A Case for NUMA-Aware Contention Management on Multicore Systems

Sergey Blagodurov
School of Computing Science
Simon Fraser University
Vancouver, BC, Canada
sergey_blagodurov@sfu.ca

Sergey Zhuravlev
School of Computing Science
Simon Fraser University
Vancouver, BC, Canada
sergey_zhuravlev@sfu.ca

Alexandra Fedorova
School of Computing Science
Simon Fraser University
Vancouver, BC, Canada
fedorova@cs.sfu.ca

Ali Kamali
School of Computing Science
Simon Fraser University
Vancouver, BC, Canada
ali_kamali@sfu.ca

ABSTRACT

On multicore systems contention for shared resources occurs when memory-intensive threads are co-scheduled on cores that share parts of the memory hierarchy, such as last-level caches and memory controllers. Previous work investigated how contention could be addressed via scheduling. A contention-aware scheduler separates competing threads onto separate memory hierarchy domains to eliminate resource sharing and, as a consequence, mitigate contention. However, all previous work on contention-aware scheduling assumed that the underlying system is UMA (uniform memory access latencies, single memory controller). Modern multicore systems, however, are NUMA, which means that they feature non-uniform memory access latencies and multiple memory controllers. We discovered that contention management is a lot more difficult on NUMA systems, because the scheduler must not only consider the placement of threads, but also the placement of their memory. This is mostly required to eliminate contention for memory controllers contrary to the popular belief that remote access latency is the dominant concern. In this work we quantify the effects on performance imposed by resource contention and remote access latency. This analysis inspires the design of a contention-aware scheduling algorithm for NUMA systems. This algorithm significantly outperforms a NUMA-unaware algorithm proposed before as well as the default Linux scheduler. We also investigate memory migration strategies, which are the necessary part of the NUMA contention-aware scheduling algorithm. Finally, we propose and evaluate a new contention management algorithm that is priority-aware.

Categories and Subject Descriptors

D.4 [**OPERATING SYSTEMS**]: Process Management Scheduling

General Terms

Algorithms, Design, Measurement, Performance

Keywords

Multicore processors, NUMA systems, shared resource contention, scheduling

1. INTRODUCTION

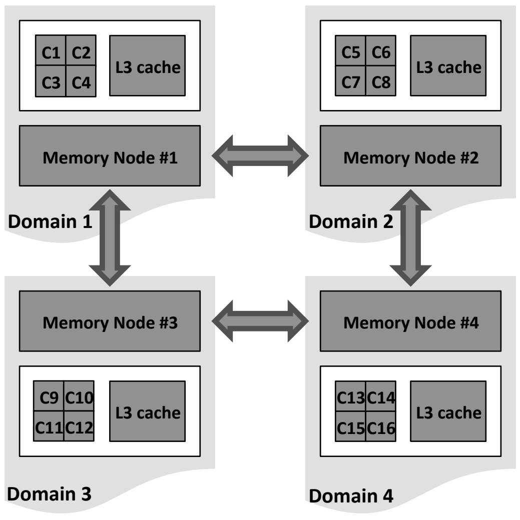

Figure 1: A schematic view of a system with four memory domains and four cores per domain. There are 16 cores in total, and a shared L3 cache per domain.

Contention for shared resources on multicore processors is a well-known problem. Consider a typical multicore system, schematically depicted in Figure 1, where cores share parts of the memory hierarchy, which we term *memory domains*, compete for resources such as last-level caches (LLC), system request queues and memory controllers. Multiple studies investigated ways of reducing resource contention and one of the promising approaches that emerged recently is contention-aware scheduling [4, 2, 1].

Although contention-aware scheduling algorithms are quite new, studies that evaluated them considered primarily centralized memory solutions typical for UMA (Uniform Memory Access) systems, where a single memory node, equipped with the single memory controller can be accessed with the same latency from any core. And yet, newer systems are increasingly turning to NUMA (Non-Uniform Memory Access) designs due to their decentralized and scalable nature. In modern NUMA systems, there are multiple memory nodes, one per memory domain (see Figure 1). Local nodes can be accessed in less time than remote ones, and each node has its own memory controller. We discovered that on NUMA systems the problem of contention-aware scheduling is a lot more complex, and simple strategies proposed in the past need serious revising.

