IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

DeLoc: A Locality and Memory-congestion-aware Task Mapping Method for Modern NUMA Systems

MULYA AGUNG[1], MUHAMMAD ALFIAN AMRIZAL[2], RYUSUKE EGAWA[3], AND HIROYUKI TAKIZAWA.[3]

[1]Graduate School of Information Sciences, Tohoku University, Sendai, 980-8578 Japan (e-mail: agung@dc.tohoku.ac.jp)
[2]Research Institute of Electrical Communication, Tohoku University, Sendai, 980-8577 Japan (e-mail: alfian@ci.cc.tohoku.ac.jp)
[3]Cyberscience Center, Tohoku University, Sendai, 980-8578 Japan (e-mail: {egawa, takizawa}@tohoku.ac.jp)

Corresponding author: Mulya Agung (e-mail: agung@dc.tohoku.ac.jp).

This work is partially supported by MEXT Next Generation High-Performance Computing Infrastructures and Applications R&D Program "R&D of A Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications" and Grant-in-Aid for Scientific Research(B) #16H02822 and #17H01706.

ABSTRACT The mapping of tasks to processor cores, called task mapping, is crucial to achieving scalable performance on multicore processors. On modern NUMA (non-uniform memory access) systems, the memory congestion problem could degrade the performance more severely than the data locality problem because heavy congestion on shared caches and memory controllers could cause long latencies. Conventional work on task mapping mostly focuses on improving the locality of memory accesses. However, our previous work showed that on modern NUMA systems, maximizing the locality can degrade the performance due to memory congestion. In this work, we propose a task mapping method that addresses the locality and the memory congestion problems to improve the performance of parallel applications. In the proposed method, first, the spatial and temporal communication behaviors of the applications are analyzed from the time-series dataset of communications among the parallel tasks. Then, a data clustering technique is employed to detect groups of tasks that potentially cause the memory congestion. Finally, this information is used to compute the task mapping to improve the locality and reduce the memory congestion. We also provide a set of metrics to describe the communication behaviors and to evaluate if the target application can benefit from our method. The proposed method is evaluated with the NPB and PARSEC applications on a real NUMA system and a multicore simulator. A detailed analysis of the sources of performance gain is also provided. Experimental results show that our method can achieve up to a 61% performance improvement compared with the state-of-the-art locality-based method.

INDEX TERMS high-performance computing, locality, memory congestion, NUMA, process mapping, task mapping, thread mapping.

I. INTRODUCTION

Task mapping is an important step in achieving scalable performance on modern multicore processors. These processors have on-chip memory controllers that form the base for NUMA (non-uniform memory access) multiprocessors. Each processor consists of a group of processor cores that is physically associated with one or more memory controllers and memory devices. This group of processor cores is referred to as a NUMA node [1]–[3]. Although the NUMA nodes are generally connected by high-speed interconnect links such as QuickPath Interconnect (QPI) [4] and HyperTransport [1], accessing a remote NUMA node still requires a longer latency than that required to access the data of the local NUMA node.

A parallel application consists of multiple *tasks*, each of which is executed on a processor core as a thread or a process. A unit of executing each task is a thread in shared-memory parallel processing, usually expressed with OpenMP directives, while it is a process in distributed-memory parallel processing, usually programmed with a message passing

interface (MPI). In NUMA systems, the communication between tasks is called local-access communication if it is performed by tasks that are executed by processor cores of the same NUMA node. On the other hand, it is called remote-access communication if it is performed by tasks that are executed by the processor cores from different NUMA nodes. Thus, remote-access communication needs a longer latency than does local-access communication.

On a NUMA system, mapping tasks of an application to processor cores, called *task mapping* [5], has a significant impact on the application's performance because the mapping affects the cost of the communications among tasks. During the execution of the application, a task does not necessarily need to communicate with all the others, and the cost of the communication depends on the physical location of the processor cores executing the communicating tasks. As the communication performance varies greatly on NUMA systems, use of the communication behavior to optimize the task mapping is important in increasing performance. These task mapping methods are called *communication-aware task mapping* [6]–[8].

In modern NUMA systems, the task mapping becomes more challenging because a large number of processor cores in a system can induce a large number of accesses to the memory devices, causing congestion on shared last-level caches and memory controllers. We refer to this congestion as *memory congestion*. As the number of processor cores increases, the number of communications that can simultaneously occur will also increase. When multiple communications among different tasks are in progress, they are referred to as *concurrent communications*. In the case of many concurrent communications, maximizing the locality will increase the memory congestion because the memory-access traffics will be concentrated more on particular NUMA nodes. In that case, it is important to manage the congestion by mapping the tasks of the concurrent communications to different NUMA nodes.

Most of the conventional approaches [9]–[13] to efficient task mapping have focused only on improving the locality of communication by mapping tasks, which frequently communicate with each other, to processor cores that are closer to each other in the memory hierarchy. Improving the locality of communications is important because it will reduce the remote-access penalty and congestion on interconnects. However, recent work [2], [14] and our previous work [15] have shown that on modern NUMA systems, maximizing locality does not always minimize the execution time. On the contrary, locality-based mapping might reduce the performance of applications due to the memory congestion.

Some related studies [16], [17] focus on improving the balance of the communication load among the NUMA nodes by analyzing the spatial communication behavior of the tasks. However, to effectively reduce the memory congestion, it is necessary to consider the temporal communication behavior of the tasks because the memory congestion occurs only when multiple tasks running on different processor cores access a particular NUMA node at the same time. Thus, to address the locality and memory congestion problems, it is necessary to consider both spatial and temporal communication behaviors of the application.

In this work, we present a task mapping method, called *decongested locality (DeLoc)*, that considers both the spatial and temporal communication behaviors of a parallel application to improve the locality and to reduce the memory congestion on modern NUMA systems. The method consists of a task mapping algorithm and several techniques to gather the NUMA node topology of the target NUMA system and to analyze the spatial and temporal communication behaviors of parallel applications. These techniques include a toolchain to obtain the time-series dataset of communications among tasks and a data clustering method to identify groups of tasks that potentially cause the memory congestion. For validation purposes, we have conducted experimental evaluations using a real NUMA system and a multicore simulator. In this paper, we provide the following contributions.

- A generalized task mapping method that detects implicit communications among tasks in shared-memory parallel processing. The preliminary study of this work focuses only on MPI process mapping [15].
- A weighted data clustering technique to analyze the spatial and temporal behaviors of communications among tasks and identify the communications that potentially cause the memory congestion.
- A mapping algorithm, called *decongested locality mapping (DeLocMap)*, for computing the task mapping that can address the locality and memory congestion problems.
- Metrics to describe the communication behaviors of parallel applications that affect the locality and memory congestion. These metrics can be used to evaluate whether a parallel application can benefit from locality and memory-congestion-aware task mapping.

The rest of the paper is organized as follows. We present the proposed method in Section II. The procedure of DeLoc is described in detail in this section. The metrics for characterizing the communication behaviors of applications are presented in Section III. To validate the proposed method, in Section IV, we analyze the experimental results using the proposed metrics. The related studies are described in Section V. The importance and effects of considering the temporal communication behavior are also discussed in this section in comparison with other existing approaches. Finally, Section VI gives the conclusions and future work of this paper.

II. A LOCALITY AND MEMORY-CONGESTION-AWARE TASK MAPPING METHOD

In this section, we describe DeLoc, which can address both the locality and memory congestion problems. The procedure of DeLoc is summarized as follows.

1) Gather the NUMA node topology information of the target system.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access

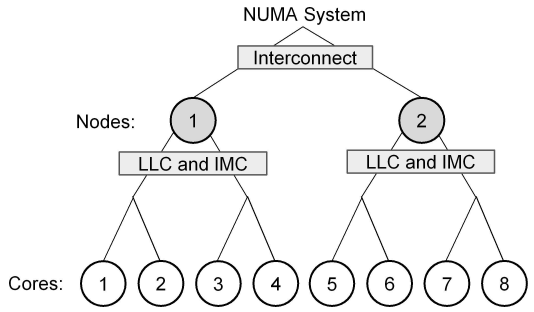


FIGURE 1: An example of the NUMA node topology

2) Analyze the spatial and temporal communication behaviors of the target application.
3) Identify the concurrent communications that potentially cause the memory congestion.
4) Compute a mapping between tasks and the processor cores using the DeLocMap algorithm.

A. GATHERING THE NODE TOPOLOGY INFORMATION OF THE TARGET SYSTEM

The first step is to retrieve information about the NUMA node topology of the target NUMA system. Some tools such as Hwloc [18] and numactl [19] are available for the information retrieval. The topology is modeled as a tree to express the information on the locations of shared last-level caches, memory controllers and interconnect links. This information is required because DeLoc focuses on reducing the amount of remote-access communication through interconnects and reducing the congestion on the shared caches and memory controllers. Note that in the NUMA systems considered in this work, such as Intel-based and AMD-based NUMA systems [1], each NUMA node is physically associated with a shared last-level cache (LLC) and an integrated memory controller (IMC). Thus, the location of memory controllers also represents the location of the last-level caches.

Figure 1 depicts an example of the model of a two-node NUMA system that consists of eight processor cores. The topology information also includes the identity information of the NUMA nodes and processor cores. The identity information is required later by the mapping algorithm to match the tasks with the processor cores. To obtain all of the information required in this step, the use of a specific tool is not mandatory.

B. ANALYZING THE COMMUNICATION BEHAVIORS OF A PARALLEL APPLICATION

The second step is to analyze the spatial and temporal communication behaviors of the target application. Since the communication behaviors may change during the execution, we dynamically analyze the communication behaviors by preliminary executing the application on a real system. We trace the communication events among tasks during the preliminary execution. Then, we create a time-series dataset of the communication events. This dataset represents the spatial and temporal communication behaviors of the application. Note that the tracing methods can be different for different parallel processing methods. This is because the method of task communication is dependent on the communication paradigm used by the parallel processing method [20]. In the next two subsections, we present methods to analyze the communication behaviors of parallel applications based on distributed-memory and shared-memory parallel processing.

1) Analyzing the Communication Behaviors of an Application Based on Distributed-memory Parallel Processing

In parallel applications based on distributed-memory parallel processing, such as the message passing interface (MPI) [21], a task is represented by a process, and each process has a unique identifier called a process ID. Thus, in the MPI, process mapping is also referred to as task mapping [22]. In the MPI, communication is explicit and is performed by sending and receiving messages. The process that sends the message is called a sender, and the process that receives the message is called a receiver. For each communication, there is a pair of sender and receiver processes. Thus, we can define a communication event as one message with its corresponding sender and receiver pair. This pair is also referred to as a *task pair*.

In the MPI, a process can communicate with other processes by using point-to-point (P2P) operations and collective operations. In P2P operations, a process sends messages to another process by explicitly specifying the ID of the receiver. On the other hand, rather than explicitly sending and receiving these messages, a collective operation involves communications among all processes in a communicator. Note that in an internal MPI, a collective operation is typically implemented using multiple P2P operations [23], [24]. Thus, in that case, each collective operation can be decomposed into P2P operations.

