Brief Announcement: Coloring-based Task Mapping for Dragonfly Systems

Ink Chinavinijkul
Knox College
Galesburg, IL, USA
cchinavinijkul@knox.edu

Jacob Newcomb
Knox College
Galesburg, IL, USA
jlnewcomb@knox.edu

Lingzhi Xi
Knox College
Galesburg, IL, USA
lxi@knox.edu

David P. Bunde
Knox College
Galesburg, IL, USA
dbunde@knox.edu

ABSTRACT

Task mapping is the assignment of job tasks to nodes. Traditionally, the goal of task mapping is to maximize locality, reducing the number of network hops needed to deliver messages as a way of reducing bandwidth consumption. We show that on a Dragonfly topology, such a strategy can be counterproductive because, while traffic is reduced, it is also concentrated on only a few global links, creating hot spots. We formulate the balanced adjacency coloring problem to design mappings that evenly spread network traffic, give optimal algorithms to solve it for a number of cases, and use simulations to show that mappings based on these algorithms can reduce the communication time of a stencil job by up to 20%.

CCS CONCEPTS

• **Computing methodologies → Parallel algorithms**;

KEYWORDS

Dragonfly topology; task mapping; stencil communication; balanced adjacency coloring

ACM Reference Format:
Ink Chinavinijkul, Jacob Newcomb, Lingzhi Xi, and David P. Bunde. 2018. Brief Announcement: Coloring-based Task Mapping for Dragonfly Systems. In *SPAA '18: 30th ACM Symposium on Parallelism in Algorithms and Architectures, July 16–18, 2018, Vienna, Austria.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3210377.3210665

1 INTRODUCTION

This paper looks at task mapping for stencil jobs on Dragonfly systems. Dragonfly [7] is a hierarchical network topology based on a tuple of parameters (p, a, h). To construct a Dragonfly system, p compute nodes are connected to each network switch. The switches are organized into groups of size a, with all switches in a group connected by *local links*. In addition, each switch also has h *global links* to other groups, arranged to give a single global link between each pair of groups. A big selling point for Dragonfly is its constant network diameter; each pair of switches is separated by at most three hops, one local link within the group of the source switch, one global link to the group of the destination switch, and then a final local link to the destination switch.

A stencil job is one using a nearest neighbor communication pattern in which tasks correspond to cells of a 2D or 3D grid and each task communicates with the adjacent tasks, those with whom it shares an edge in 2D or a face in 3D. This is an important communication pattern for scientific computations, with each task simulating its geographic region and periodically exchanging boundary conditions with its neighbors.

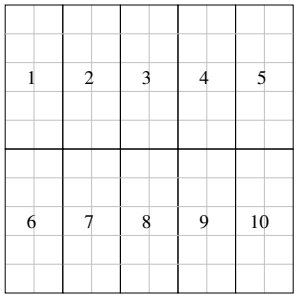
Task mapping is the assignment of job tasks to nodes. It is performed after the job is started, meaning that the job has already been assigned specific nodes. For jobs running MPI, task mapping is the assignment of roles to MPI ranks. Task mapping has been shown to significantly effect stencil job performance on a mesh topology [3]. Several researchers have previously considered its effect for stencil jobs on Dragonfly systems [1, 2, 5, 6, 8]. Several of these, starting with Bhatele et al. [1], utilize *blocking*, which maps a contiguous group of cells to the nodes of each switch and/or group. Blocking increases the proportion of communication that stays within a switch or group. Unfortunately, it can interact poorly with the hierarchical structure of Dragonfly. For example, consider the blocked mapping shown in Figure 1(a), in which a mesh is divided into 2×5 blocks, each assigned to a single group (numbers shown). With this mapping, group 1 needs to exchange 5 cells of boundary information in the horizontal direction with group 2, 2 cells worth in the vertical direction with group 6, and nothing with any other group. If the network uses shortest path routing, this causes a large network load imbalance, which would worsen with mesh size.

The network imbalance of blocking led some to advocate for random mapping of cells to nodes [1], but this has been shown to lose about half the potential throughput in several cases [6]. Instead, we propose a type of coloring called *balanced adjacency coloring* that balances the number of times each pair of colors is adjacent. A mapping based on this coloring is shown in Figure 1(b). Every pair of groups is adjacent 4 times, giving perfect network load balance.

A possible alternative to clever task mapping is improved routing. The strategy of always using the local-global-local link pattern is called *minimal routing*. As observed above, this can lead to hotspots on global links. One solution, called Valiant routing [7], routes each

SPAA '18, July 16–18, 2018, Vienna, Austria

ACM ISBN 978-1-4503-5799-9/18/07.
https://doi.org/10.1145/3210377.3210665



(a)

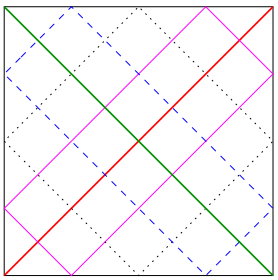
1	2	3	4	5	6	7	8	9	10
3	1	5	2	7	4	9	6	10	8
2	4	1	6	3	8	5	10	7	9
5	3	7	1	9	2	10	4	8	6
4	6	2	8	1	10	3	9	5	7
7	5	9	3	10	1	8	2	6	4
6	8	4	10	2	9	1	7	3	5
9	7	10	5	8	3	6	1	4	2
8	10	6	9	4	7	2	5	1	3
10	9	8	7	6	5	4	3	2	1

(b)

Figure 1: Two mappings (group numbers shown). (a) Blocking. (b) Balanced adjacency coloring.

