A Locality and Memory Congestion-aware Thread Mapping Method for Modern NUMA Systems

Poster · November 2018

CITATIONS
0

READS
191

4 authors:

Mulya Agung
Tohoku University
13 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Muhammad Alfian Amrizal
Tohoku University
19 PUBLICATIONS **23** CITATIONS
SEE PROFILE

Ryusuke Egawa
Tohoku University
110 PUBLICATIONS **366** CITATIONS
SEE PROFILE

Hiroyuki Takizawa
Tohoku University
162 PUBLICATIONS **858** CITATIONS
SEE PROFILE

A Locality and Memory Congestion-aware Thread Mapping Method for Modern NUMA Systems

Mulya Agung
Graduate School of Information Sciences, Tohoku University
Sendai, Miyagi
agung@dc.tohoku.ac.jp

Muhammad Alfian Amrizal
Research Institute of Electrical Communication, Tohoku University
Sendai, Miyagi
alfian@ci.cc.tohoku.ac.jp

Ryusuke Egawa
Cyberscience Center, Tohoku University
Sendai, Miyagi
egawa@tohoku.ac.jp

Hiroyuki Takizawa
Cyberscience Center, Tohoku University
Sendai, Miyagi
takizawa@tohoku.ac.jp

ABSTRACT

On modern NUMA systems, the memory congestion problem could degrade performance more than the memory access locality problem because a large number of processor cores in the systems can cause heavy congestion on memory controllers. In this work, we propose a thread mapping method that considers the spatio-temporal communication behavior of multi-threaded applications to improve the locality and to reduce the memory congestion on modern NUMA systems. We evaluate the proposed method using NPB applications on a NUMA system. Experiments show that our proposed method can achieve up to 20% performance improvement compared with locality-based and balance-based methods.

1 INTRODUCTION

Thread mapping is an important step to achieve scalable performance on modern multi-core processors. These processors have on-chip memory controllers that form the base for NUMA (Non-uniform Memory Access) multi-processors. Each processor is a node. It consists of a group of processor cores that is physically associated with one or more memory controllers and memory devices. A communication is called a remote-access communication if it is performed by threads that are executed by the processor cores from different nodes. In NUMA systems, the remote-access communication is slower than the local-access communication.

Our previous work [2] shows that on modern NUMA systems, maximizing locality can reduce the performance of applications because it can increase the data traffic congestion on memory controllers. Some related works on thread mapping consider locality and balance of memory accesses by analyzing the spatial communication behavior of the threads in parallel applications [3]. However, to effectively reduce the memory congestion, considering the temporal communication behavior of the threads is necessary because the memory congestion only happens when multiple threads from different processor cores access the same node at the same time. In this work, we present a thread mapping method that considers both spatial and temporal communication behaviors to improve the locality and to reduce the memory congestion. This work is an extension of our previous work on MPI process mapping [2]. The extension includes a method to detect the implicit communications between threads, a weighted data clustering method to analyze the spatio-temporal behavior of the communications between threads, and an algorithm that uses the analysis result to optimize the mapping between threads and processor cores.

2 LOCALITY AND MEMORY CONGESTION-AWARE THREAD MAPPING

This work targets static thread mapping to address the locality and memory congestion problems caused by the communications between threads. In parallel applications based on shared memory parallel processing, such as OpenMP and Pthreads, communication between threads is performed implicitly by accessing the shared memory space. In this work, we consider a communication event between two different threads as two consecutive memory accesses by the threads to the same cache line. Two different threads that communicate each other are called a pair of threads. A collective communication such as broadcast is detected as a set of point-to-point communications.

Our proposed method is performed in three steps: first, we obtain the time-series communication events by running the target application using a tracing tool based on binary instrumentation. The tool detects communication from memory accesses of the threads at a granularity of 64 byte-wide memory blocks. Each communication event contains its unique identification number (event ID), the identification numbers of the pair of threads (thread IDs), and the timestamp of communication with μs resolution. Second, we identify groups of threads that frequently communicate at the same time by using a weighted k-means clustering method [1]. We use the number of communication events as the weights for the clustering because a larger number of concurrent communications from different threads indicate a higher risk of memory congestion. Given a set of communication timestamps $\{t_1, t_2, ..., t_n\}$ and a set of clusters $\{C_1, C_2, ..., C_k\}$, the clustering method aims to minimize the objective function $j = \sum_{i=1}^{k} \sum_{t \in C_i} Acomm_t \|t - \mu_i\|^2$, where k is the number of clusters, μ_i is the mean of timestamps in a cluster, and $Acomm_t$ is the number of communication in timestamp t.

Figure 1(a) shows the communication behavior of the CG application of NAS Parallel Benchmarks (NPB-OMP), where darker bars illustrate more communications. This figure shows that the number of unique communicating threads and the number of communications change during the application's execution, with the highest number of communications happen during the time around 43% of the total execution time. Figure 1(b) shows the clustering result of the CG application. The communication events are partitioned into three clusters, which are illustrated by different colors. We obtain the optimal number of clusters k by using the Bayesian Information Criterion. As shown in the figures, the communication events during the time around 43% of the total execution time are clustered

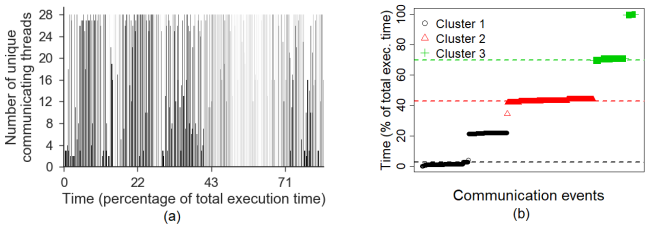

Figure 1: (a) The temporal communication behavior and (b) clustering result of CG

into Cluster 2. The number of communication events of this cluster is the highest among the clusters. It means that this cluster has the highest risk of memory congestion among the clusters.