To analyze the communication behaviors of an MPI application, we preliminary run the target application and trace the communication events among the MPI processes. For this tracing purpose, we develop a low-level monitoring tool. The monitoring tool is implemented as a component of the monitoring framework, which was proposed in [25]. This framework is built on top of the point-to-point management layer (PML) of the Open MPI stack [26]. Since the PML can monitor point-to-point operations organizing a collective communication, the communication events can be traced in both point-to-point and collective communications. Furthermore, the PML monitors not only MPI_Send-MPI_Recv but also MPI_Isend-MPI_Irecv operations. Thus, the monitoring tool also traces both blocking and non-blocking communications. The tool generates the time-series dataset of communication events by recording the IDs of the sender and receiver, the timestamp, and the data size of each event.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access

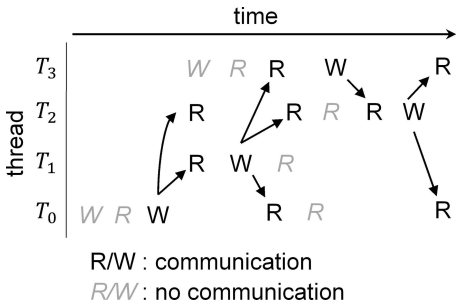


FIGURE 2: Memory accesses of different threads can be seen as their communications

2) Analyzing the Communication Behaviors of an Application Based on Shared-memory Parallel Processing

In parallel applications based on shared-memory parallel processing, such as OpenMP and Pthreads, a task is represented by a thread of execution, and each thread has a unique identifier called a thread ID. Thus, in shared-memory parallel processing, thread mapping is also referred to as task mapping [3]. The communication among threads is performed implicitly by accessing the shared memory space. By tracing accesses to memory addresses at the cache line granularity, we can define a communication event as two consecutive write and read memory accesses from different threads to the same cache line. These two memory accesses are referred to as *communicating memory accesses* [27]. Figure 2 shows communicating and non-communicating memory accesses for one cache line. The communicating memory accesses are shown in black, while non-communicating memory accesses are shown in gray. Two different threads that perform the communicating memory accesses are called a pair of threads, and this pair is also referred to as a task pair.

To analyze the communication behaviors of a parallel application based on shared-memory parallel processing, we preliminarily run the application and trace the memory accesses performed by the application threads. For this tracing purpose, we developed a tool. The tool is based on a dynamic binary instrumentation framework, called a pin [28]. The tool detects communication from memory accesses of the threads at a granularity of 64 byte-wide memory blocks. It generates the time-series dataset of communications by recording the IDs of the pair of threads, the timestamp, and the data size of memory access of each event.

C. IDENTIFYING CONCURRENT COMMUNICATIONS

In this section, we describe the step to identify the concurrent communications among tasks. After the time series data of communications of the target application are obtained, they are analyzed to identify the concurrent communications by using a weighted k-means clustering method [29]. We use the number of communication events as the weights for the clustering because a higher number of concurrent communications indicates 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 defined by equation (1), where k is the number of clusters, μ_i is the mean of timestamps in a cluster, and $Ncomm_t$ is the number of communication events in timestamp t.

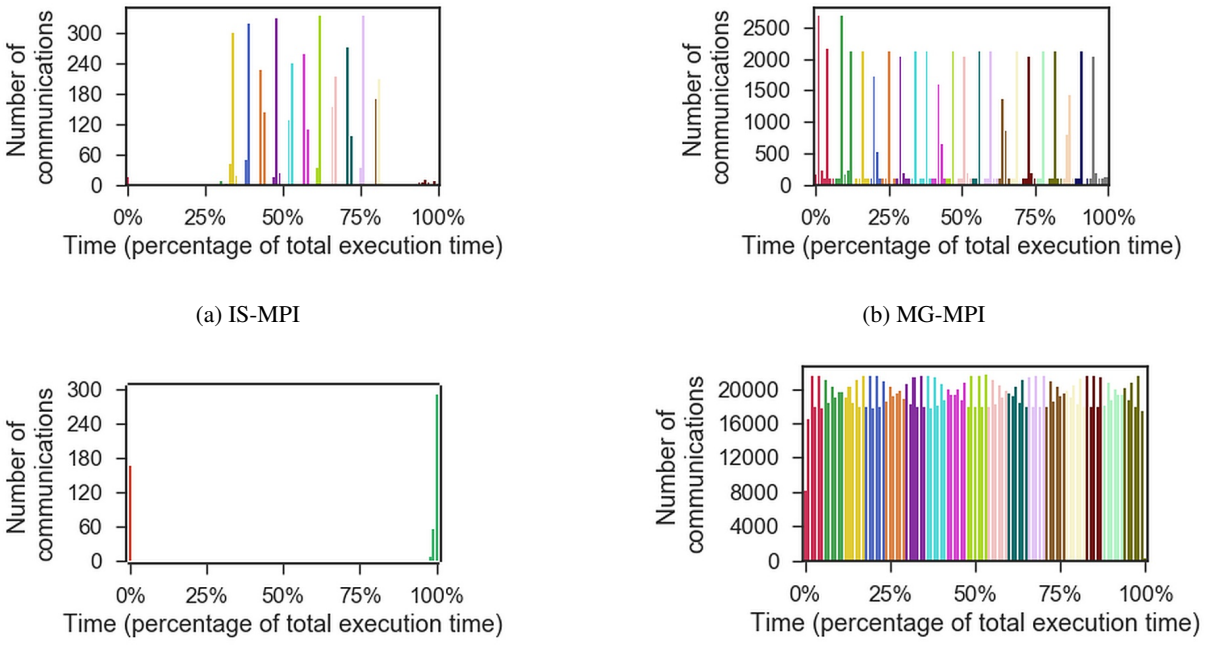
$$j = \sum_{i=1}^{k} \sum_{t \in C_i} Ncomm_t \|t - \mu_i\|^2. \quad (1)$$

The clustering method needs to predefine a parameter, k, to specify the number of clusters. However, the actual number of clusters is generally unknown in advance, even though the parameter certainly affects the clustering results. This parameter affects the accuracy of detecting the concurrent communications. If the value of k is too small, two communication events may be falsely identified as concurrent communications. Thus, it is important to estimate an optimal number of clusters for the clustering. In this work, we find k by using the Bayesian information criterion (BIC) [30]. We use the BIC because it has been empirically shown not only to find an optimal value of k but also to accelerate the clustering process for large datasets [31]. Since the number of communication events in long-running parallel applications can be very large, accelerating the clustering process becomes important.

Figure 3 shows the communication behaviors of the IS-MPI, MG-MPI, EP-MPI and LU-MPI applications of NAS parallel benchmarks (NPBs) [32], with the class C input size. The x-axis and y-axis show the time elapsed during the execution and the number of communication events, respectively. The figure shows that the number of communication events changes during the application's execution. In IS, the communications mostly occur between 30% and 81% of the total execution time. In EP, the communications occur only at the beginning and the end of the execution. This is because EP is an embarrassingly parallel application, and the communications are required only for the allreduce operation. On the other hand, the communications in MG and LU are distributed over the execution time.

The colors in Figure 3 show the clustering results of the four NPB applications, with different colors representing different clusters. The figure shows that the time periods that are close to each other and have a high number of communication events are likely to be grouped in the same cluster. The communication events that belong to the same cluster are identified as concurrent communications. In IS, the clusters with the highest number of communication events are shown in the middle of the execution. In MG, the clusters with the highest number of communication events are shown at the beginning of the execution. In EP, there are only two clusters, and the cluster shown at the end of the execution has the highest number of communication events. On the other hand, in LU, the number of communication events is distributed over all of the clusters. From Figure 3, we can see that a cluster can have a high number of communication events. If all of the communication events of this cluster

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access



(a) IS-MPI

(b) MG-MPI

(c) EP-MPI

(d) LU-MPI

FIGURE 3: The communication behaviors and clustering results of the NPB applications (colors represent clusters)

happen in the same NUMA node, then memory congestion will likely occur. Thus, the DeLocMap algorithm aims to prevent the memory congestion by distributing the concurrent communications of the same cluster over the NUMA nodes.

D. COMPUTING THE TASK MAPPING

The final step is to compute the mapping between tasks and processor cores using the DeLocMap algorithm. This algorithm, depicted in Algorithm 1, can compute a match between task IDs and processor core IDs using the node topology information and the clustering result.

The algorithm works as follows: first, DeLocMap uses the topology model to construct the map of processor core IDs and task IDs (Line 1). The keys of the map represent the IDs of processor cores available in the system, and each value represents the ID of the task associated with the key. At the beginning of the algorithm, each value is set to empty. Then, the algorithm uses the clustering results to generate groups of task pairs. This step is performed by the $buildTaskGroups$ function (Line 2). Each group consists of task pairs of the communications that belong to the same cluster. Note that a task pair can belong to multiple groups because the pair can communicate at different times. Since the mapping is static, the groups with the larger amount of communication must take precedence over the other groups. In this case, avoiding congestion may increase the amount of remote-access communication. We discuss this case in the experimental evaluation of Section IV-B.

To determine the order of the groups, we calculate two load metrics. The algorithm first calculates the load of a task pair Wp by normalizing the size of data exchanged by the task pair $S\mathrm{comm}$ to its highest value, as defined by Equation (2). $S\mathrm{comm}$ represents the amount of communication of the task pair. Then, the load of a group, Lc, is calculated by Equation (3), where P is the total number of pairs in all groups and P_c

Algorithm 1 The DeLocMap Algorithm.

Input: T {The topology tree}
Input: C {The clusters of communication events}
Output: M {The map of processor core IDs and task IDs}

1: $M \leftarrow createMap(T)$
2: $G \leftarrow buildTaskGroups(C)$
3: $sorted\_G \leftarrow sortByLc(G)$
4: $i \leftarrow 0$
5: $current\_node \leftarrow firstNode(T)$
6: **while** $i < size(G)$ **and** $isAvailable(M)$ **do**
7: $P \leftarrow sorted\_G[i]$
8: $sorted\_P \leftarrow sortByWp(P)$
9: $j \leftarrow 0$
10: **for** $j = 0$ to $size(sorted\_P)$ **do**
11: **if not** $mapped(sorted\_P[j])$ **then**
12: $mapPair(sorted\_P[j], current\_node, M)$
13: $current\_node \leftarrow nextNode(T)$
14: **end if**
15: **end for**
16: $i \leftarrow i + 1$
17: **end while**

is the total number of task pairs in the group c.

$$Wp = \frac{S\text{comm}}{\sum_{i=1}^{P} S\text{comm}_i}, \tag{2}$$

$$Lc = \sum_{i=1}^{P_c} Wp_i. \tag{3}$$

After calculating the load metrics, the algorithm selects a task pair that has not been mapped to processor cores sequentially from the groups with the highest to the lowest Lc and from the pairs with the highest to the lowest Wp. This selection is achieved by the sorting steps in the algorithm (Lines 3 and 8). A NUMA node is available for the mapping if it has one or more unmapped cores. The algorithm then maps each task pair to the processor cores that are currently available in the current NUMA node (Line 12). The current NUMA node is obtained by traversing the NUMA nodes in the topology tree (Lines 5 and 13). The algorithm aims to improve the locality by mapping two tasks of a pair to the same NUMA node while also reducing the memory congestion by mapping the task pairs in the same group to the different NUMA nodes.

