Task Mapping on a Dragonfly Supercomputer

Ozan Tuncer
Boston University
Boston, MA, USA
otuncer@bu.edu

Yijia Zhang
Boston University
Boston, MA, USA
zhangyj@bu.edu

Vitus J. Leung
Sandia National Laboratories
Albuquerque, NM, USA
vjleung@sandia.gov

Ayse K. Coskun
Boston University
Boston, MA, USA
acoskun@bu.edu

***Abstract*—The dragonfly network topology has recently gained traction in the design of high performance computing (HPC) systems and has been implemented in large-scale supercomputers. The impact of task mapping, i.e., placement of MPI ranks onto compute cores, on the communication performance of applications on dragonfly networks has not been comprehensively investigated on real large-scale systems. This paper demonstrates that task mapping affects the communication overhead significantly in dragonflies and the magnitude of this effect is sensitive to the application, job size, and the OpenMP settings. Among the three task mapping algorithms we study (in-order, random, and recursive coordinate bisection), selecting a suitable task mapper reduces application communication time by up to 47%.**

I. INTRODUCTION

Optimizing performance continues to be key objective in HPC, and the role of inter-process communication in performance calls for strategies to reduce communication latency by ensuring data locality. Task mapping, i.e., the process of placing the MPI ranks of a parallel program onto the compute cores designated by the system software, can effectively improve locality. Recent work has shown 34% application running time reduction by selecting a better task mapping method [1], resulting in both higher resource utilization and lower energy consumption.

Dragonfly is an emerging hierarchical network topology for HPC systems due to its low diameter and high bisection bandwidth [2]. Dragonfly is composed of groups of routers which act as high-radix virtual routers connected to compute nodes. Prior work has studied task mapping on dragonflies mostly through simulations [3], [4], [5], [6]. Using small-scale experiments (up to 256 nodes) on a real Cray XC30 system, a recent study concluded that the impact of task mapping is minimal on dragonflies [7].

In this paper, we experiment with three task mapping algorithms, *in-order, random, and recursive coordinate bisection (RCB)*, on a real large-scale dragonfly system, and demonstrate that the impact of task mapping on communication overhead becomes significant for large-scale dragonfly networks.

II. EXPERIMENTAL METHODOLOGY

We assess the impact of task mapping on dragonfly systems using the Trinity supercomputer[1], two mini-applications developed by the Department of Energy community for performance evaluation in HPC systems, and three task mapping algorithms.

A. Target System

Trinity is a 8.1PFlop/s, 4.2MW supercomputer with a Cray XC30 architecture. It consists of over 9000 compute nodes with 32 processing cores per node, and is the tenth most powerful supercomputer in the June 2017 Top500 list[2]. Trinity uses a dragonfly topology with 26 groups, each with 384 nodes. The nodes within a group are connected to each other with flattened butterfly topology, whereas the groups are connected to each other with all-to-all topology.

B. Applications

We use the Mantevo benchmark suite [8], which is designed for performance evaluation and network scaling studies and represents the computational cores of various HPC applications. We select two mini-applications that are sensitive to task mapping in torus networks: MiniGhost, which represents modeling of complex multi-dimensional problems such as large deformations and/or strong shocks, and MiniMD, which is a proxy for the force computations in molecular dynamics applications. While both applications focuses on a 3D problem with nearest-neighbor communication, MiniMD also uses `MPI_Allreduce` for FFT calculations and sends messages to the MPI ranks that are not nearest neighbors but a few hops away from the source rank in the problem geometry, in a single time step.

C. Task Mapping Algorithms

We use the following three task mapping approaches:

- **In-order** is the default task mapper. It assigns the MPI ranks in-order to the cores of the allocated compute nodes, which are sorted by the allocation order.
- **Random** randomly assigns the MPI ranks to cores.
- **RCB** [1] recursively splits the allocated system nodes as well as the MPI ranks of a given 3-D application into equal halves based on the x, y, and z coordinates of the nodes/ranks. In both network space and the application space, the split is performed on the longest dimension. At the end of recursive splits, the remaining rank is mapped to the remaining core. RCB is originally built for 3-D mesh topologies. To adapt this algorithm to dragonfly, we use the group number of a compute node as its z-coordinate, and the row and column numbers within the group as the x- and y-coordinates of that node. While

[1]Trinity supercomputer: http://www.lanl.gov/projects/trinity/

[2]Top 500 Supercomputer Sites: http://www.top500.org/

our adapted version loses some information on the exact dragonfly topology such as the global link locations, it can reduce the distance messages must travel.

D. Experiments Conducted

We run the selected applications on 1, 2, 4, ..., 4096 nodes. For each application size (i.e., number of nodes), we repeat our experiments 8 times using different sets of nodes that are assigned by the system software depending on the system state. For each node allocation, we fully utilize the given nodes by running one thread on each core using 6 different openMP settings, where we use 1, 2, 4, ..., 32 threads per MPI rank. For each OpenMP setting, we re-run the same application using different task mappers.

