Author Retrospective for
Cooperative Cache Partitioning for Chip Multiprocessors

Jichuan Chang
Google, Inc.
jichuan@google.com

Gurindar S. Sohi
University of Wisconsin-Madison
sohi@cs.wisc.edu

ABSTRACT
In this paper, we reflect on our experiences and the lessons learned in designing and evaluating the cooperative cache partitioning technique for chip multiprocessors.

Original paper: http://dx.doi.org/10.1145/1274971.1275005

Categories and Subject Descriptors
B.3.2 [**Memory Structures**]: Design Styles – *Cache Memory.*
C.4 [**Performance of Systems**]: Design Studies.

Keywords
Multicore, last-level cache, cache partitioning, fairness, QoS.

1. BACKGROUND
The 1990s were what have been referred to as the Golden Age of Microarchitecture. There were many innovations in the basic microarchitecture of a uniprocessor, enabled by the increasing number of transistors provided by Moore's Law. Classical uniprocessor organizations were totally transformed as a result of these innovations. At the turn of the decade, chips transitioned from uniprocessors to chip multiprocessors (CMPs). The initial organization (or microarchitecture) of the "multiprocessor portion" of a CMP, i.e., the hardware that was not in the processor cores (e.g., shared caches, interconnect) still resembled a canonical symmetric multiprocessor (SMP). It was clear to the second author that continuing transistor bounty could be used to rethink the microarchitecture of the multiprocessor portion of a CMP. Specifically, since the designers of a CMP had complete control of what hardware it contained and how it would function, they could contemplate and implement techniques that could never be practical if they required interactions between distinct chips over which the designers may not have complete control, as was the case in canonical SMPs. Since caches accounted for a significant portion of this hardware, rethinking the organization and functioning of caches in CMPs was a logical place to start.

Our first foray into different cache operations, albeit in the SMP context, was Coherence Decoupling [1], which targeted the latency of coherence misses. Here we proposed to separate two major operations that are needed for correct cache operation on a coherence miss: obtaining the accessed data, and obtaining the coherence permissions to the data. We observed that data could typically be accessed quicker than all the necessary coherence permissions had been obtained, so a processor could (speculatively) start working with the data while the permissions were still pending, with corrective actions in case the speculative access was incorrect. Encouraged by the promising results over here, we started to contemplate other novel mechanisms for optimizing cache operations.

About the same time, there was a lot of work in the community on optimizing the latency of on-chip caches using novel last-level cache organizations. Although the last-level caches at that time were the L2 caches, the proposed techniques are typically applicable to large on-chip caches (e.g., L3 cache in today's server processors). The NUCA cache work was an early proposal in the direction [2]. Another line of work observed that running parallel programs, or multiple programs, on the shared resources of a CMP introduced new problems due to interference in the shared resources. A significant body of work was already underway in trying to alleviate the negative impact of such interference.

2. DEVELOPING THE IDEAS
The plethora of ongoing work in optimizing L2 cache operation motivated us to try our own proposal for CMP cache organization. We wanted to achieve the latency benefits of private L2 caches, and also the capacity benefits of shared L2 caches. The idea was to use available capacity in another on-chip L2 cache when replacing a cache block from a private L2 cache, rather than evict the block entirely from the chip. We quickly became aware of similar concepts of cooperative caching proposed in file systems research. Further development of the basic ideas led to the "Cooperative Caching for Chip Multiprocessors" paper that appeared in *ISCA 2006*. CMP cooperative caching (CC) enabled fine-grained capacity sharing between private caches to combine the best attributes of both private and shared cache designs.

The observation that led to the work on Cooperative Cache Partitioning (CCP) was actually quite serendipitous. As we experimented with our CC design, we observed some interesting results. In particular, in an experiment with 4 cores running 4 copies of SPEC2000 benchmark `art`, the performance of CC was better than both the private and shared cache baselines. The contemporaneous work on cache partitioning showed that such a result was possible when co-scheduling different benchmarks, due to capacity sharing between applications with different cache footprints. However, it was not clear why this happened when running multiple copies of the same benchmark, which should have the same cache footprint.