III. COMMUNICATION BEHAVIORS THAT AFFECT THE LOCALITY AND THE MEMORY CONGESTION

In this section, we present five metrics to describe the communication behaviors that affect the locality of communication and the memory congestion. The purpose of these metrics is to determine if a parallel application can gain performance improvement from our proposed method. The first two metrics are *communication load* and *communication-to-memory ratio*. These two metrics are used to describe the communication behaviors that can benefit from communication-aware task mapping. The two other metrics are called *communication concurrency* and *DRAM-to-memory ratio*. In conjunction with the previous metrics, these two metrics are proposed to describe the communication behaviors that can benefit from memory-congestion-aware task mapping. The last metric, called *communication locality*, is proposed to describe the communication behavior that can benefit from locality-based task mapping.

An improvement according to a specific communication-aware task mapping method depends on how much tasks are communicating. The improvement is expected to be higher for parallel applications, in which the total amount of transferred data is larger. To describe the load of communication, we use the $L\text{comm}$ metric, defined as the total amount of communication by all tasks. $L\text{comm}$ is calculated by

$$L\text{comm} = \sum_{i=1}^{T}\sum_{j=1}^{T} S\text{comm}[i][j], \tag{4}$$

where T is the total number of tasks and $S\text{comm}[i][j]$ is the amount of communication between a pair of tasks i and j. However, the load of communication itself is not sufficient to evaluate if an application will gain performance benefit from communication-aware task mapping. If the number of non-communicating accesses is a lot higher than the number of communicating accesses, a communication-aware task mapping method might not significantly affect the overall memory access behavior. For this reason, we define the communication-to-memory ratio metric $CommR$, which is a ratio of the load of communication to the total size of memory accesses of the tasks. $CommR$ is calculated by

$$CommR = \frac{L\text{comm}}{\sum_{i=1}^{T} MemV[i]}, \tag{5}$$

where $MemV[i]$ is the size of memory accessed by task i. The expected performance gains are higher for parallel applications that have higher values of $L\text{comm}$ and $CommR$.

For communication-aware task mapping methods that aim to reduce the memory congestion, it is necessary to evaluate how the communication among tasks affects the memory congestion. Even if the load of communication is high, the task mapping method might not give a performance benefit if most of the communication events do not occur simultaneously. In addition, the communication events may not access the memory controllers. If tasks have much more memory access to the cache memory than to the DRAM, a communication-aware task mapping method might not affect the congestion on memory controllers. In that case, the task mapping can affect the congestion on the shared caches. For these two reasons, we introduce the communication concurrency and DRAM-to-memory ratio metrics.

Communication concurrency ($CommC$) is defined as the average number of tasks per group. It is calculated by

$$CommC = \frac{\sum_{g=1}^{G} TaskN[g]}{T \cdot G}, \tag{6}$$

where G is the total number of groups and $TaskN[g]$ is the number of tasks in group g. These groups are obtained from the $buildTaskGroups$ function of the DeLocMap algorithm.

The DRAM-to-memory ratio ($DramR$) is defined as the ratio of the number of DRAM accesses to the total number of memory accesses. $DramR$ is calculated by

$$DramR = \frac{\sum_{i=1}^{T} DramV[i]}{\sum_{i=1}^{T} MemV[i]}, \tag{7}$$

where $DramV[i]$ is the size of the DRAM accesses performed by task i. A communication-aware mapping method will have higher impacts on the memory congestion of parallel applications that have higher values of $CommR$, $CommC$ and $DramR$.

For communication-aware task mapping methods that aim to improve the locality of communication, it is necessary to have a high variance in the amount of communication per task pair. This variance is necessary because the locality-based mapping focuses on mapping tasks that have a larger amount of communication than that of other tasks. We define the communication locality metric $CommLoc$ to describe

the variance. We adopt a related work [7] to formulate this metric. First, we normalize the amount of communication of each task pair to the largest amount of communication of all task pairs. This normalization is shown by

$$S\text{comm}_{norm}[i][j] = \frac{S\text{comm}[i][j]}{max(S\text{comm})}. \quad (8)$$

Then, $CommLoc$ is calculated by

$$CommLoc = \frac{\sum_{i=1}^{T} var(Scomm_{norm}[i][1..T])}{T}, \quad (9)$$

where max and var are the functions that calculate the maximum and variance, respectively. A locality-based task mapping method will gain higher performance improvements when considering parallel applications that have higher values of $CommLoc$.

In a parallel application that has a low or zero communication-to-memory ratio, a task mapping method can still affect the memory access behavior. In this case, distributing the non-communicating memory accesses can reduce the memory congestion because it will improve the balance of memory accesses among the NUMA nodes. However, we focus on analyzing the communicating memory accesses because of two reasons. First, as shown in related work [8], [14], the cost of remote-access communication remains the limiting factor in modern NUMA systems. Second, in many parallel applications, improving the communication locality also significantly affects the balance of memory accesses [16], [33], indicating that tasks that have higher amounts of communication also perform numerous memory accesses. We discuss the impacts of our method on both the communication locality and the memory congestion in Sections IV-A-3 and IV-B.

IV. EXPERIMENTAL EVALUATION

To evaluate the effectiveness of the proposed method, we conducted experiments on a real system and a simulation environment. In this section, we present the experimental setup and discuss the performance results.

A. PERFORMANCE EVALUATION ON A REAL SYSTEM

The experiments were conducted on a 2-node Intel-based NUMA system consisting of two Intel Xeon E5-2680v4 processors, named Purple. These NUMA nodes are connected with QuickPath Interconnect (QPI), and each node has 28 logical cores, 64 GB of main memory, and an integrated memory controller (IMC) [34]. Each NUMA node or processor has private L1 and L2 caches and an L3 cache as a last-level cache that is shared among all cores of the node. The system has 56 logical cores in total, and it runs the Linux OS kernel v4.4.

We evaluate the proposed method using three sets of parallel benchmarks: the MPI and OpenMP implementations of the NPB [32] v3.3.1 and the PARSEC benchmark suite [35] v2.1. We execute all of the NPB applications with the class C input size. For PARSEC, we use the native input size, which is the largest size available. The applications of PARSEC and the OpenMP implementation of NPB (NPB-OMP) are executed with 56 threads. For the MPI implementation (NPB-MPI), the number of processes executing the CG-MPI, FT-MPI, IS-MPI, and MG-MPI must be a power of two. Thus, 32 processes are launched to execute these four applications. The number of processes of BT-MPI and SP-MPI is required to be a square, and thus, 49 processes are launched to execute BT-MPI and SP-MPI. EP-MPI and LU-MPI are executed with 56 processes using all the cores. For the NPB-MPI applications, we use Open MPI v3.1 [26] as the MPI runtime system. By default, Open MPI uses the *vader* BTL component to optimize the data transfer between NUMA nodes.

We apply the task mapping obtained in Step 4 by assigning tasks to processor cores when the application is launched. To assign tasks to processor cores, the use of a specific tool is not mandatory. For the MPI applications, we assign tasks to processor cores by specifying the mapping between MPI ranks and processor core IDs in the rank file. For the multi-threaded applications, we assign tasks to processor cores by setting the processor affinity for each thread according to the mapping result. Some tools such as Hwloc-bind [18] and Likwid-pin [36] can be used to set the processor affinity of the threads.

To detect the communications in multi-threaded applications, we trace the timestamp of a communication event in the ns and μs time resolutions. We use only μs for the NPB-OMP due to the tracing time constraints. The number of memory accesses for NPB-OMP applications with the class C input size is much higher than that for PARSEC applications, and dynamic binary instrumentation drastically increases the duration of the tracing process. However, the results of analysis of the communication behaviors of the NPB-OMP applications show that we can still effectively analyze the communication behaviors of the NPB-OMP applications using the μs time resolution. These results are presented in Section IV-A-1.

We compare DeLoc with a dynamic mapping method and five static mapping methods. The dynamic mapping method is called AutoNUMA [3], [14], [37], which is the default thread and memory mapping algorithm used in the Linux kernel. It can be enabled and disabled through the sysctl interface by setting `kernel.numa_balancing` to 1 and 0, respectively. AutoNUMA uses information on page faults of parallel applications to dynamically detect memory accesses and performs thread and memory mapping. During the application runtime, it may migrate memory pages and threads to improve the locality and balance of memory access among the NUMA nodes, and thus, it incurs overhead from the migration.

The five static methods used for the evaluation are Packed, Scatter, Balance, Locality, and Random mapping. Both Packed and Scatter do not consider the communication behaviors of the application. Packed maps the neighboring tasks to the same NUMA node, while Scatter maps the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access

neighboring tasks to different NUMA nodes. In our previous work [15], Scatter corresponded to the Socket-span mapping policy. In the case where neighboring tasks have a larger amount of communication than that of the other tasks, Packed will increase the locality of communication, while Scatter will reduce the communication load imbalance among the NUMA nodes. We describe this case in more detail in Section IV-A-2.

In contrast to Packed and Scatter, Balance and Locality consider the spatial communication behavior of the application. Balance minimizes the communication load imbalance among the NUMA nodes. It iteratively maps the unmapped task with the largest amount of communication to the NUMA node that currently has the smallest amount of communication. Locality minimizes the amount of remote-access communication by mapping tasks that have a larger amount of communication to the same NUMA node. We use the TreeMatch algorithm [13] to calculate the mapping for Locality. For Random, we randomly generate a task mapping for each execution. We use Random mapping to evaluate the importance of task mapping. In the case of executing the benchmarks with the static mapping methods, we disable AutoNUMA to avoid the effects of the memory page and thread migrations on the results of the static methods. For data mapping, we use the *first-touch* mapping [38], which is the default mapping policy of the Linux kernel.

We are aware that data mapping can also improve the locality of memory accesses on NUMA systems. As shown in [7], [39], task mapping is a prerequisite of data mapping, and the primary benefit of data mapping is that it can prevent unnecessary task migrations between NUMA nodes to improve the locality. However, DeLoc and the other static mapping methods apply task mapping only when the target application is launched, and thus, these methods do not need to migrate tasks during the execution of the application. Furthermore, in first-touch data mapping, a memory page is allocated to the same node with the task that first uses the page, and the page is not migrated during the execution. Therefore, in the case of DeLoc and the other static methods, the task mapping also determines the data mapping. However, in contrast to the other static mapping methods, the DeLocMap algorithm computes the task mapping that can both improve the memory access locality and reduce the memory congestion.

