LCA: A Memory Link and Cache-Aware Co-Scheduling Approach for CMPs

Alexandros-Herodotos Haritatos
School of ECE
NTUA
aharit@cslab.ece.ntua.gr

Georgios Goumas
School of ECE
NTUA
goumas@cslab.ece.ntua.gr

Nikos Anastopoulos
School of ECE
NTUA
anastop@cslab.ece.ntua.gr

Konstantinos Nikas
School of ECE
NTUA
knikas@cslab.ece.ntua.gr

Kornilios Kourtis
Dept. of Computer Science
ETH
kkourt@inf.ethz.ch

Nectarios Koziris
School of ECE
NTUA
nkoziris@cslab.ece.ntua.gr

ABSTRACT

This paper presents LCA, a memory Link and Cache-Aware co-scheduling approach for CMPs. It is based on a novel application classification scheme that monitors resource utilization across the entire memory hierarchy from main memory down to CPU cores. This enables us to predict application interference accurately and support a co-scheduling algorithm that outperforms state-of-the-art scheduling policies both in terms of throughput and fairness. As LCA depends on information collected at runtime by existing monitoring mechanisms of modern processors, it can be easily incorporated in real-life co-scheduling scenarios with various application features and platform configurations.

Keywords

LCA; scheduling; contention aware

1. INTRODUCTION

Chip multiprocessors (CMPs) execute concurrently a large number of threads, which share hardware resources such as caches, memory links and memory controllers, This sharing can create resource contention, which can impact a thread's performance compared to isolated execution. Contention-aware *co-scheduling* attempts to mitigate the effects of co-execution and meet a mix of objectives including system throughput, fairness, QoS, or energy consumption.

Proposed contention-aware scheduling frameworks rely on an application classification scheme that predicts application interference under a co-execution scenario [5,7,8,11,12]. Our classification approach has the following novel objectives: (a) detect contention on both the shared memory link and LLC; (b) rely only on information that can be collected at runtime by existing monitoring mechanisms of modern processors; and (c) require no additional hardware support. LCA decreases average application interference and distributes co-execution penalties between applications in a much fairer way compared to state-of-the-art schedulers, such as the Linux scheduler or prior contention-aware schemes balancing LLC misses (LLC_MRB) [3,4] or memory link bandwidth (LBB) [3,10].

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.
ACM 978-1-4503-2809-8/14/08.
http://dx.doi.org/10.1145/2628071.2628123.

2. CLASSIFICATION SCHEME

We focus on contention situations that can occur on the memory link, the shared LLC or both. We identify the following four application classes:

- N: Applications that restrict their activity to their private part of the memory hierarchy or within the core. They create no contention to the shared resources and therefore no interference.
- C: Applications with heavy activity on the shared LLC. The impact of Cs co-execution is difficult to predict. Cache organization and replacement policies are expected to handle high activity from different applications on the LLC efficiently. However, if these exhibit similar data access patterns, contention will be high.
- LC: Applications with significant activity on both the LLC and the memory link. These cause some interefence and thus slowdowns to C applications. On the other hand, competition between multiple LCs is not high.
- L: Memory link intensive applications. Due to their streaming nature, they thrash the LLC which is catastrophic for C applications. When co-executing with an LC, the L will suffer some interference due to the LC's medium demands for memory link bandwidth, while the LC is expected to suffer higher slowdown due to both memory bandwidth shortage and cache thrashing by the L application. Finally, multiple Ls compete for the memory link and a slowdown is expected.

Our classification method is implemented as a decision tree. First we look at the memory link utilization to distinguish L from LC applications. When this is too low, we discern C from N applications using cache link utilization, reuse location and memory uops to total uops ratio. All data are collected at runtime by performance counters utilities.

3. SCHEDULING ALGORITHM

The interference pattern between the four classes imposes a prioritization in the formation of application pairs. Co-execution of $L - C$, $L - L$ and $L - LC$ needs to be avoided as much as possible. Our algorithm is a greedy one with $O(n)$ complexity that forms pairs in a predefined order. It starts by working on N applications and co-schedules them first with all the *Ls* (that impose the greatest harm), then with *Cs* (that suffer the greatest harm) and finally with *LCs*. Remaining *Cs* are first matched with *LCs* and if needed with *Cs*, and then remaining *LCs* are matched first with *Ls* and if needed with *LCs*. Finaly pairs are created from any remaining L applications.

