Automatic Hardware Execution Throttling for Multi-core Resource Management

Xiao Zhang (student) Rongrong Zhong (student) Sandhya Dwarkadas Kai Shen
Department of Computer Science, University of Rochester

With the emergence of cloud computing environments and the dominance of multi-core chips on today's server market, performance isolation and QoS for multiprogrammed workloads on this kind of platform has recently been in the limelight in the research community.

A significant fraction of the proposals focus on applying cache partitioning mechanisms to address the problem of performance or resource guarantees. Cache partitioning, either by hardware or software page coloring, effectively prevents inter-thread cache conflicts but at the expense of limiting cache sharing opportunities. In addition, hardware-based cache partitioning usually requires modifying the hardware cache block replacement policy and may involve additional design complexities. Page coloring partitions the cache at a page granularity, which is fairly coarse, and also suffers from other drawbacks such as the expense of page recoloring/copying in software and a potential for artificial memory pressure due to address range constraints imposed by page coloring [3].

Recent studies [1, 2] advocate using an existing hardware throttling mechanism — *duty cycle modulation* — for multi-core resource management. Duty cycle modulation is a per-core based hardware feature that can specify the fraction of time that a CPU is halting during non-duty cycles. It does not require significant modification on operating systems and cost of configuration is low (hundreds of cycles to read and write a register). These properties make it a good choice for multi-core resource management. However, identifying the appropriate duty-cycle configuration for an n-core platform is challenging because each core can have multiple duty-cycle levels (*i.e.* 8 on Intel processors) and the configuration space grows exponentially with the number of cores.

In this work, we propose a model-driven approach to determine the appropriate duty-cycle configuration given a user-specified service-level agreement (SLA). The SLA could be a fairness bound that enforces equal or proportional progress for concurrently executing programs, or a relative performance guarantee for a particular application. The goal of our model is to predict configurations that meet the SLA constraint meanwhile also optimize overall performance.

Our model utilizes a set of reference configurations, whose performance is already known through past measurements, to estimate the per-core performance at any duty cycle configuration of the system. The per-core performance can then be used to predict system-wise metrics.

With a given set of reference configurations with known performance, the model will make performance predictions at all remaining unobserved configurations and choose the best one. Since the resulting configuration will be chosen for execution, actual performance measurement on the new configuration can be made and the result can be added to the set of references. Because the new reference sample was a chosen "best" configuration and likely to be close to the optimal, it may enable more accurate model predictions and selection of a better configuration. This is an iterative refinement process.

The refinement ends when a predicted best configuration is already in the reference set (and therefore would not lead to a growth of the reference set or better modeling). The iterative refinement aspect of our approach is illustrated in the top portion of Figure 1 (undotted part).

Our per-core performance estimation is based upon a cache performance model. Normally speaking, in a cache resource competing environment, if an application $\mathcal{A}$ accesses the shared cache more frequently (*e.g.* its duty-cycle ratio is increased), its data reuse distance (in time) is shorter. If other applications maintain their original data reuse distances (in time), then $\mathcal{A}$ would likely to see a reduction in its cache miss ratio under an LRU-like cache replacement policy.

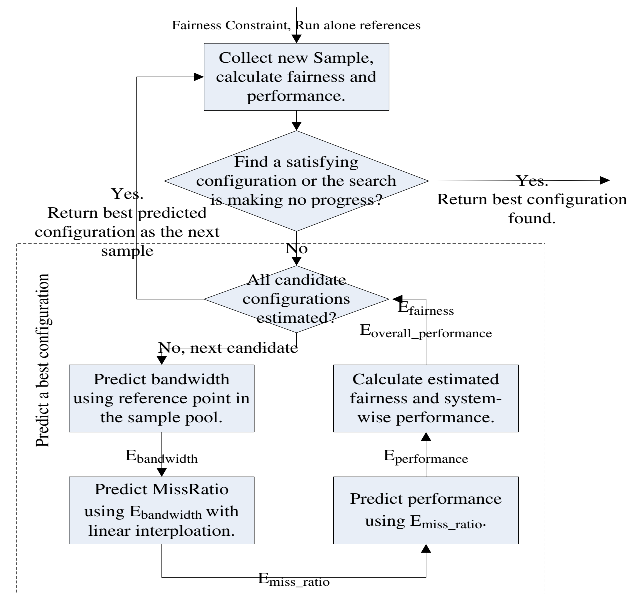

Figure 1. Control flow graph for our model-driven iterative refinement approach. The top portion illustrates the iterative refinement process. The bottom portion (within the dotted-line box) describes the performance estimation model that is invoked at each step of the refinement.

Given such a relationship between application cache access speed and its miss ratio, our model proceeds in three steps—estimating the application cache access speed (or reference bandwidth), modeling its cache miss ratio, and finally deriving its overall performance. The overall control flow of our model is illustrated in the bottom portion (within the dotted-line box) of Figure 1.

Our results on an Intel "Nehalem" quad-core machine shows that our model usually converges to an optimal system dutycycle configuration within 5 samples out of a searching space of 369 possible configurations.

References

[1] A. Herdrich, R. Illikkal, R. Iyer, D. Newell, V. Chadha, and J. Moses. Rate-based qos techniques for cache/memory in cmp platforms. In *23rd International Conference on Supercomputing (ICS)*, Yorktown Heights, NY, June 2009.

[2] X. Zhang, S. Dwarkadas, and K. Shen. Hardware execution throttling for multi-core resource management. In *USENIX Annual Technical Conf. (USENIX)*, Santa Diego, CA, June 2009.

[3] X. Zhang, S. Dwarkadas, and K. Shen. Towards practical page coloring-based multicore cache management. In *Proceedings of the Fourth EuroSys Conference*, Nuremberg, Germany, April 2009.