1) Communication Behaviors of the Benchmarks

In this subsection, we analyze the communication behaviors of the benchmarks using the metrics introduced in Section III. All of the metrics, except the DRAM-to-memory ratio, are obtained from Steps 2 and 3 described in Section II. The DRAM-to-memory ratio is obtained by measuring performance counters with the Linux perf tool [40]. Figures 4, 5, and 6 show the values of the metrics for the NPB-MPI, NPB-OMP, and PARSEC applications, respectively. The vertical axis of each figure represents the values of the metric shown by the figure. The values of $CommC$, $CommR$, $DramR$ and $CommLoc$ are shown as percentages, and the values of $L\mathrm{comm}$ are in gigabytes.

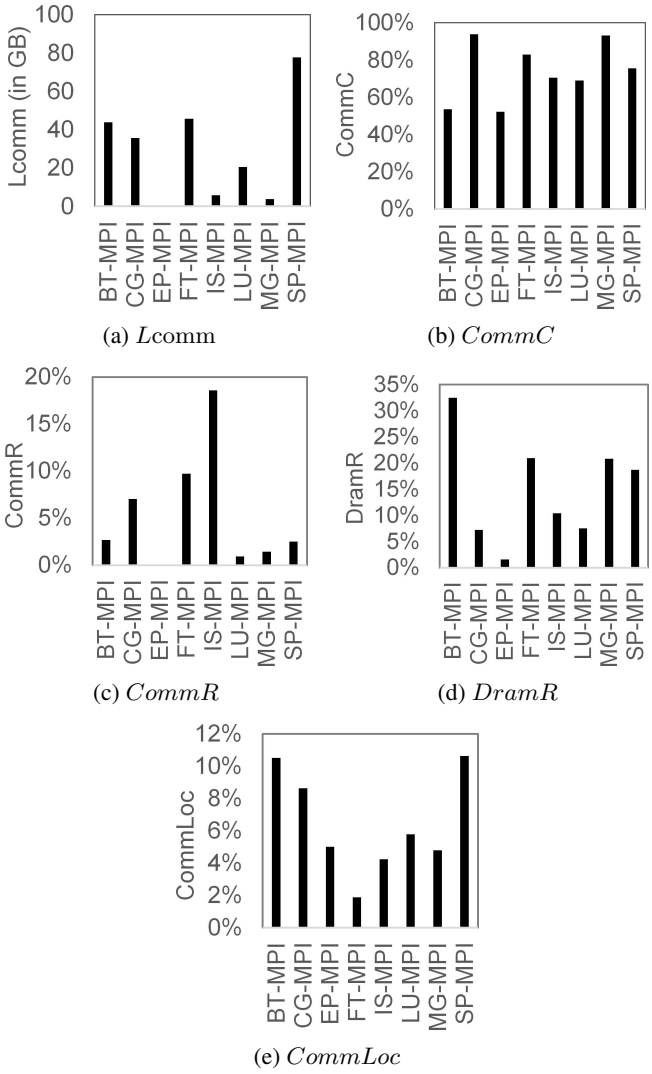


FIGURE 4: Communication behaviors of the NPB-MPI

EP-MPI has a low communication load (Figure 4(a)) and communication-to-memory ratio (Figure 4(c)), indicating that it cannot gain improvement from communication-aware task mapping. CG-MPI, FT-MPI, IS-MPI, MG-MPI and SP-MPI have a higher communication concurrency (Figure 4(b)) and DRAM-to-memory ratio (Figure 4(d)) than those of the other applications. These results indicate that they can gain a higher performance improvement from memory-congestion-aware task mapping than that from the other applications. In BT-MPI, CG-MPI, LU-MPI, and SP-MPI, the communication locality is higher than that of the other applications (Figure 4(e)). However, LU-MPI has a low communication-to-memory ratio and DRAM-to-memory ratio, indicating that it cannot gain significant performance improvement from communication-aware task mapping. On the other hand, BT-MPI, CG-MPI, and SP-MPI can gain a higher performance improvement from locality-based mapping than from the other applications. The results in Figure 4 indicate that all

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access

NPB-MPI applications, except EP-MPI and LU-MPI, are expected to benefit from the proposed method.

For NPB-OMP applications, the results of communication-to-memory (Figure 5(c)) and communication locality (Figure 5(e)) indicate that all the applications, except EP-OMP and FT-OMP, can benefit from locality-based mapping. In EP-OMP and FT-OMP, the load of communication (Figure 5(a)) and communication-to-memory ratio are low, indicating that these two applications cannot gain performance improvement from communication-aware task mapping. The results of the communication concurrency (Figure 5(b)), communication-to-memory ratio and DRAM-to-memory ratio (Figure 5(d)) show that all NPB-OMP applications, except EP-OMP and FT-OMP, have a high risk of memory congestion. In CG-OMP, although the DRAM-to-memory ratio is low, the communication-to-memory ratio is the highest among the NPB-OMP applications. This result means that CG-OMP has many more memory accesses to the cache memory than that to the DRAM and that it can benefit from the proposed method to reduce the congestion of memory access to the shared caches. The results in Figure 5 suggest that all NPB-OMP applications, except EP-OMP and FT-OMP, are expected to benefit from the proposed method.

Although most PARSEC applications have a high communication concurrency (Figure 6(b)), some applications have a low communication-to-memory ratio (Figure 6(c)), which means that not all PARSEC applications will benefit from communication-aware task mapping. In Blackscholes, Swaptions and Vips, the load of communication (Figure 6(a)) and communication-to-memory ratio are negligible, indicating that these three applications cannot gain performance improvement from communication-aware task mapping. In Freqmine, although the communication load is higher than that of other PARSEC applications, the communication-to-memory ratio is low, which means that Freqmine cannot gain significant improvement from communication-aware task mapping. Although Canneal, Dedup, Ferret and Fluidanimate have a lower communication load than that of Freqmine, these four applications have higher communication-to-memory and DRAM-to-memory ratios (Figure 6(d)). Thus, Canneal, Dedup, Ferret and Fluidanimate can still gain performance improvements from memory-congestion-aware task mapping.

In Bodytrack, the DRAM-to-memory ratio is low. However, it has a higher communication-to-memory ratio than that of most of the other applications, which means that Bodytrack has many more memory accesses to the cache memory than to the DRAM. Thus, it can benefit from the proposed method to reduce the congestion of memory access to the shared caches. In contrast to Bodytrack, Raytrace has a higher DRAM-to-memory ratio than that of most of the PARSEC applications, which means that it can benefit from the proposed method to reduce the congestion on memory controllers. On the other hand, Facesim, Streamcluster and X264 have higher loads of communication, communication concurrency, communication-to-memory ratio and DRAM-to-memory ratio than those of most of the other applications, indicating that these three applications will gain significant performance improvements from memory congestion-aware task mapping. In Facesim, the communication locality (Figure 6(e)) is the highest among the PARSEC applications, indicating that it will gain a higher performance improvement from locality-based mapping compared with the other applications. The results in Figure 6 indicate that all PARSEC applications, except Blackscholes, Freqmine, Swaptions, and Vips, are expected to benefit from the proposed method.

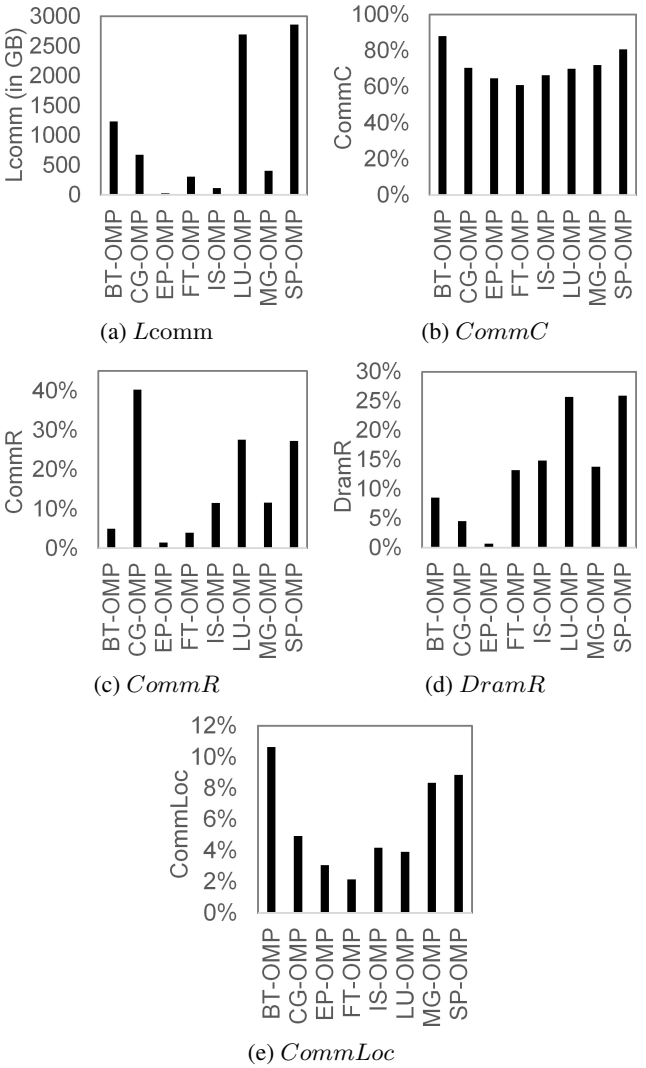
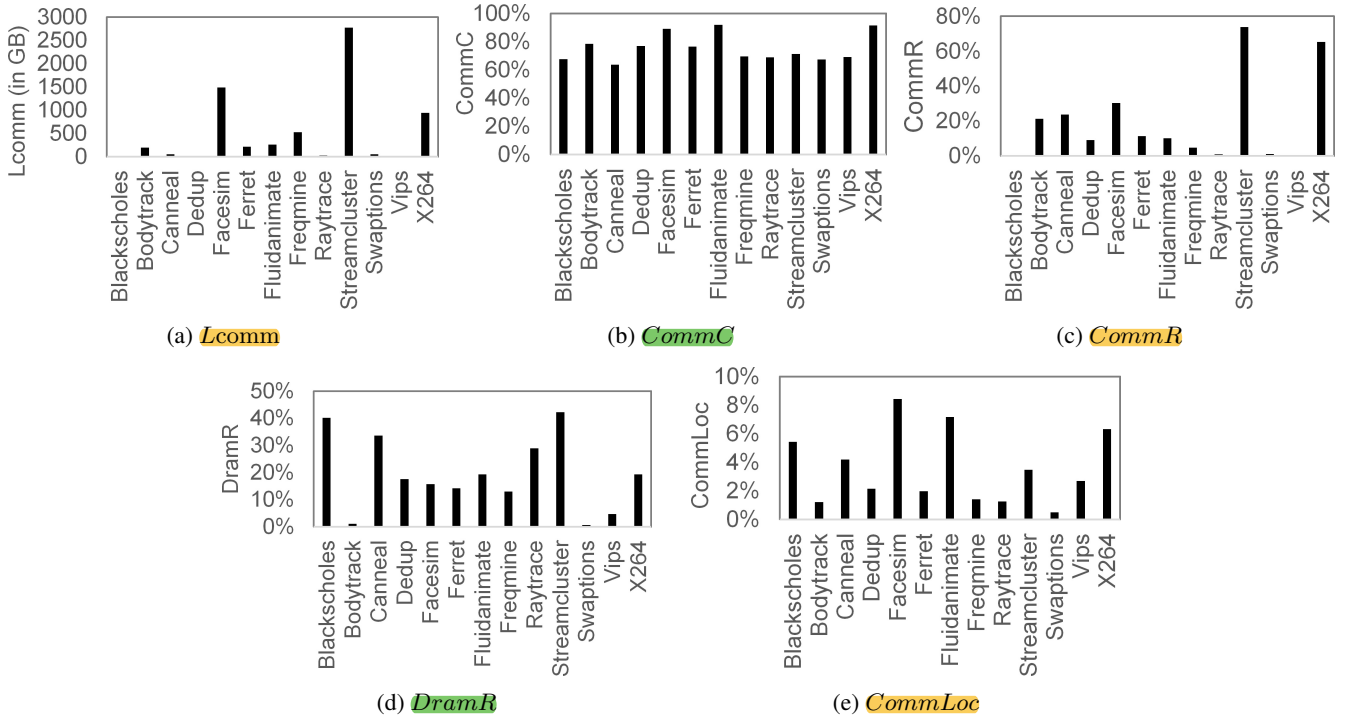


