Dynamic Cache Partitioning Based on Software Hints

Grant Jinilang

ABSTRACT

1. INTRODUCTION

In traditional chip-multi-processors (CMPs), the last-level cache is shared among all cores and LRU is usually used as the replacement policy. Shared last-level cache is long konwn to suffer from interference among cores. The LRU policy implicitly partitions the shared cache among cores based on their requests for cache resources. The resulted cache partitioning among cores is unlikly to be optimal in terms of the total number of cache misses. This is because the marginal benefit gained from increasing cache space differ across cores. Limited cache resources should allocated to applications that may get more marginal benefits from them.

Application developers might be able to predict their applications' cache usage and thus request an optimal amoumt of shared cache for his application. For example, the application developer might know that his application is a streaming application. In that case, although the application demands a large amount of cache, the marginal benefits the application gains from the additional cache space is small. Thus he requests a small amount of cache according the application's working set size. So application programmers may provide hints for requesting shared cache space.

Compiler might be able to analyze the program at compile time and predict the application's working set size and thus make request accordingly. The operating sytem might be able to analyze the application's live trace and utilize more sophisticated algorithms to predict its working set size that what can be done in hardware. Thus the operating system might as well provide hints for requesting shared cache space.

We understand that generating those hints might be a hard problem itself. However, as a logical first step, we'd like to know whether those hints would be helpful assuming idea hints can be generated. Therefore, in this project, we performed a limit study and designed a cache partitioning mechanism based on software hints to investigate how software hints may be utilized in cache partitioning to minimize cache misses. We believe our work can serve as the foundation of future works that intend to investigate cache partitioning based on software hints.

2. BACKGROUND AND RELATED WORKS

There have been serveral previous works that explored dynamic cache partitioning to improve the scalability of the multi-core processors. [Qureshi and Patt(2006)] proposed a utility-based cache partitioning strategy. In their approach, each core is associated with a utility monitor (UMON). UMON has a tag directory which caches a tag per set per way. In order to reduce hardware overhead, UMON may cache only one tag for the same cache way in all sets. By using UMON, a cache miss line can be obtained, which depicts the relationship between number of cache misss and the number of ways assigned to this core. By greedily assigning each cache way to the core where the cache utility can be maximized (reducing most cache misses), this approach partitions the cache to maximize utility (minimize cache misses). This is called look ahead algorithm.

[Qureshi et al.(2006)Qureshi, Lynch, Mutlu, and Patt] pointed out that cache misses are not equivalently expensive. Parallel misses are much cheaper than isolated misses because they can be served in parallel. Instead of minimizing the number of cache misses, a better cache partitioning strategy might be minimizing MLP-based cache cost. This strategy was explored by [Moreto et al.(2007)Moreto, Cazorla, Ramirez, and Valero] which demonstrated MLP-aware cache partitioning indeed achieves better performance. Recently, [Beckmann and Sanchez(2013)] studied partitioning fully-associated cache to data and it reduced the cost of look-ahead algorithm by peek-ahead algorithm.

All previous works predict future cache usage based on previous cache miss curve. This works if the cache usage pattern remains similar across different phases of the program. When the memory usage pattern of the program changes across phases, the prediction might be inaccurate. In our project, we'd like to further improve cache partitioning based on program hints for future memory usage given by users, OS, or compiler.

[Ipek et al.(2008)Ipek, Mutlu, Martínez, and Caruana] describes a novel memory controller design which would use adaptive scheduling based on machine learning. Using reinforcement learning, their scheduler would optimize scheduling on the fly. Controller-state action pairs are assigned reward values, and when commands are issued, the controller tries to choose the command with the greatest long term value. A learning controller brings about some great benefits to program performance. Primary amongst these is that the controller optimize for bus bandwidth, and does so

on the fly. Many scheduling algorithms attempt provide the best bandwidth in the general case, though there often weak spots in their approaches which reduce memory throughput. By learning and being adaptable, the authorsâĂŹ scheme fights this weakness. In addition, the rewards system takes the core where the memory request originated from, allowing the scheme to fight against starvation as well. However the capabilities of the scheduling system are limited. While in theory, the machine learning algorithm they chose should be able to take into account infinitely many states and inputs, hardware and computation time hampers the scheduling optimization possibilities. While the authors optimized their algorithm for the resources they had available, there are various scenarios which they were not able to account for due to hardware limitations, creating weak spots in their system.

[Knauerhase et al.(2008)Knauerhase, Brett, Hohlt, Li, and Hahn] proposes a thread scheduling scheme to minimize LLC con-tention based on architectural observations made by the OS at run- time. While the paper does not discuss the modification of architectural features, it demonstrates how the OS can interface with the architecture to find optimize system behavior. In this particular instance, by leveraging features of the chosen architecture, the authors were able to track cache hit/miss ratios as well as absolute counts of hits and misses on caches in the system. Using these metrics, threads running in the system were assigned weights which corresponded to their cache usage. Threads were then scheduled on the cores in such a way that these weights were spread as evenly as possible across shared caches. The benefits to this approach are easily tangible. Scheduling threads in this way ensures that memory intensive processes are given an appropriate amount of resources rather than being forced to share an unnecessary and counter productive portion of cache. The feature we are proposing could possibly extend this scheme by allowing the OS to interface with shared caches to get better partitioning.

3. REFERENCES

[Beckmann and Sanchez(2013)] N. Beckmann and D. Sanchez. Jigsaw: Scalable software-defined caches. In *PACT*, pages 213–224. IEEE, 2013. ISBN 978-1-4799-1018-2. URL `http://dblp.uni-trier.de/db/conf/IEEEpact/pact2013.html#BeckmannS13`.

[Ipek et al.(2008)Ipek, Mutlu, Martínez, and Caruana] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 39–50, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3174-8. doi: 10.1109/ISCA.2008.21. URL `http://dx.doi.org/10.1109/ISCA.2008.21`.

[Knauerhase et al.(2008)Knauerhase, Brett, Hohlt, Li, and Hahn] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn. Using os observations to improve performance in multicore systems. *IEEE Micro*, 28(3):54–66, 2008. ISSN 0272-1732. doi: http://doi.ieeecomputersociety.org/10.1109/MM.2008.48.

[Moreto et al.(2007)Moreto, Cazorla, Ramirez, and Valero] M. Moreto, F. J. Cazorla, A. Ramirez, and M. Valero. Mlp-aware dynamic cache partitioning. In *PACT*, page 418. IEEE Computer Society, 2007. URL `http://dblp.uni-trier.de/db/conf/IEEEpact/IEEEpact2007.html#MoretoCRV07`.

[Qureshi and Patt(2006)] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechtanism to partition shared caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 423–432, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2732-9. doi: 10.1109/MICRO.2006.49. URL `http://dx.doi.org/10.1109/MICRO.2006.49`.

[Qureshi et al.(2006)Qureshi, Lynch, Mutlu, and Patt] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A case for mlp-aware cache replacement. *SIGARCH Comput. Archit. News*, 34(2):167–178, May 2006. ISSN 0163-5964. doi: 10.1145/1150019.1136501. URL `http://doi.acm.org/10.1145/1150019.1136501`.