Coloring the Cloud for Predictable Performance

Alberto Scolari, Filippo Sironi, Davide B. Bartolini, Donatella Sciuto, Marco D. Santambrogio
Politecnico di Milano

Motivation and Contribution The commodity multicores that power cloud infrastructures hide memory latency through deep memory hierarchies, with the last-level cache (LLC) usually shared among cores. While a shared LLC improves utilization of on-chip resources, it may also lead to unpredictable performance of co-located virtual machines (VMs) as a result of unanticipated contention. Past research showed that the operating system page allocator can favor performance predictability on a physically-addressed shared LLC through page coloring [4, 8, 9]: a software technique that can work on commodity multicores, unlike hardware approaches [2, 7]. The main drawback of page coloring is the high cost of modifying allocations (i.e., recoloring), making this technique almost impractical for applications with varying memory footprints [6].

We aim at finding the simplest technique applicable to commodity multicores to avoid unpredictable performance of co-located VMs. Since VMs have a bounded memory footprint (i.e., stated at deployment time), we can leverage page coloring and avoid the cost of recoloring. We designed and implemented *Rainbow*: a page allocator exploiting page coloring to expose a new VM configuration knob: cache allocation, which implicitly determines proportional memory allocation. This knob provides users with predictable performance and clouds with a safe way of co-locating VMs.

Work Done We build on the kernel virtual machine (kvm) hypervisor, based on Linux 3.9, by extending the page allocator to support page coloring. In Linux, the page allocator uses a buddy data structure [5]: an array of lists tracks free pages, where the k-th list contains *buddies* (i.e., sets) of 2^k contiguous pages.

We define *physical colors* (*pcolors*) and *virtual colors* (*vcolors*) as partitions of the LLC. The maximum number of pcolors of a platform is $N_p = \frac{C}{W \cdot P}$, where

SOCC '13, Oct 01-03 2013, Santa Clara, CA, USA
ACM 978-1-4503-2428-1/13/10.
http://dx.doi.org/10.1145/2523616.2525955

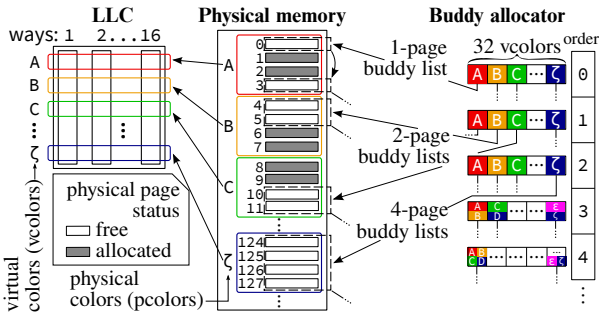

Figure 1. Modified buddy data structures.

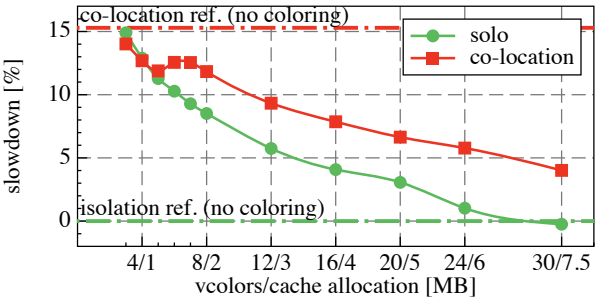

Figure 2. Slowdown of 2-threaded *canneal* running in solo and co-located with 2-threaded *streamcluster* on a 4-core processor with many vcolor allocations.

C, W, and P are the LLC size and associativity and the page size, respectively. For example, an Intel Xeon Processor W3530 has $N_p = 128$. A vcolor is a set of 2^n contiguous pcolors, with $n \in [0, \log_2 N_p]$; this concept abstracts over pcolors, enforcing architectural constraints (i.e., filling the private L2 cache). We modified the buddy data structure, breaking the lists into sublists, as in Figure 1. When a VM triggers a page fault, the hypervisor searches for free pages in the appropriate sublists; freed pages will be eventually added back to the right sublist.

Work in Progress We are finalizing our implementation to evaluate *Rainbow* with VMs running representative cloud workloads [3]. Figure 2 plots a preliminary validation with the PARSEC 2.1 benchmark suite [1].

Open Issues Non-overlapping vcolor allocations effectively partition the LLC; however, they may also impair utilization. We could support overlapping vcolor allocations to improve utilization by co-scheduling co-located VMs to ensure that those with overlapping vcolor allocations never execute at the same time.

Rainbow lacks support for huge pages due to an inherent limitation of page coloring; we will evaluate the performance penalty versus commodity hypervisors.

References

[1] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, Jan. 2011.

[2] H. Cook, M. Moreto, S. Bird, K. Dao, D. A. Patterson, and K. Asanovic. A Hardware Evaluation of Cache Partitioning to Improve Utilization and Energy-efficiency While Preserving Responsiveness. In *Proceedings of the 40th International Symposium on Computer Architecture*, ISCA '13, pages 308–319, New York, NY, USA, 2013. ACM. doi: 10.1145/2485922.2485949.

[3] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 37–48, New York, NY, USA, 2012. ACM. doi: 10.1145/2150976.2150982.

[4] R. E. Kessler and M. D. Hill. Page Placement Algorithms for Large Real-indexed Caches. *ACM Trans. Comput. Syst.*, 10(4):338–359, Nov. 1992. doi: 10.1145/138873.138876.

[5] K. C. Knowlton. A Fast Storage Allocator. *Commun. ACM*, 8(10):623–624, Oct. 1965. doi: 10.1145/365628.365655.

[6] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems. In *Proceedings of the 14th International Symposium on High Performance Computer Architecture*, HPCA '08, pages 367–378, Washington, DC, USA, 2008. IEEE Computer Society. doi: 10.1109/HPCA.2008.4658653.

[7] D. Sanchez and C. Kozyrakis. Vantage: Scalable and Efficient Fine-grain Cache Partitioning. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 57–68, New York, NY, USA, 2011. ACM. doi: 10.1145/2000064.2000073.

[8] T. Sherwood, B. Calder, and J. Emer. Reducing Cache Misses Using Hardware and Software Page Placement. In *Proceedings of the 13th International Conference on Supercomputing*, ICS '99, pages 155–164, New York, NY, USA, 1999. ACM. doi: 10.1145/305138.305189.

[9] X. Zhang, S. Dwarkadas, and K. Shen. Towards Practical Page Coloring-based Multi-core Cache Management. In *Proceedings of the 4th European Conference on Computer Systems*, EuroSys '09, pages 89–102, New York, NY, USA, 2009. ACM. doi: 10.1145/1519065.1519076.