FIGURE 5: Communication behaviors of the NPB-OMP

2) Performance Results

The performance results obtained in the real system are shown in Figure 7. We measure the execution time of the applications with each mapping method. The results are the averages obtained from 10 sample executions, which are normalized to the results of Scatter mapping. We also provide the 95% confidence interval calculated with Student's t-distribution. The error line of the bar represents the confi-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access



(a) *Lcomm* (b) *CommC* (c) *CommR*

(d) *DramR* (e) *CommLoc*

FIGURE 6: Communication behaviors of the PARSEC

dence intervals of the samples. We use Scatter as the baseline because, as shown in our previous work [15] and related work [14], [41], [42], the memory access imbalance among the NUMA nodes can increase the memory congestion, and Scatter can reduce the memory access imbalance among the NUMA nodes without the need for information about the communication behavior of the application.

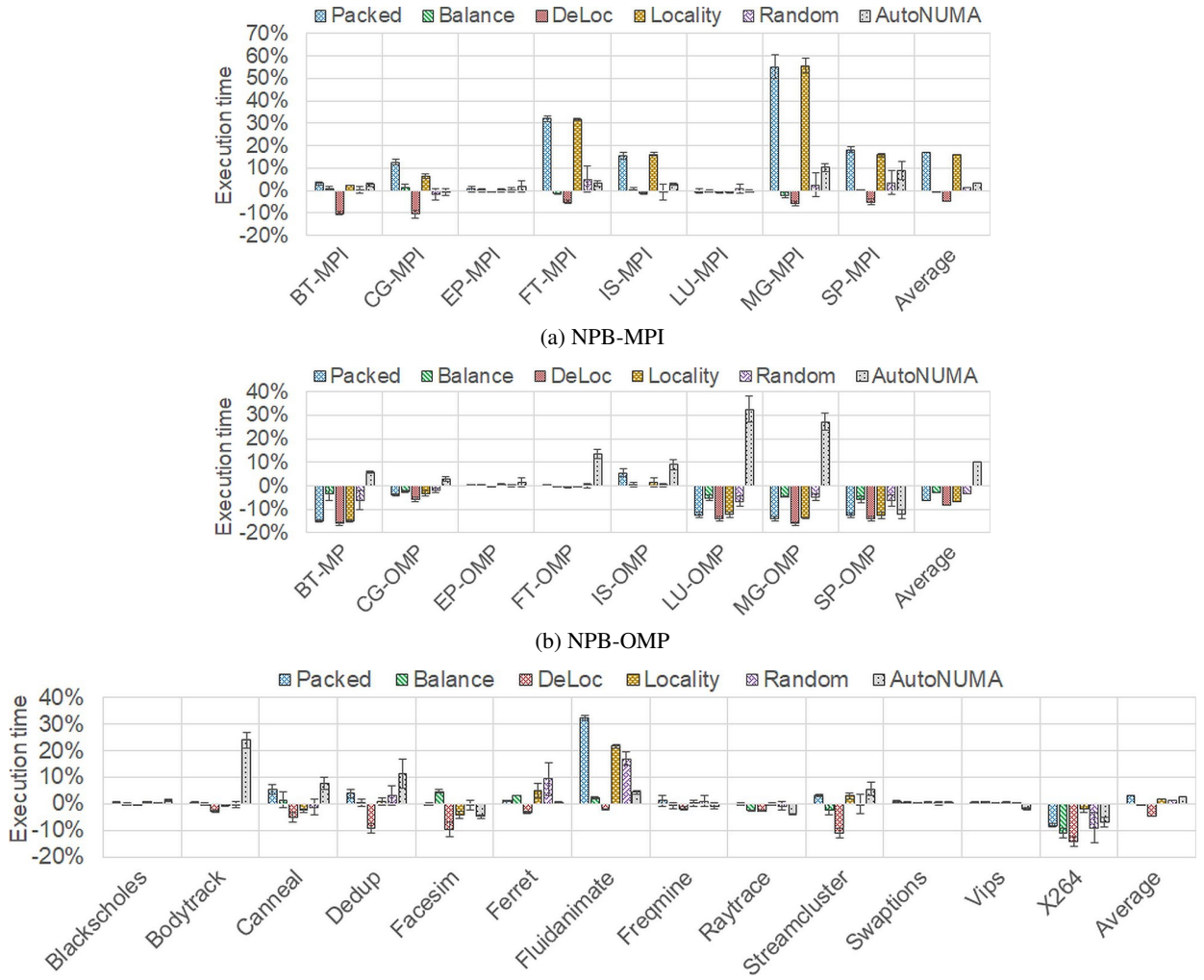
Figure 7(a) depicts the execution time of the NPB-MPI applications. The results of Random mapping show that most of the NPB-MPI applications are affected by the task mapping. On average, DeLoc shows the highest improvements among the methods, by 4.8% compared with Scatter. As predicted by our analysis of the communication behaviors of the NPB-MPI applications, DeLoc can achieve the highest improvements for all NPB-MPI applications, except EP-MPI and LU-MPI. Compared with Locality, DeLoc gains the highest performance improvements for FT-MPI and MG-MPI, by 36.8% and 61%, respectively.

In BT-MPI, CG-MPI and SP-MPI, Locality has a shorter execution time than that of Packed because these three applications have the highest communication locality among the NPB-MPI applications (Figure 4(e)). However, in most of the NPB-MPI applications, DeLoc, Balance, AutoNUMA and Scatter outperform Locality, indicating that locality-based mapping cannot achieve the best performance among the applications. These results suggest that in the Purple system, the impact of memory congestion on the performance of the NPB-MPI applications is higher than that of the locality.

We note that in the case where the number of tasks is less than the number of processor cores available, Locality can map more tasks to one NUMA node to reduce the amount of remote-access communication. Since the number of concurrent communications on one NUMA node increases, the memory congestion increases on that particular node. However, the results of Balance and DeLoc also show that minimizing the communication load imbalance itself is not sufficient to achieve the best performance and that considering both the locality and the memory congestion is still crucial to achieving the best performance.

For NPB-OMP applications, task mapping also affects most of the applications. However, as shown in Figure 7(b), Scatter has the lowest performance among the methods in most of the NPB-OMP applications. These results are in contrast to those of NPB-MPI applications. Moreover, on average, Locality can achieve higher performance improvements than can Packed, Balance and Scatter. These results indicate that most of the NPB-OMP applications gain more benefit from locality-based mapping. On the other hand, DeLoc achieves the highest performance improvements among the methods in most of the NPB-OMP applications, by up to 16.1% in the cases of BT-OMP and MG-OMP (8.3% on average). As predicted by our analysis of the communication behaviors of NPB-OMP applications, DeLoc can achieve the highest performance improvements in BT-OMP, LU-OMP, MG-OMP and SP-OMP. As shown in the results of the communication concurrency, communication-to-memory ratio and DRAM-to-memory ratio, these four applications have the highest risk of memory congestion among the NPB-OMP applications. This fact indicates that considering only the locality is not sufficient to achieve the best performance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access



(a) NPB-MPI

(b) NPB-OMP

(c) PARSEC

FIGURE 7: Performance results (normalized to Scatter)

for these applications.

As discussed in Section IV-A-1, DeLoc can reduce the execution times of most of the PARSEC applications. Moreover, on average, DeLoc can achieve the highest improvements among the methods. Among the PARSEC applications, DeLoc can achieve the highest performance improvements in Facesim, Streamcluster, and X264, by 9.7%, 11% and 14.1%, respectively. Compared with Balance, DeLoc shows the highest improvement in Facesim by 14.3%. These results suggest the importance of increasing locality to improve the performance of Facesim. In Fluidanimate, DeLoc can achieve a 23.7% shorter execution time than that of Locality. Moreover, compared with Packed, Balance and DeLoc, Locality shows the lowest performance in Fluidanimate, Streamcluster and X264, indicating that maximizing the locality degrades the performance of these three applications.

In the case of a parallel application that has a higher communication locality and a lower DRAM-to-memory ratio, such as Facesim, reducing the amount of remote-access communication is more effective in improving the performance. In this case, we may further increase the performance by tuning the DeLocMap algorithm to maximize the locality of the communication among task pairs that have a higher amount of communication. However, the impacts of the locality and memory congestion on the execution time may be different for different NUMA systems, even for the same application. Thus, to accurately estimate the impacts of the locality and memory congestion on the execution time of an application, we also need to take into account the latency and bandwidth characteristics of the NUMA system. For this reason, we consider the tuning of our method as our future work.

In most of the NPB-OMP applications and some PARSEC applications, AutoNUMA has the longest execution time among the methods. In LU-OMP, AutoNUMA experiences the highest performance degradation, by 46.5% and 32.4% compared with DeLoc and Scatter, respectively. Furthermore, in FT-OMP and Bodytrack, although most of the static methods show a similar execution time, AutoNUMA has a much

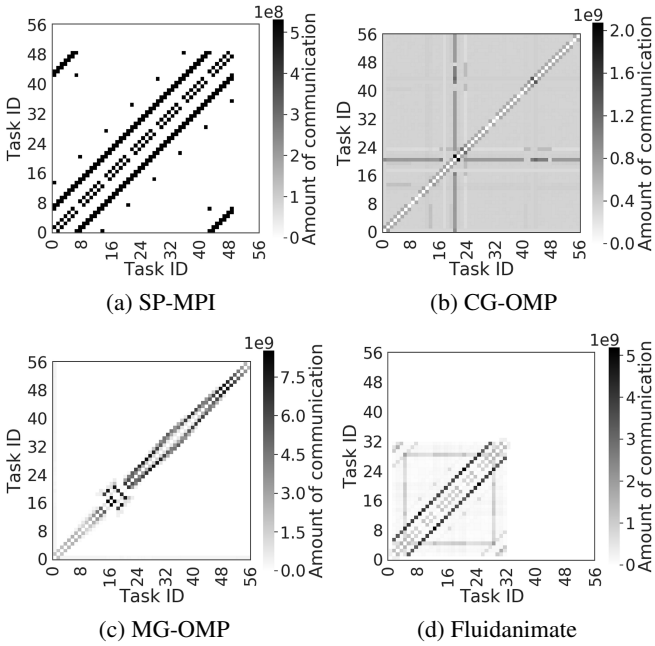
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access

longer execution time than that of the other methods. In contrast to the static mapping methods, AutoNUMA suffers from overhead from migrating memory pages and threads during the application runtime. These results show that the migration overhead significantly degrades the performance of the applications. For NPB-MPI applications, AutoNUMA shows higher performance improvements than those of NPB-OMP applications, as NPB-OMP applications have much more memory accesses compared with NPB-MPI applications. The migration overhead has higher impacts on the performance of NPB-OMP applications. We discuss the impacts of the migration overhead in more detail in Section IV-A-3.