1	2	3	4	5	6	7	8
3	1	5	2	7	4	8	6
2	4	1	6	3	8	5	7
5	3	7	1	8	2	6	4
4	6	2	8	1	7	3	5
7	5	8	3	6	1	4	2
6	8	4	7	2	5	1	3
8	7	6	5	4	3	2	1

(a)



(b)

Figure 2: An 8×8 balanced adjacency coloring. (a) Coloring itself. (b) Lines and cycles generating it.

message by way of a randomly-chosen intermediate group. This increases the maximum distance between switches to five hops: local link, global link to intermediate group, local link within intermediate group, global link to destination group, and local link. Although it uses longer paths, Valiant routing spreads traffic more evenly among the links. Work on routing continues (e.g. [4]), with most of the focus on *adaptive routing*, which adjusts the routing based on the congestion level. We use UGAL-L [7] as a representative of adaptive routing; it chooses between minimal and Valiant routing on a per-packet basis using an estimate of network latency. Our results show that balanced mapping achieves better performance than good routing on top of a baseline mapping.

Also related is work of Zhang et al. [9] on allocation, the assignment of nodes to jobs. Their algorithm strives to localize small jobs, but to spread out large jobs to balance the global link load. The goal is thus similar to ours, but their algorithm does not apply in this work, which concerns jobs occupying the entire machine.

2 BALANCED ADJACENCY COLORING

We define a *balanced adjacency coloring* of an $n \times m$ mesh as a coloring using k colors with the following properties:

(1) Every color is used either $\lfloor nm/k \rfloor$ or $\lceil nm/k \rceil$ times.
(2) Each pair of colors is adjacent $\lfloor A/\binom{k}{2} \rfloor$ or $\lceil A/\binom{k}{2} \rceil$ times, where A is the number of adjacent cells with distinct colors.

The conditions balance the group workload and the load on global links, respectively.

Chakaravarthy et al. [2] also proposed a coloring problem to capture task mapping in a Dragonfly system, but they required that all neighbors of each color be distinct. Balanced adjacency coloring is a generalization of their notion. Their algorithms are fairly restricted, but incomparable with the results of this paper.

Our main algorithm colors an $n \times n$ mesh with n colors. Initially, assume n is even. The algorithm begins with a permutation of the colors on one row, which we call the *reference row*.

To extend the coloring of the reference row to the rest of the mesh, we consider a collection of lines and cycles in a visualization of the mesh with square unit cells as shown in Figure 2(b). The two lines are created by joining opposite corners. Each cycle has a starting point (x, n) satisfying $1 < x < n$ with x even. It proceeds at slope -1 to the mesh's right edge ($x = n$). Each time an edge of the mesh is encountered, change the slope as if it is reflecting off that edge. We say a mesh cell is *on* a line or cycle if the line or cycle passes through the cell's geometric center.

It is not hard to show that every mesh cell is on one line or cycle. To assign colors based on the lines and cycles, we copy colors of the reference row along them. The reference row has one cell on each line and we copy the color of this cell to all the line's cells. The reference row has two cells on each cycle and we alternate these colors around the cycle. Following this process using the lines and cycles in Figure 2(b) gives the coloring in Figure 2(a).

This algorithm uses every color n times, either because it occupies an entire diagonal or because it alternates around a cycle which is twice as long. The remaining step to show we have a balanced adjacency coloring is to count color adjacencies:

Lemma 1. *Each pair of colors is adjacent 4 times in the coloring.*

Proof. We count the adjacencies within and between the lines and cycles. Lines contain no adjacent cells. This is true of cycles except at the 4 mesh boundaries, when they reflect and cross adjacent cells, each making one adjacency between the cycle's colors.

The two lines intersect once. Since lines contain only a single color, each diagonally-opposite pair of cells around the intersection point have a shared color. These pairs of cells make four adjacencies.

Consider a line-cycle intersection. The diagonally-opposite cells along the line share its color. Since colors alternate on the cycle, each of the other cells has a distinct color of the cycle. Each of these is adjacent with the line's color twice. The line and cycle intersect twice, doubling the number of adjacencies to four.

In a cycle-cycle intersection, diagonally-opposite cells are split between the cycle's colors. Thus, the intersection forms 1 adjacency for each pair of colors in different cycles. Each pair of cycles has 4 intersections, creating 4 adjacencies between their colors. □

Generalizations. The algorithm when n is odd is the same as the even case except only one line is used. It also generalizes to algorithms for an $n \times n$ mesh using $2n$ colors (use 4 colors per cycle and two per line) and an $n \times n \times n$ cube (color 2D slices).

