POSTER: Improving NUMA System Efficiency with A Utilization-based Co-scheduling

Younghyun Cho, Camilo A. Celis Guzman, and Bernhard Egger
Department of Computer Science and Engineering
Seoul National University, Seoul, South Korea
{*younghyun, camilo, bernhard*}@*csap.snu.ac.kr*

***Abstract*—This work proposes a co-scheduling technique for co-located parallel applications on Non-Uniform Memory Access (NUMA) multi-socket multi-core platforms. The technique allocates core resources for running parallel applications such that both the utilization of the memory controllers and the CPU cores are maximized. Utilization is predicted using an online performance prediction model based on queuing systems. At runtime, the core allocation is periodically re-evaluated and cores are re-assigned to executing applications. Experimental results show that the proposed co-scheduling technique is able to execute co-located parallel applications in significantly less total execution time than the default Linux scheduler and a conventional scalability-based scheduler.**

***Keywords*-Resource management, NUMA, runtime system**

I. Introduction

Modern NUMA systems comprise dozens of CPU cores and several memory nodes in order to provide sufficient computational power and memory bandwidth. However, as applications tend to be either compute- or memory-bound, only few applications can fully utilize the available CPU and memory resources of large-scale NUMA platforms [6].

A conventional approach for runtime core allocation is to estimate the performance scalability of parallel applications and give more cores to scalable applications in order to utilize CPU resources well [4]. This approach is simple to apply, but often fails to minimize total turnaround time when scalable compute-bound and unscalable memory-bound applications are executed simultaneously. Through proper co-scheduling of concurrently executing parallel applications, it is possible to reduce the total execution time of co-located applications that exhibit different performance characteristics and resource requirements.

In this work, we propose a utilization-based runtime co-scheduling technique for NUMA systems. Our technique is fully automatic, and requires no offline information about the applications. The aim is to allocate the proper amount of cores for applications and, at the same time, improve utilization of the memory system. We determine the degree of parallelism (DoP) (i.e., the number of allocated cores) of running parallel programs that maximizes the total utilization of every memory controller and CPU core in a NUMA multi-core platform. This core allocation policy differs from other scalability-based approaches [4] that consider only CPU utilization. The utilization prediction is based on an online performance model [3] employing queuing theory to estimate the memory response time. The prediction technique only requires querying of the performance counters in the last-level caches (LLC) and NUMA interconnection networks and are easily obtained on modern AMD/Intel NUMA systems. Once the DoP has been determined for all co-located applications, LLC performance counters are monitored and core resources periodically re-allocated if an over- or underuse of an LLC is detected.

We leverage dynamic work scheduling in the parallel application runtimes to dynamically adapt the number of assigned cores to the applications. Intercepting calls to the parallel runtime (in this work, we consider the `parallel for` construct of the GNU OpenMP runtime) allows us to dynamically co-schedule pre-compiled applications without any source code modifications.

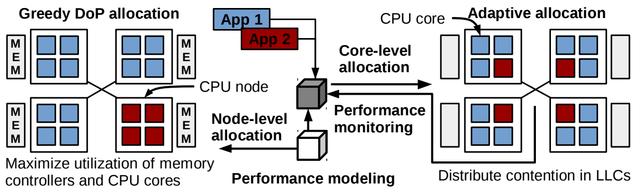

Figure 1. Overview of the co-scheduling framework.

II. Co-scheduling Technique

Figure 1 illustrates the proposed co-scheduling framework. A system comprises a number of memory and CPU nodes. A CPU node contains a number of cores located behind an LLC. Applications can be executed at any time without providing any information about the workload performance characteristics or its expected execution time.

Scheduling is performed periodically in epochs. In each epoch, the framework collects the number of requests to the memory controllers and the number of LLC accesses and misses of each application. Based on this information, core allocation is performed. At each scheduling epoch, the co-scheduler performs one of the following three different core allocation schemes, **clean-profiling allocation**, **greedy DoP allocation**, and **adaptive allocation** (Figure 2). First,

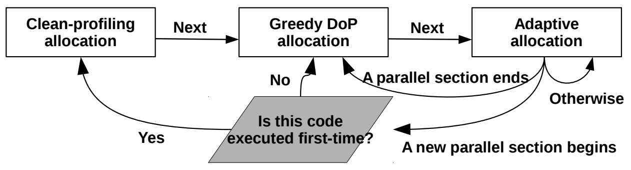
Clean-profiling allocation
Next
Greedy DoP allocation
Next
Adaptive allocation
No
A parallel section ends
Otherwise
Is this code executed first-time?
Yes
A new parallel section begins

Figure 2. The scheduling states.