Figure 7 shows that DeLoc can consistently achieve the highest performance among the methods. By taking into account both the spatial and temporal communication behaviors of the applications, DeLoc can effectively reduce the amount of remote-access communication and memory congestion. The experimental results also show the effectiveness of our metrics in evaluating whether the applications can benefit from task mapping.

We observe that Scatter can achieve shorter execution times than can Locality and Packed for most NPB-MPI applications and that Packed can achieve shorter execution times than can Balance for most NPB-OMP applications. Although Packed and Scatter do not consider the communication behaviors of applications, these two mapping methods can effectively improve the performance of applications that have a communication behavior in which neighboring tasks have a larger amount of communication than that of the other tasks. However, as shown in the results of the NPB-MPI, NPB-OMP and PARSEC applications, both Packed and Scatter cannot consistently improve performance. Furthermore, as shown in our previous work [15], when the number of NUMA nodes in the system becomes larger, Scatter may suffer from high latencies of the remote-access communication, and it cannot effectively reduce the memory congestion. The experimental results show that to effectively reduce the amount of remote-access communication and memory congestion, it is necessary to consider the communication behaviors of the application.

Figure 8 shows examples of the communication behavior that can benefit from Packed and Scatter mapping. Figures 8(a), 8(b), 8(c) and 8(d) show the spatial communication behaviors of SP-MPI, CG-OMP, MG-OMP and Fluidanimate, respectively. In the figures, the x-axis and y-axis show the task ID, and each cell represents the amount of communication ($S\mathrm{comm}$) between a task pair of the corresponding axes. The darker cells indicate a larger amount of communication. As shown in the figures, SP-MPI, CG-OMP, MG-OMP and Fluidanimate exhibit a similar communication behavior, with a large amount of communication between neighboring tasks, such as task pair $(0, 1)$. These results show that in these four applications, Packed will reduce the amount of remote-access communication, and Scatter will reduce the communication load imbalance among the NUMA nodes. Moreover, SP-MPI and Fluidanimate show a similar amount of communication



(a) SP-MPI (b) CG-OMP

(c) MG-OMP (d) Fluidanimate

FIGURE 8: The communication behaviors of SP-MPI, CG-OMP, MG-OMP and Fluidanimate

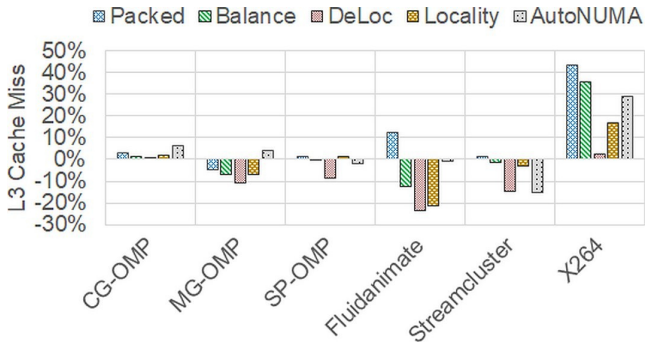
between tasks that are further apart, such as task pairs $(1, 7)$ and $(2, 8)$ in SP-MPI and task pairs $(1, 5)$ and $(2, 6)$ in Fluidanimate. These results show that in SP-MPI and Fluidanimate, Scatter can also reduce the amount of remote-access communication.

3) Performance Results Analysis

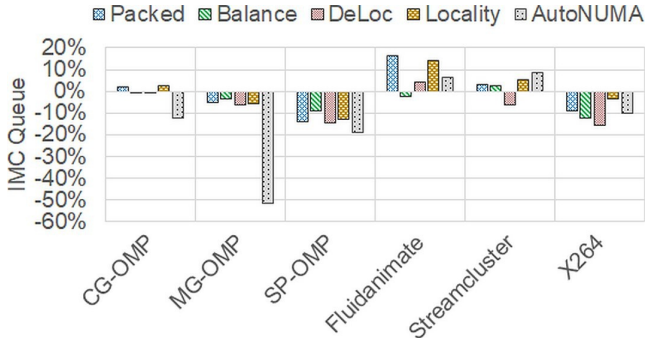
To investigate the sources of performance improvements, we analyze the performance characteristics of six applications selected from NPB-OMP and PARSEC. These applications are CG-OMP, MG-OMP, and SP-OMP of the NPB and Fluidanimate, Streamcluster and X264 of the PARSEC. We use the last-level cache misses, IMC queue, and QPI volume metrics for the analysis. These metrics are obtained by measuring the Intel performance counters [34] with the Linux perf tool. The L3 cache misses represent the number of last-level cache misses across all NUMA nodes. The IMC queue is the total wait time of memory accesses in the queue of memory controllers. A higher value of this metric indicates a longer queuing delay caused by the congestion on memory controllers. We also use last-level cache misses to evaluate the impact of memory congestion because the congestion of memory access to last-level caches will increase the number of cache misses [43]. The QPI volume is the volume of data sent through interconnects and represents the amount of remote-access communication. A higher value of this counter indicates longer latencies from remote-access communication. In this evaluation, we do not evaluate Random mapping because the performance monitoring results of Random mapping can significantly change for different executions.

Figure 9 shows the performance monitoring results of

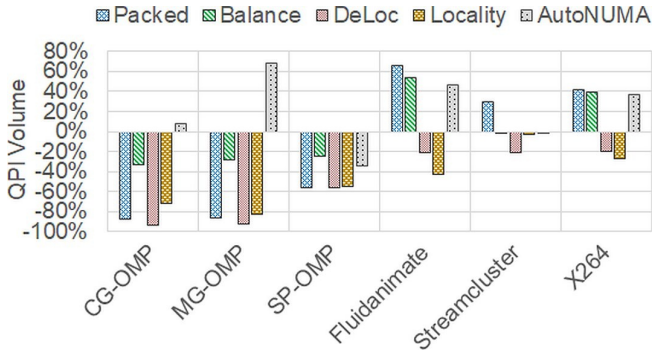
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access



(a) L3 cache misses



(b) IMC queue



(c) QPI volume

FIGURE 9: The monitoring results of the NPB and PARSEC applications

the six applications, which are normalized to the results of Scatter mapping. In MG-OMP and SP-OMP, DeLoc can achieve the highest improvement by reducing the number of last-level cache misses, IMC queue and amount of remote-access communication. The results of MG-OMP and SP-OMP show that DeLoc increases the locality of communication. Furthermore, by distributing the communication load over the NUMA nodes, DeLoc can reduce not only the congestion on memory controllers but also the congestion of memory access to the last-level caches. In CG, Packed shows a significant reduction in QPI volume because, as shown in Figure 8(b), CG has a communication behavior that can benefit from Packed mapping. The results of the IMC queue show a small difference among the methods, which means that in CG, the locality has a higher impact than the memory congestion; thus, Packed and Locality have higher performance improvements than does Balance. On the other hand, DeLoc shows the lowest QPI volume and IMC queue; thus, it can achieve the highest performance improvement among the methods.

In Fluidanimate, DeLoc and Locality have fewer last-level cache misses than do the other methods because both methods improve the locality of communication. We note that Scatter has a lower IMC queue than that of Packed and DeLoc and a lower QPI volume than that of Packed and Balance. This is because, as shown in Figure 8(d), this application has a communication behavior that can benefit from Scatter mapping. In the case of Locality, although the number of cache misses is lower than that of Balance and Scatter, the IMC queue is much higher than that of Balance and Scatter. Thus, Locality has a lower performance than that of Balance, DeLoc and Scatter. On the other hand, DeLoc can achieve the highest performance improvements by simultaneously reducing the memory congestion and the amount of remote-access communication.

In Streamcluster, DeLoc gains the highest performance improvements by simultaneously reducing the number of cache misses, IMC queue and QPI volume. Locality has fewer cache misses than do Packed, Balance and Scatter. However, Locality has the highest IMC queue. On the other hand, Balance and Scatter have a lower IMC queue and execution time than those of Locality. As predicted by our analysis of the communication behaviors of the PARSEC applications, in Streamcluster, the impact of memory congestion is higher than that of the locality. The performance monitoring results show that maximizing the locality can degrade the performance of this application because doing so will significantly increase the memory congestion.

In X264, Locality has the lowest QPI volume among the methods. However, Balance and DeLoc have the lowest IMC queue and shortest execution time among the methods. As shown in Figure 6(d), X264 has a higher DRAM-to-memory ratio than that of most of the PARSEC applications. These results show that both methods can achieve a higher performance improvement than can the other methods by significantly reducing the congestion on memory controllers. Note that Locality can achieve a lower IMC queue than that of Scatter, indicating that in X264, improving the locality of communication can also reduce the communication load imbalance among the NUMA nodes. On the other hand, DeLoc can achieve the highest performance improvements from reductions in the IMC queue and QPI volume, which means that it can effectively reduce the congestion on memory controllers and the amount of remote-access communication.

In CG-OMP, MG-OMP, and SP-OMP, AutoNUMA has the lowest IMC queue, indicating that this method effectively reduces the memory congestion in these three applications. However, in all six applications, AutoNUMA has a higher QPI volume than that of DeLoc and Locality. The highest QPI volume is shown in MG-OMP, by 67.9% compared

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access

TABLE 1: Simulation configuration

Parameter	Value
NUMA nodes (processors)	4x 16-core processors; L1I/L1D cache per core; L2 cache per core; L3 cache shared between 16 cores; 4x memory controllers; 5x bidirectional interconnects
Processor cores	2.4 GHz; Nehalem performance model
L1I/L1D caches	256 KB; 8-way; 64-byte line size; LRU policy
L2 caches	2 MB; 8-way; LRU policy
L3 caches	20 MB; 16-way; LRU policy
Memory controllers	60 ns latency; 36 GB/s bandwidth; 14-way interleave; DRAM directory model
Interconnects	25.6 GB/s bandwidth; network bus model

with Scatter. By migrating memory pages and threads to a different NUMA node, AutoNUMA potentially increases the data traffic on interconnects because the threads may need to access data that reside in a remote NUMA node. These results suggest that the migration overhead has a significant impact on the volume of data traffic on the interconnects. On the other hand, DeLoc and Locality do not suffer from the migration overhead, and thus, these two methods have a lower QPI volume than that of AutoNUMA.