4. EVALUATION

Evaluation is performed on an Intel® Xeon® CPU E5-4620 with 8 cores, private L1 and L2, 16MB 16-way shared L3 and 64GB memory. The platform runs Debian with kernel 3.7.10. All the schedulers are implemented in userspace.

Each class is populated with four benchmarks from a variety of suites [1,2,9]. All applications run with four threads and exhibit a single execution phase. To evaluate the performance of the system at full load, we define a time window, in which every application that terminates starts again.

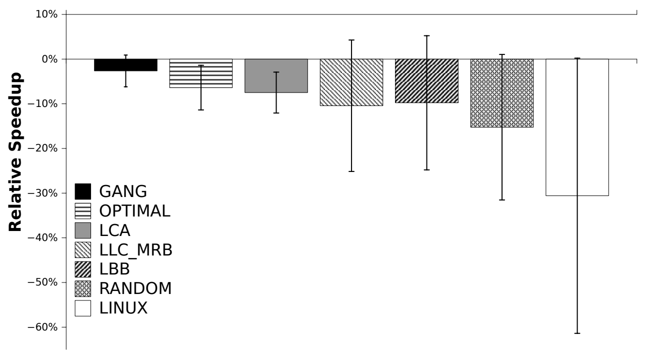

Figure 1: Average slowdown over standalone execution

Figure 1 illustrates the average speedups across all applications. Error bars indicate standard deviation. LCA exhibits similar performance to the optimal scheduler [6], which demonstrates the efficiency of our classification approach. More importantly, LCA exhibits significantly lower deviation than LLC_MRB, LBB and random and even optimal in some cases. This indicates that LCA is much fairer, as it manages to distribute co-execution penalties more uniformly across applications.

5. CONCLUSIONS AND FUTURE WORK

We have proposed a novel application classification scheme that inspects data traffic and resource utilization across the entire memory hierarchy. Based on this, we devised the LCA scheduling algorithm which alleviates the effects of resource contention. LCA outperforms all other schedulers both in terms of average application throughput, achieving performance levels that vary by $\pm 2\%$ with respect to optimal performance, and in terms of fairness as it distributes overheads across applications in a more uniform way.

As future work, we intend to extend this work in the following directions: a) handle more complex scheduling scenarios such as varying number of co-runners and I/O-bound applications, and b) incorporate our proposed scheduler in large-scale cloud environments for scheduling cloud workloads and managing data-centre resources efficiently.

6. ACKNOWLEDGEMENT

This research was funded by project I-PARTS: “Integrating Parallel Run-Time Systems for Efficient Resource Allocation in Multicore Systems” (code 2504) of Action ARISTEIA, co-financed by the European Union (European Social Fund) and Hellenic national funds through the Operational Program “Education and Lifelong Learning” (NSRF 2007-2013).

7. REFERENCES

[1] pChase benchmark. https://github.com/maleadt/pChase.

[2] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks - Summary and preliminary results. In *SC '91*, New York, NY, USA, 1991. ACM.

[3] Major Bhadauria and Sally A. McKee. An approach to resource-aware co-scheduling for CMPs. In *ICS '10*, pages 189–199, New York, NY, USA, 2010. ACM.

[4] Sergey Blagodurov, Sergey Zhuravlev, and Alexandra Fedorova. Contention-aware scheduling on multicore systems. *ACM Trans. Comput. Syst.*, 28(4):8:1–8:45, December 2010.

[5] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *HPCA '05*, 2005.

[6] Yunlian Jiang, Xipeng Shen, Jie Chen, and Rahul Tripathi. Analysis and approximation of optimal co-scheduling on chip multiprocessors. In *PACT '08*, New York, USA, 2008. ACM.

[7] Evangelos Koukis and Nectarios Koziris. Memory bandwidth aware scheduling for SMP cluster nodes. In *PDP '05*. IEEE, 2005.

[8] Jiang Lin, Qingda Lu, Xiaoning Ding, Zhao Zhang, Xiaodong Zhang, and P. Sadayappan. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In *HPCA '08*, 2008.

[9] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, December 1995.

[10] Andreas Merkel, Jan Stoess, and Frank Bellosa. Resource-conscious scheduling for energy efficiency on multicore processors. In *EuroSys '10*, USA, 2010. ACM.

[11] Lingjia Tang, Jason Mars, and Mary Lou Soffa. Contentiousness vs. sensitivity: improving contention aware runtime systems on multicore architectures. In *EXADAPT '11*, USA, 2011. ACM.

[12] Yuejian Xie and Gabriel Loh. Dynamic classification of program memory behaviors in CMPs. In *CMP-MSI '08*, China, 2008.