefetching Technique for Chip Multiprocessors", Proceedings of the 14th Parallel Architectures and Compilation Techniques, 2005.

[9] A. Flores, J. L. Aragón, and M. E. Acacio "Energy-efficient hardware prefetching for CMPs using heterogeneous interconnects," in Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on, pp. 147–154, IEEE, 2010.

[10] N. Chidambaram Nachiappan, A. K. Mishra, M. Kademir, A. Sivasubramaniam, O. Mutlu, and C. R. Das "Application-aware prefetch prioritization in on-chip net-works," in Proceedings of the 21st international conference on Parallel architectures and compilation techniques, pp. 441–442, ACM, 2012.

[11] J. Lee, H. Kim, M. Shin, J.-H. Kim, and J. Huh "Mutually aware prefetcher and on-chip network designs for multi-cores," Computers, IEEE Transactions on, vol. 63, no. 9, pp. 2316–2329, 2014.