The performance monitoring results show that DeLoc gains performance improvements from the reductions in the last-level cache misses, IMC queue and QPI volume. Higher improvements are achieved by the applications that have a higher communication concurrency, communication-to-memory ratio and DRAM-to-memory ratio.

B. PERFORMANCE EVALUATION WITH A SIMULATOR

To evaluate the effectiveness of the proposed method on a larger-scale system, we conduct experiments in a multicore simulator, called Sniper [44]. A 4-node NUMA system is used for the simulation configuration. We set the specifications of each processor, memory controllers, and QPI according to the hardware specifications of the Purple system.

Table 1 shows the configuration parameters for the simulation. As the benchmark applications, we use six applications that are used in the performance analysis of the real system evaluation (Figure 9). These applications are executed with 64 threads. We use the simlarge input size for the PARSEC applications. For the NPB-OMP applications, we use the class B input size for CG-OMP and MG-OMP and the class A input size only for SP-OMP. We do not use the larger input sizes due to the simulation time constraints. The simulation time drastically increases with the input size. We observed that for one CG-OMP execution, the simulation time with the class B input size is slower than that with the class A input size by two orders of magnitude.

In this evaluation, DeLoc is compared with Locality, Balance, and Scatter. We choose these three methods because Locality and Balance consider the spatial communication behavior of the application, and in the real system evaluation, Scatter shows a higher performance improvement than achieved by Packed, Balance, Locality and AutoNUMA in the case of Fluidanimate. Moreover, in Fluidanimate, the performance difference between Scatter and Packed is the largest among the NPB-OMP and PARSEC applications.

Figure 10 shows the results of executing the six applications in the simulator, which are also normalized to the results of Scatter mapping. In this evaluation, we use four performance metrics used in the performance analysis of the real system evaluation. These metrics are obtained from the simulation output of Sniper. The IMC queue and QPI volume metrics are obtained by measuring the DRAM queuing delay and network packet counters in the simulator. As shown in Figure 10, on average, DeLoc can achieve the highest performance improvement among the methods, by up to 19.4% in the case of SP-OMP.

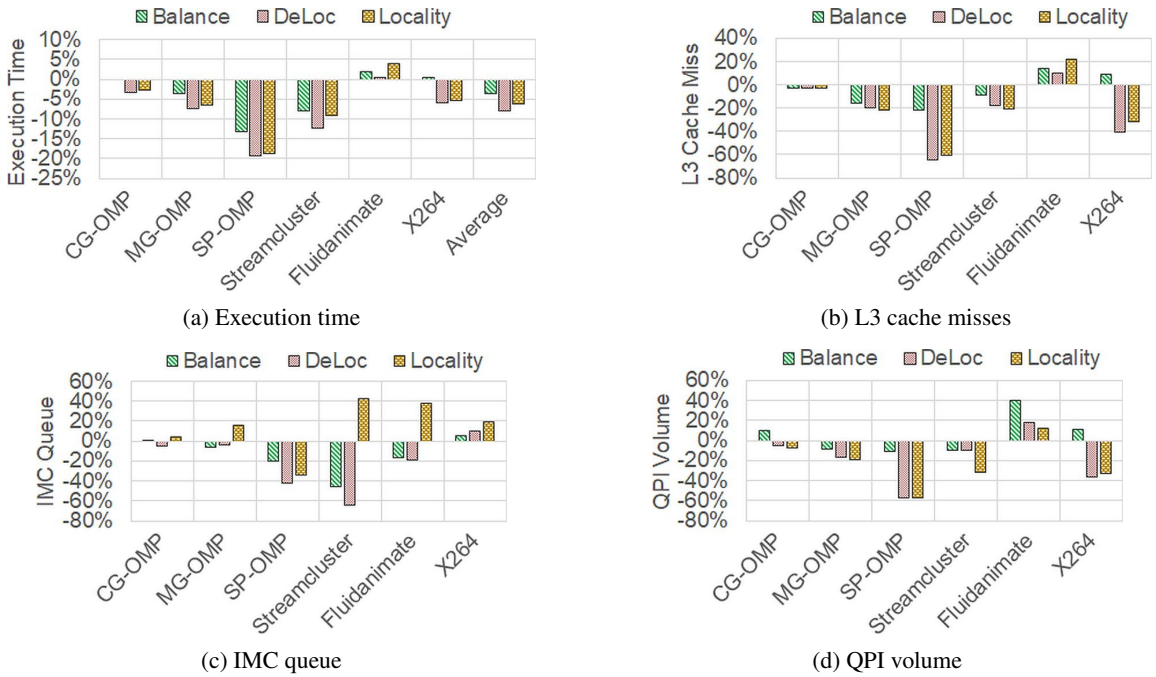
In all the tested applications, except Fluidanimate, DeLoc and Locality achieve higher performance improvements than do Scatter and Balance. The performance improvements are mainly obtained from the reductions in last-level cache misses and QPI volume, with the highest reductions exhibited in SP-OMP and X264. In X264, the reductions in QPI volume of DeLoc and Locality are 36.4% and 32.9%, respectively. In CG-OMP and Streamcluster, Balance has a higher QPI volume than that of Scatter and Locality, respectively, which is contrary to the results of the real system evaluation. These results suggest that on the simulated system, the impact of communication locality on the execution time is higher than that of the real system. Thus, reducing the amount of remote-access communication becomes more effective in improving the performance of most of the applications.

In SP-OMP, DeLoc has not only the lowest last-level cache misses and QPI volume but also the lowest IMC queue; thus, it achieves the highest performance improvement among the methods. Moreover, the reductions in the IMC queue and execution time are higher than those of the real system evaluation. This fact shows that on the simulated system, the impact of memory congestion on the execution time of SP-OMP is also higher than that in the real system, as the number of cores of each NUMA node in the simulated system is higher than that in the Purple system.

In Fluidanimate, DeLoc and Scatter achieve the highest improvements among the methods. Scatter can achieve a comparable performance with DeLoc because this application has a communication behavior that can benefit from Scatter mapping, which is also exhibited in the real system evaluation. These results show that the communication behavior of Fluidanimate is not changed, even if the input size is changed. We observe that Locality has the highest IMC queue and number of last-level cache misses among the methods. By minimizing the amount of remote-access communication, it increases the congestion of memory access to the last-level caches and memory controllers. On the other hand, DeLoc can achieve a shorter execution time than that of Balance and Locality from the reductions in both the memory congestion and the amount of remote-access communication.

We note that reducing memory congestion may increase the amount of remote-access communication. As shown in Streamcluster and Fluidanimate, DeLoc has a higher QPI volume than that of Locality. This is because, to reduce memory congestion, DeLoc distributes the concurrent com-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2963726, IEEE Access



(a) Execution time

(b) L3 cache misses

(c) IMC queue

(d) QPI volume

FIGURE 10: Performance results in the simulator (normalized to Scatter)

munications over the NUMA nodes. However, as discussed in the real system evaluation, the memory congestion has a high impact on the execution time of Streamcluster and Fluidanimate. Thus, in these applications, DeLoc can still achieve shorter execution times than those of Locality.

The simulation results show that in most of the tested applications, the impact of communication locality on the execution time increases with the number of nodes. The applications that have a higher communication-to-memory ratio and communication locality, such as SP-OMP and X264, will gain a higher performance improvement from locality-based task mapping. DeLoc can achieve the highest performance in most of the applications by simultaneously reducing the amount of remote-access communication and the memory congestion.

V. RELATED WORK

In this section, we review the related work. First, the methods for MPI process mapping and thread mapping are reviewed. Then, we compare our proposed method with two thread and memory placement methods considering memory congestion on modern NUMA systems.

Various MPI process mapping methods have been proposed in related studies. Related work in this area mostly focuses on improving the locality of communication. The MPIPP framework uses the execution profile to place MPI processes on different nodes of a cluster [9]. Hendrickson and Leland proposed graph-based partitioning algorithms to optimize the process mapping [10]. Zhang et al. [11] and Ma et al. [12] proposed process placement strategies for MPI collective operations. A more recent method was proposed by Jeannot et al. with a tree-based algorithm called TreeMatch [13], where they took into account the application communication pattern and the hardware topology of the NUMA clusters. TreeMatch achieves a better performance than that of the previous graph-based partitioning algorithms by taking into account the application communication pattern and the qualitative information of the node topology. A key difference between the above related work and our proposed method is that the related work mainly focused on the locality. In contrast, our method focuses on improving the locality and reducing the memory congestion on modern NUMA systems.

A thread mapping method was proposed by Diener et al. [22]. The method analyzes the spatial communication behavior of multi-threaded applications to improve the locality and balance of the communication load. An online-based communication detection method, called CDSM, was proposed by Diener et al. [8]. During the application runtime, it periodically monitors and analyzes the spatial communication behavior of the application and performs thread mapping. In contrast to these related work, DeLoc considers not only the spatial communication behavior but also the temporal communication behavior of the application. As shown in our evaluation, by considering both the spatial and temporal communication behaviors, DeLoc is more effective in reducing the memory congestion. It can achieve the lowest queuing delay caused by the memory congestion in most of the tested applications.

Dashti et al. [14] proposed a memory placement method, called Carrefour, to reconcile the data access locality and memory congestion on modern NUMA systems. Carrefour

works as a Linux kernel policy to dynamically place memory pages on NUMA nodes to avoid congestion. It struggles to reduce the runtime overheads from the memory access monitoring and the memory migration. Lepers et al. [45] proposed a thread and memory placement method, called AsymSched, that takes into account the bandwidth asymmetry of asymmetric NUMA systems to minimize congestion on interconnect links and memory controllers on modern NUMA systems. It relies on continuous monitoring of the communication volume, thread migration, and memory migration. As a result, AsymSched also incurs runtime overheads from the monitoring and migrations.

CDSM, Carrefour and AsymSched introduce monitoring and migration overheads to the execution of the parallel application. In contrast to these methods, DeLoc assigns tasks to processor cores when an application is launched. The overhead of DeLoc is incurred by the profiling step to analyze the communication behaviors of the application. However, a rerun of the profiling is required only when the communication behaviors of the application have changed.

We cannot compare DeLoc with CDSM, Carrefour and AsymSched because CDSM depends on a previous version of the Linux kernel and because Carrefour and Asymsched require a profiling mechanism that is available only in AMD processors. However, as shown in our evaluation and our previous work [33], the migration overhead can have a significant impact on performance, and DeLoc does not suffer from this overhead. Moreover, unlike these methods, DeLoc works on the application level and does not rely on a specific operating system or hardware. Compared with AsymSched, DeLoc focuses on reducing not only memory congestion but also the amount of remote-access communication. As shown in our evaluation, Facesim and most of the NPB-OMP applications can gain significant improvements from reducing the amount of remote-access communication.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a task mapping method, DeLoc, to address both the locality and memory congestion problems. The proposed method analyzes the spatial and temporal communication behaviors of parallel applications to identify groups of tasks that potentially cause memory congestion. We also introduced metrics to describe the communication behaviors and determine if a parallel application can benefit from locality and memory-congestion-aware task mapping.