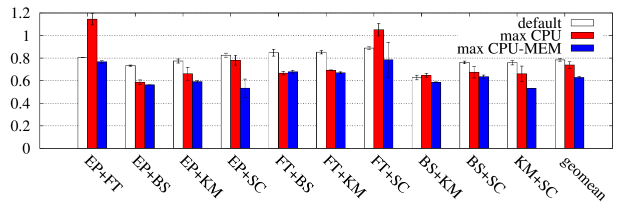

Figure 3. NTT of two co-located applications. EP and FT are from NPB [5], Blackscholes(BT) and Kmeans(KM) are from Parsec [1] benchmark suite, and Streamcluster(SC) is obtained from Rodinia [2].

whenever a parallel section is executed for the first time, the co-scheduler selects the **clean-profiling allocation** in which the new parallel section monopolizes all core resources for the next epoch. This allows us to collect stable memory access pattern because all cores behind an LLC execute the same program code.

After the first clean-profiling allocation, we perform **greedy DoP allocation**. Here, the proper DoP for running parallel programs is determined by a greedy algorithm. In this allocation, CPU nodes are allocated one by one to running applications. For each allocation, we predict the utilization metric for each possible allocation scenario (i.e., for each application in the system) by using the performance model in [3], then allocate the CPU node to the application that is expected to maximize system utilization.

Once the DoP allocation has completed, **adaptive allocation** is performed. In this state, we relax the restriction of CPU node level allocation and allow individual core re-allocations. We select the two CPU nodes that exhibit the highest (busy node) and lowest LLC accesses (free node) during the last epoch. We repeat this until all pairs are constructed among all CPU nodes in the NUMA system. If the ratio of LLC accesses between the CPU nodes in each pair is over a certain threshold, we select the application that has the highest LLC miss rate in the busy node and the application with the lowest LLC access rate from the free node, then exchange a number of cores of the two applications in order to distribute contention and to increase utilization of the LLCs. This adaptive allocation is repeated every epoch until a parallel section begins or ends.

III. Evaluation

A prototype of the proposed co-scheduling technique for Linux-based NUMA systems is evaluated on a 64-core (8-node) AMD Opteron 6380 platform with 8 memory nodes. The default scheduling epoch is set to 50ms, clean-profiling is performed for 150ms.

The proposed scheme, labeled **max CPU-MEM**, is compared against the default Linux scheduler, **default**, and a scalability-based scheduler where its core allocation policy is maximizing CPU utilization [4], called **max CPU**. Figure 3 shows the performance of the three different co-scheduling schemes. We consider the total turnaround time of all co-located applications (the makespan). We evaluate each scheduler using the normalized turnaround time (NTT) with regard to the total turnaround time in batch execution. Experimental results show that our co-scheduling technique is able to execute co-located OpenMP applications using a dynamic loop scheduler in less NTT than the default Linux scheduler and scalability-based scheduling by more than 20% percent on average for a wide number of benchmarks.

Acknowledgments

This work was supported in part by BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea (NRF) (Grant 21A20151113068), the Basic Science Research Program through NRF funded by the Ministry of Science, ICT & Future Planning (Grant NRF-2015K1A3A1A14021288), and by the Promising-Pioneering Researcher Program through Seoul National University in 2015. ICT at Seoul National University provided research facilities for this study.

References

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, Oct 2009.

[3] Y. Cho, S. Oh, and B. Egger. Online scalability characterization of data-parallel programs on many cores. In *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 191–205, Sept 2016.

[4] H. Sasaki, S. Imamura, and K. Inoue. Coordinated power-performance optimization in manycores. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, pages 51–61, Sept 2013.

[5] S. Seo, G. Jo, and J. Lee. Performance characterization of the nas parallel benchmarks in opencl. In *2011 IEEE International Symposium on Workload Characterization (IISWC)*, pages 137–148, Nov 2011.

[6] W. Wang, J. W. Davidson, and M. L. Soffa. Predicting the memory bandwidth and optimal core allocations for multi-threaded applications on large-scale numa machines. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 419–431, March 2016.