Consider a workload of memory-intensive applications, i.e., applications that are characterized by a high rate of requests to main memory. Following the terminology adopted in an earlier study [3] we will refer to these applications as *devils*. Applications with a low rate of memory requests are referred to as *turtles*; a more precise definition of these classes is provided later. Previous studies claimed that given two devils it is simply sufficient to separate them into different domains to significantly mitigate resource contention. On NUMA systems, however, the scheduler must consider not only where the applications themselves are placed but also *the placement of their memory.*

Several observations obtained on the real NUMA servers lead us to this conclusion. The first and quite straightforward observation is that, if a scheduler migrates an application from domain A to domain B in an effort to reduce contention, it may subject the application to remote-access penalties if the memory of the application remains in A. The second more interesting observation, and one that is new to this work, is that even if competing threads themselves are placed on different memory domains, *they would still experience contention if their memory is placed on the same node.* The contended resource in this case would be the node's memory controller. We discovered that contention for memory controllers is the *dominant* cause of performance degradation, compared to contention for other resources. Therefore, not only must memory-intensive applications themselves be separated on different memory domains, but so must be their memory. While this idea is simple, implementing NUMA-aware contention management in practice may be difficult, because to avoid contention the scheduler not only needs to migrate the threads, but also their memory. Memory migration incurs overhead. Our study for the first time investigates how to reap the benefits of contention-aware scheduling on NUMA systems while avoiding excessive memory migration overheads.

Based on our findings we implemented a scheduling algorithm DI-NUMA (Distributed Intensity with NUMA awareness) that mitigates resource contention in a NUMA-aware fashion. DI-NUMA is an improvement over a NUMA-unaware DI algorithm, which was proposed in earlier work [4]. Figure 2 shows the average performance across ten runs. These experiments clearly make the case for NUMA-aware contention management on modern multicore systems.

We also observe that in the presence of contention it is difficult to reduce contention significantly for *all* threads in the workload. However, if some threads are deemed high-priority, the scheduler can improve their performance significantly by preventing them from contending with other applications. This ensures QoS and provides performance isolation for "important" applications. We implement this concept in an NUMA-aware DIP algorithm (Distributed Intensity with Priorities). The experimental results of testing DIP are provided on Figure 3. The applications that were prioritized are shown in capital letters. We used scientific applications from SPEC CPU 2006 and SPEC OMP 2001 in our experiments, focusing on those that extensively use the memory hierarchy of the system. In our experiments, DIP provides a significant performance boost for the prioritized devils by up to 35% relative to DI-NUMA.

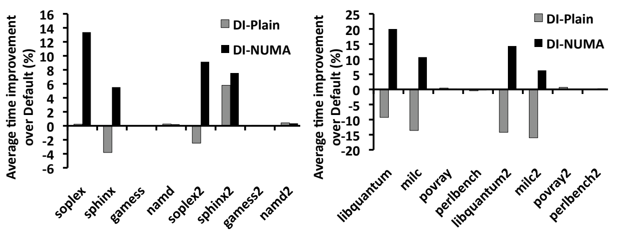

Figure 2: Average time improvement of DI-NUMA and DI-Plain over Default Linux scheduler.

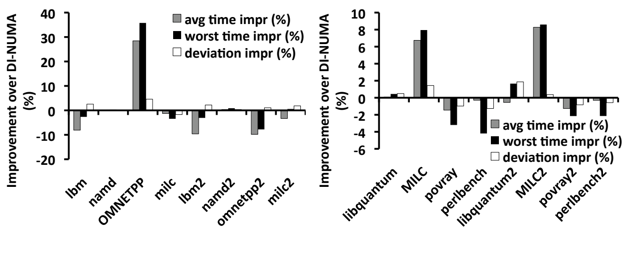

Figure 3: DIP-NUMA improvement over DI-NUMA for NUMA machine with 2 nodes

The contributions of this work in relation to previous work on contention aware scheduling are as follows: (1) We quantify the factors contributing to performance degradation on NUMA, as opposed to UMA, systems and find that memory controller contention is crucial to optimize. (2) To alleviate the memory controller contention, we redesign an existing contention-aware algorithm, such that it works on NUMA systems and delivers significantly better performance relative to NUMA-unaware contention-management algorithms. (3) Finally, we develop a new algorithm that relies on the observation that contention mitigation is especially powerful in reducing the worst-case execution times of chosen applications. This algorithm, Distributed Intensity with Priorities, is for the first time presented and evaluated in this work.

2. REFERENCES

[1] G. Dhiman, G. Marchetti, and T. Rosing. vGreen: a System for Energy Efficient Computing in Virtualized Environments. In *ISLPED*, 2009.

[2] 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.

[3] Y. Xie and G. Loh. Dynamic Classification of Program Memory Behaviors in CMPs. In *Proc. of CMP-MSI, held in conjunction with ISCA-35*, 2008.

[4] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing Contention on Multicore Processors via Scheduling. In *ASPLOS*, 2010.