To evaluate the proposed method, we compared DeLoc with a dynamic mapping method, two greedy-based methods, a random method, a balance-based method and a locality-based method. The evaluation was conducted on a real NUMA system using MPI and OpenMP implementations of the NPBs and the PARSEC benchmark suite. In addition, experiments with a larger number of NUMA nodes have been conducted on a multicore simulator. The experiments show that our proposed method consistently outperforms the other methods. For parallel applications that can benefit from communication-aware task mapping, DeLoc can achieve the highest performance by improving the locality and reducing the memory congestion.

DeLoc can achieve performance improvements of up to 46.5%, 61%, and 14.3% compared with the dynamic mapping, locality-based mapping, and balance-based mapping methods, respectively. It can achieve higher performance improvements in applications that have a higher communication concurrency, communication-to-memory ratio and DRAM-to-memory ratio. The performance improvements are obtained from the reductions in last-level cache misses, queuing delay in memory controllers, and data traffics in interconnects.

Our future work will focus on improving DeLoc to make it applicable to a large-scale cluster of modern NUMA systems, which needs hybrid MPI/OpenMP parallel programming. Another interesting issue is the effect of the proposed method on the power and energy consumption of NUMA systems, which will be discussed in our future work.

ACKNOWLEDGMENTS

The authors would like to thank Kazuhiko Komatsu of Tohoku University for valuable discussions on this work.

REFERENCES

[1] D. Molka, D. Hackenberg, and R. Schöne, "Main Memory and Cache Performance of Intel Sandy Bridge and AMD Bulldozer," in *Proceedings of the Workshop on Memory Systems Performance and Correctness*, ser. MSPC '14. New York, NY, USA: ACM, 2014, pp. 4:1–4:10.

[2] F. Gaud, B. Lepers, J. Funston, M. Dashti, A. Fedorova, V. Quéma, R. Lachaize, and M. Roth, "Challenges of Memory Management on Modern NUMA Systems," *Queue*, vol. 13, no. 8, p. 70, 2015.

[3] M. Diener, E. H. Cruz, M. A. Alves, P. O. Navaux, and I. Koren, "Affinity-based thread and data mapping in shared memory systems," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 64, 2017.

[4] D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek, "Intel QuickPath Interconnect Architectural Features Supporting Scalable System Architectures," in *2010 18th IEEE Symposium on High Performance Interconnects*, Aug 2010, pp. 1–6.

[5] J. E. Boillat and P. G. Kropf, "A fast distributed mapping algorithm," in *CONPAR 90 — VAPP IV*, H. Burkhart, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 405–416.

[6] J. M. Orduña, F. Silla, and J. Duato, "On the development of a communication-aware task mapping technique," *J. Syst. Archit.*, vol. 50, no. 4, pp. 207–220, Mar. 2004.

[7] M. Diener, E. H. Cruz, L. L. Pilla, F. Dupros, and P. O. Navaux, "Characterizing communication and page usage of parallel applications for thread and data mapping," *Performance Evaluation*, vol. 88, pp. 18–36, 2015.

[8] M. Diener, E. H. Cruz, P. O. Navaux, A. Busse, and H.-U. Heiíß, "Communication-aware process and thread mapping using online communication detection," *Parallel Comput.*, vol. 43, no. C, pp. 43–63, Mar. 2015.

[9] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn, "MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters," in *Proceedings of the 20th annual international conference on Supercomputing*. ACM, 2006, pp. 353–360.

[10] B. Hendrickson and R. Leland, "The chaco user's guide: Version 2.0," Technical Report SAND95-2344, Sandia National Laboratories, Tech. Rep., 1995.

[11] J. Zhang, J. Zhai, W. Chen, and W. Zheng, "Process mapping for MPI collective communications," in *European Conference on Parallel Processing*. Springer, 2009, pp. 81–92.

[12] T. Ma, T. Herault, G. Bosilca, and J. J. Dongarra, "Process Distance-Aware Adaptive MPI Collective Communications," in *2011 IEEE International Conference on Cluster Computing*, Sept 2011, pp. 196–204.

[13] E. Jeannot, G. Mercier, and F. Tessier, "Process placement in multicore clusters:algorithmic issues and practical techniques," *IEEE Transactions*

on Parallel and Distributed Systems, vol. 25, no. 4, pp. 993–1002, April 2014.

[14] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth, "Traffic management: a holistic approach to memory placement on NUMA systems," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 381–394.

[15] M. Agung, M. A. Amrizal, K. Komatsu, R. Egawa, and H. Takizawa, "A memory congestion-aware mpi process placement for modern numa systems," in *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*, Dec 2017, pp. 152–161.

[16] M. Diener, E. H. M. Cruz, M. A. Z. Alves, M. S. Alhakeem, P. O. A. Navaux, and H.-U. Heiß, "Locality and balance for communication-aware thread mapping in multicore systems," in *Euro-Par 2015: Parallel Processing*, J. L. Träff, S. Hunold, and F. Versaci, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 196–208.

[17] A. Bhatelé, L. V. Kalé, and S. Kumar, "Dynamic topology aware load balancing algorithms for molecular dynamics applications," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 110–116.

[18] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Feb 2010, pp. 180–186.

[19] A. Kleen, "A NUMA API for Linux," *Novel Inc*, 2005.

[20] H. Kasim, V. March, R. Zhang, and S. See, "Survey on parallel programming model," in *IFIP International Conference on Network and Parallel Computing*. Springer, 2008, pp. 266–275.

[21] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," http://www.mpi-forum.org, Sept. 2012.

[22] J. Diaz, C. Munoz-Caro, and A. Nino, "A survey of parallel programming models and tools in the multi and many-core era," *IEEE Transactions on parallel and distributed systems*, vol. 23, no. 8, pp. 1369–1386, 2012.

[23] R. L. Graham and G. Shipman, "MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, A. Lastovetsky, T. Kechadi, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 130–140.

[24] H. Zhu, D. Goodell, W. Gropp, and R. Thakur, "Hierarchical Collectives in MPICH2," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, M. Ropo, J. Westerholm, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–326.

[25] G. Bosilca, C. Foyer, E. Jeannot, G. Mercier, and G. Papauré, "Online Dynamic Monitoring of MPI Communications," in *European Conference on Parallel Processing*. Springer, 2017, pp. 49–62.

[26] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2004, pp. 97–104.

[27] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of Splash-2 and Parsec," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2009, pp. 86–97.

[28] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Acm sigplan notices*, vol. 40, no. 6. ACM, 2005, pp. 190–200.

[29] M. Ackerman, S. Ben-David, S. Brânzei, and D. Loker, "Weighted clustering," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, ser. AAAI'12. AAAI Press, 2012, pp. 858–863.

[30] G. Schwarz *et al.*, "Estimating the dimension of a model," *The annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[31] D. Pelleg, A. W. Moore *et al.*, "X-means: Extending k-means with efficient estimation of the number of clusters." in *Icml*, vol. 1, 2000, pp. 727–734.

[32] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The NAS parallel benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[33] M. Agung, M. A. Amrizal, R. Egawa, and H. Takizawa, "An automatic MPI process mapping method considering locality and memory congestion on NUMA systems," in *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, Oct 2019, pp. 17–24.

[34] Intel. (2016) Intel Xeon Processor E5 and E7 v4 Product Families Uncore Performance Monitoring Reference Manual. https://www.intel.com/content/www/us/en/products/docs/processors/xeon/xeon-e5-e7-v4-uncore-performance-monitoring.html.

[35] C. Bienia and K. Li, "PARSEC 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[36] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA, 2010.

[37] J. Corbet. (2012) AutoNUMA: the other approach to NUMA scheduling. Retrieved Oct 1, 2019 from https://lwn.net/Articles/488709.

[38] C. Lameter *et al.*, "NUMA (Non-Uniform Memory Access): An overview." *Acm queue*, vol. 11, no. 7, p. 40, 2013.

[39] C. Terboven, D. an Mey, D. Schmidl, H. Jin, and T. Reichstein, "Data and thread affinity in OpenMP programs," in *Proceedings of the 2008 Workshop on Memory Access on Future Processors: A Solved Problem?*, ser. MAW '08. New York, NY, USA: ACM, 2008, pp. 377–384.

[40] A. C. De Melo, "The new Linux perf tools," in *Slides from Linux Kongress*, vol. 18, 2010.

[41] M. Awasthi, D. W. Nellans, K. Sudan, R. Balasubramonian, and A. Davis, "Handling the problems and opportunities posed by multiple on-chip memory controllers," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010, pp. 319–330.

[42] R. Lachaize, B. Lepers, and V. Quéma, "Memprof: A memory profiler for NUMA multicore systems," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'12, Berkeley, CA, USA, 2012, pp. 5–5.

[43] A. Fedorova, S. Blagodurov, and S. Zhuravlev, "Managing contention for shared resources on multicore processors," *Queue*, vol. 8, no. 1, pp. 20:20–20:35, Jan. 2010.

[44] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 3, p. 28, 2014.

[45] B. Lepers, V. Quéma, and A. Fedorova, "Thread and memory placement on NUMA systems: Asymmetry matters," in *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '15, Berkeley, CA, USA, 2015, pp. 277–289.

MULYA AGUNG received a Master's degree in Informatics from Bandung Institute of Technology in 2015. He has been working toward a Ph.D. degree at Tohoku University. His work addresses data science and high-performance computing.

MUHAMMAD ALFIAN AMRIZAL is currently an assistant professor at the Research Institute of Electrical Communication (RIEC), Tohoku University. His research interests are in the area of high-performance computing (HPC), including the dependability of HPC systems, novel fault tolerance techniques, performance modeling, and optimization. He received a B.E. degree in Mechanical Engineering and M.S. and Ph.D. degrees in Information Sciences from Tohoku University in 2012, 2014 and 2017, respectively.

RYUSUKE EGAWA received a B.E. degree and Master's degree in Information Sciences from Hirosaki University in 1999 and 2001, respectively, and a Ph.D. degree in Information Sciences from Tohoku University in 2004. He is currently an associate professor at the Cyberscience Center, Tohoku University. His research interests include computer architecture, VLSI design, and high-performance computing. He is a member of IEEE CS, IEICE, and IPSJ.

HIROYUKI TAKIZAWA is currently a professor at the Cyberscience Center, Tohoku University. His research interests include high-performance computing systems and their applications. He received a B.E. degree in Mechanical Engineering and M.S. and Ph.D. degrees in Information Sciences from Tohoku University in 1995, 1997 and 1999, respectively. He is a member of IEEE CS, ACM SIGHPC, IEICE, and IPSJ.