Exploring further we observed the interaction of two factors. First, allocating slightly more cache resources to an application

ICS 25th Anniversary Volume, 2014
ACM 978-1-4503-2840-1/14/06.
http://dx.doi.org/10.1145/2591635.2591669

can disproportionately increase its performance while slowing down other co-scheduled applications only slightly. This is because for some applications the miss-rate (and thus performance) vs. capacity curve is non-linear and at certain points additional cache capacity can be very beneficial. This observation was also made by others at about the same time. Serendipitously, benchmark `art` happened to be such an application, and the caches size we were using was the point where `art` exhibited the behavior! Had we been experimenting with another benchmark, or with different cache parameters, we may not have observed the phenomenon. Second, an unintended consequence of CC was that the logically shared cache was being partitioned unequally at a given point in time. However, over time, the different threads each got a chance at an unequal partition, thus leading to overall higher performance, while not compromising fairness.

We were quite excited by these initial observations, since it suggested that CC was also a simple and effective cache partitioning technique, naturally addressing the issues of performance isolation and cache QoS. But as we further studied the related work and ran more experiments, it dawned on us that this was not a simple problem. There were cases where CC was adequate, but there were others where it was not. We needed to find a more systematic approach for leveraging CC for cache partitioning.

We posed two main challenges for ourselves. First, we wanted to develop a technique that would simultaneously achieve better throughput, fairness and QoS. Other proposals had to sacrifice some aspects to optimize others. Second, we felt the need to come up with effective evaluation metrics, since the related work lacked common metrics, measurement standards, baselines and representative co-schedules.

We spent a fair amount of time tuning CC to address the first challenge without much success. One day, Jichuan was discussing the problem with two other students in our research group: Philip Wells and Koushik Chakraborty. They asked: why only spatial partitioning? Wouldn't the time-sharing policies from conventional OS scheduling studies help? This was the spark for introducing temporal techniques to address the problem. The example that we were trying to understand—of 4 copies of `art`—was in a sense already exhibiting this phenomenon, albeit naturally, without explicit control on our part. With the normal CC mechanisms, one `art` thread got an unfairly large partition in one time epoch, and others in a different time epoch. We just needed to come up with a way to deliberately give an unfair share of the L2 cache to a thread in a time epoch, and add a scheme to control these actions so that the overall fairness and QoS would not be compromised.

Using this inspiration, we went back and formulated the basic idea of MTP (multiple time-sharing partitions). Initial experiments quickly revealed that always using MTP wasn't a good idea, since for many cases SSP (single space-sharing partition)─the common root of contemporary work in CMP cache partitioning, including CC─ was more effective. Now we needed mechanisms for using MTP when it was likely to be effective and defaulting to SSP otherwise. After a lot of experimentation devoted to understanding behavior in different situations, we fleshed out the detailed algorithm for profiling, determining and enforcing MTPs as well as the policy of when to fall back to the default baseline (CC). This phase of the research was actually the most laborious and took a very large amount of time and effort.

Kyle Nesbit and Jim Smith at University of Wisconsin were also working on partitioning CMP resources at that time. Although we took a different approach, Jichuan enjoyed hearty discussions with them on the selection of baselines and metrics.

3. LOOKING BACK AND AHEAD

Looking back, there were multiple factors that led to the work and ideas reported in the paper. With Cooperative Caching, we started out by considering cache organization techniques that only became viable in a CMP context. Our objective was to optimize memory access latency for a single program, by making effective use of aggregative on-chip cache resources and minimizing off-chip accesses. Quite serendipitously, when experimenting with CC we realized that it was also effective in partitioning the aggregative cache resources for different threads, a problem that was actively being pursued by others using different techniques. And then came the realization that we could also enhance the proposal by adopting time sharing techniques that were proposed to address related problems in different contexts. In summary, the outcome was a mix of a different way of looking at a classical problem, serendipity, and adapting a solution from a different problem area. This is an effective mix that has been the basis of, and is likely to continue to be the basis for, many innovative solutions to problems.

Looking forward, with processing systems becoming more complex with multiple interacting hardware and software components and an increasing number of software threads, hybrid solutions that employ both space- and time-sharing approaches are likely to be more successful than approaches that employ either approach in isolation. This will also require a lot of innovation in algorithms and techniques that can monitor situations dynamically and determine the appropriate mix of techniques to deploy at a given point in time.

4. REFERENCES

1. Jaehyuk Huh, Jichuan Chang, Doug Burger, Gurindar S. Sohi. Coherence decoupling: making use of incoherence. ASPLOS 2004. pp. 97-106.
2. Changkyu Kim, Doug Burger, Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. ASPLOS 2002, pp. 211-222.