III. RESULTS

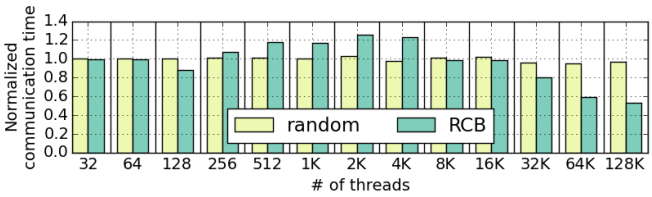
Figure 1 shows the time spent during MPI communication as reported by the applications. For each set of parameters, the communication time with the task mappers are normalized with respect to the in-order (default) mapper. To eliminate the impact of node allocation on the results, we show the median communication time (out of the 8 runs). In our experiments, the mapping overhead is negligible compared to application communication times.

Our results demonstrate that task mapping can change the communication time significantly when running parallel programs on dragonfly networks. In Fig. 1(a), RCB mapper reduces the communication time by 47% when MiniGhost is running on 128K threads with 1 thread per rank, whereas in Fig. 1(c), random mapper increases the communication time by 210% when MiniMD is running on 64K threads with the OpenMP setting as 1 thread per rank.

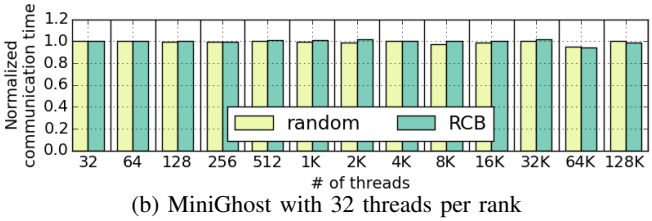
Because the two applications differ in their communication patterns, they benefit from different task mapping strategies. In Fig. 1(a) when running MiniGhost, RCB is up to 47% better than in-order mapper; meanwhile in Fig. 1(c) when running MiniMD, in-order mapper is always better than the others.

We find that the task mapper performance is also sensitive to application scale. Along the horizontal axis in Fig. 1(a), we see that the normalized communication time of RCB mapper varies more than 134%, from 0.53 to 1.25. RCB tends to perform worse than in-order mapper with less than 8K threads (corresponds to 256 nodes in our system), whereas with more than 8K threads, RCB turns out to be the best choice among the three task mappers. These results show that conclusions from small-scale experiments may not be extended to large-scale experiments.

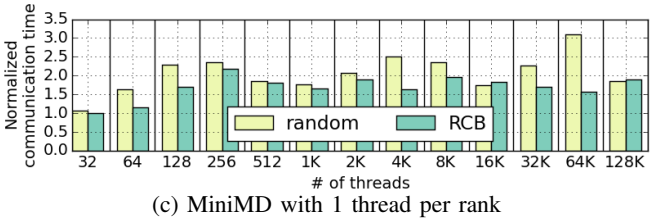
Another factor that affects the impact of task mapping is the OpenMP settings (i.e., number of threads per rank). While the performance difference between task mappers is less than 7% in Fig. 1(b), with a different threads-per-rank setting in Fig. 1(a), the performance difference reaches up to 47%.



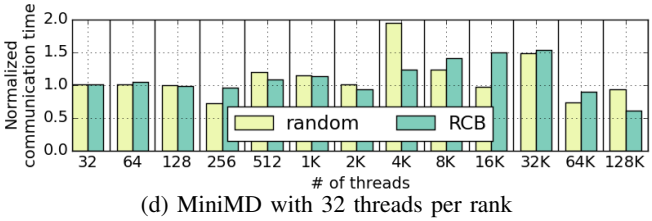
(a) MiniGhost with 1 thread per rank



(b) MiniGhost with 32 threads per rank



(c) MiniMD with 1 thread per rank



(d) MiniMD with 32 threads per rank

Fig. 1: Application communication time normalized with respect to the in-order task mapper. The results show that task mapping affects the communication overhead significantly.

ACKNOWLEDGMENT

This work has been partially funded by Sandia National Laboratories (SNL). SNL is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-NA0003525.

REFERENCES

[1] M. Deveci *et al.*, "Exploiting geometric partitioning in task mapping for parallel computers," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2014, pp. 27–36.

[2] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 77–88.

[3] V. T. Chakaravarthy *et al.*, "Mapping strategies for the PERCS architecture," *Intl. Conf. on High Performance Computing, (HiPC)*, 2012.

[4] A. Bhatele, N. Jain, W. D. Gropp, and L. V. Kale, "Avoiding hot-spots on two-level direct networks," *Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–11, 2011.

[5] B. Prisacari *et al.*, "Efficient task placement and routing in dragonfly networks," in *Proceedings of the 23rd ACM International Symposium on High-Performance Parallel and Distributed Computing. ACM*, 2014.

[6] F. Kaplan *et al.*, "Unveiling the Interplay Between Global Link Arrangements and Network Management Algorithms on Dragonfly Networks," *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2017.

[7] R. D. Budiardja, L. Crosby, and H. You, "Effect of rank placement on cray xc30 communication cost," in *The Cray User Group Meeting*, 2013.

[8] M. A. Heroux *et al.*, "Improving performance via mini-applications," *Sandia Nat. Lab., Tech. Rep. SAND2009-5574*, vol. 3, 2009.