Finally, we compute the mapping between threads and processor cores using an algorithm, called *Decongested Thread Locality* (DeTLoc). The topology information describing the number of nodes, the number of processor cores of each node, and the physical identities of processor cores are gathered by using a tool, called Hwloc. DeTLoc uses the clustering result to generate groups of thread pairs. Each group consists of thread pairs of the communication events that belong to the same cluster. Note that a thread pair can belong to multiple groups because the pair can communicate at different times. Since the mapping is static, the groups with more communications must take precedence over the other groups. To determine the order of the groups, the algorithm first calculates the load of a thread pair Wp by normalizing the $Acomm$ of the pair to its highest value, as defined by $Wp = \frac{Acomm_p}{\sum_{i=1}^{P} Acomm_i}$. Then the load of a group, Lc, is calculated by $Lc = \sum_{i=1}^{P_c} Wp_i$, where P is the total number of pairs in all groups and P_c is the total number thread pairs in the group c. The algorithm selects a thread pair that has not been mapped to cores sequentially from the groups with the highest to the lowest Lc, and from the pairs with the highest to the lowest Wp. Then, it improves the locality by mapping a thread pair to the same node, while also reducing the memory congestion by mapping the thread pairs in the same group to the different nodes. Note that a thread can belong to different pairs in one group. In such a case, avoiding the congestion may increase the number of remote-accesses. We will discuss this case in our future work.

3 PRELIMINARY EVALUATION RESULTS

We run experiments on a 2-node Intel-based NUMA system consisting of two Intel Xeon E5-2680v4 processors. Those nodes are connected with QuickPath Interconnect (QPI), and each node has 14 processor cores, 64 GB main memory, and an Integrated Memory Controller (iMC). We evaluate the proposed method using eight applications of the NPB, with 28 threads configuration, and problem size B. DeTLoc is compared with Packed, Scatter, Balance, and Locality methods. Both Packed and Scatter do not consider the communication behavior of the application. Packed maps the thread ID i to core ID i, while Scatter distributes two consecutive thread IDs to different nodes. Balance maximizes the communication balance among the nodes. It iteratively maps the unmapped thread with the highest number of communication to the node which currently has the lowest number of communication. Locality minimizes the

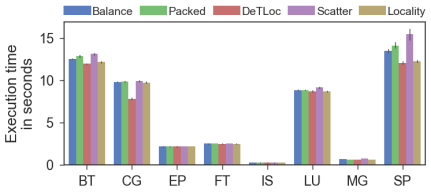

Figure 2: The average execution times of the NPB

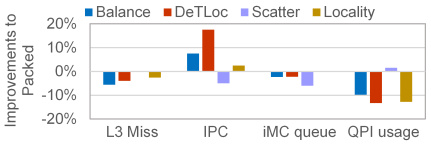

Figure 3: Performance monitoring results of CG

number of remote-access communications by mapping threads that frequently communicate to the same node. We use TreeMatch algorithm [4] to calculate the mapping for Locality.

Figure 2 shows that on average, DeTLoc can achieve shorter execution times compared to the other methods. In CG, DeTLoc achieves 20% performance improvements compared with the other methods. It can achieve the highest improvements for CG and SP because the average number of unique communicating threads and the ratio of communication to the memory accesses for CG and SP are the highest among the applications. We investigate the source of performance gain by monitoring the CG application using Linux Perf tool and Uncore module. A higher iMC queue indicates a longer queueing delay caused by the memory congestion, and a higher QPI usage indicates longer latencies from remote-access communications. Figure 3 shows the monitoring results that are normalized to the values of the Packed mapping. This figure shows that DeTLoc can achieve a higher instructions-per-cycle (IPC) by simultaneously reducing the queueing delay on the memory controllers and the number of remote-access communications.

4 CONCLUSIONS AND FUTURE WORK

The experiments show that our method can effectively improve the locality and reduce the memory congestion. We will evaluate the method with a larger number of nodes and different problem sizes. We also plan to evaluate the communication characteristics of applications that can benefit from memory congestion-aware thread mapping.

REFERENCES

[1] Margareta Ackerman, Shai Ben-David, Simina Brânzei, and David Loker. 2012. Weighted Clustering. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI'12)*. AAAI Press, 858–863.

[2] Mulya Agung, Muhammad A. Amrizal, Kazuhiko Komatsu, Ryusuke Egawa, and Hiroyuki Takizawa. 2017. A Memory Congestion-Aware MPI Process Placement for Modern NUMA Systems. In *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. 152–161.

[3] Matthias Diener, Eduardo H. M. Cruz, Marco A. Z. Alves, Philippe O. A. Navaux, and Israel Koren. 2016. Affinity-Based Thread and Data Mapping in Shared Memory Systems. *ACM Comput. Surv.* 49, 4 (Dec. 2016), 64:1–64:38.

[4] Emmanuel Jeannot, Guillaume Mercier, and Francois Tessier. 2014. Process Placement in Multicore Clusters: Algorithmic Issues and Practical Techniques. *IEEE Trans. Parallel Distrib. Syst.* 25, 4 (April 2014), 993–1002.