3 MAPPINGS

The obvious way to turn a balanced adjacency coloring into a mapping is to assign each mesh cell to a node in the group indicated

Mesh	Groups used	System (p, a, h)	System nodes
8×8	8	(2, 4, 2)	72
32×32	32	(4, 8, 4)	1,056
72×72	72	(6, 12, 6)	5,256
128×128	128	(8, 16, 8)	16,512
200×200	200	(10, 20, 10)	40,200

Figure 3: Description of simulated systems and jobs.

by its color. This puts all communication over global links and performs poorly. Thus, our algorithm uses a small amount of blocking. Specifically, we divide the job into 2×2 submeshes to form a $(n/2) \times (n/2)$ mesh of task units, each containing 4 tasks. The mesh of task units is then colored using the generalization that uses twice as many colors as the mesh dimensions. These colors are used to assign the task units to groups. We call this algorithm *Blocking Balanced Adjacency Coloring* (B-BAC) because it uses some blocking to increase traffic within groups but primarily follows the coloring.

As a baseline, we use the Blocked Supernodes Mapping (BSM) by Bhatele et al. [1]. It maximizes locality by assigning a contiguous block for each group and contiguous subblocks for each switch.

We compare the mappings with simulations of a *capability workload*, where a single job runs on most of the machine. This is an important workload for the largest systems, whose construction is justified by the need to run huge jobs. We tested a range of job and system sizes, shown in Figure 3. All jobs are square and use all but one group of a system. All systems have $2p = a = 2h$ for balance.

We used a packet-level simulator, with local and global link latencies of 100, virtual channel buffer sizes of 256, and ISLIP for buffer allocation with an internal speedup of 1.5. We assume a communication-intensive job, with boundary exchanges sending 30 packets each way. Each task sends to its neighbors, waits for their message, computes for 5 time units, and repeats for 10 rounds.

Figure 4 shows the simulation results comparing the blocking-based baseline strategy with B-BAC. Looking just at the results for minimal routing (first two bars for each mesh size), we see that B-BAC is very slightly worse than the baseline for the smallest size (less than 0.02%), but then consistently better, with a percent improvement generally in the mid-teens (16%, 8%, 14%, 17.5% respectively). Looking at the results for UGAL-L routing (last two bars for each size), B-BAC is worse for the two smaller sizes (by 1.5% and 4%), but better at the larger sizes (by 5%, 7%, and 20%). For both routing algorithms, B-BAC wins at larger job sizes.

Of particular note is that B-BAC with minimal routing outperforms both mappings with adaptive routing in nearly all cases. In fact, for a given mapping, UGAL-L routing is nearly always worse than minimal routing. This is initially surprising since UGAL-L is adaptive and can always revert to minimal routing. In intense communication patterns such as ours, however, the penalty for extra hops can be particularly high since they consume extra bandwidth. With B-BAC mapping, nearly all global links are equally loaded so Valiant routes are nearly always bad. The baseline strategy has many unused global links, but the results show that adaptive routing cannot compensate for the initial poor link load balance.

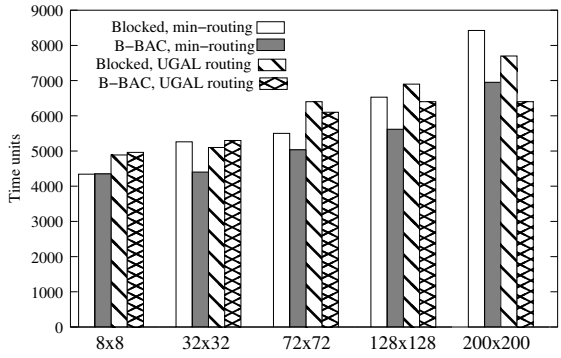


Figure 4: Simulated job times for different mesh sizes. Compares blocking-based mapping and coloring-based (B-BAC) for minimal and UGAL-L routing.

4 DISCUSSION

Our results show the potential of spreading the network traffic more fairly, even at the cost of increasing the total amount of global traffic. Going forward, one idea is to block at the switch level so that all nodes of a switch are working on a single mesh submesh. We used 2×2 submeshes since non-square submeshes make the traffic sent between neighbors depend on the direction due to the different number of cells along each boundary. This could be overcome with a weighted coloring. Another goal is to find colorings for non-square meshes and graphs representing other communication patterns.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation under grant CNS-1423413 and the Paul K. and Evalyn Richter Memorial Funds.

REFERENCES

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

[2] V.T. Chakaravarthy, M. Kedia, Y. Sabharwal, N.P.K. Katta, R. Rajamony, and A. Ramanan. 2012. Mapping Strategies for the PERCS Architecture. In *Proc. 19th Intern. Conf. High Performance Computing (HiPC)*.

[3] M. Deveci, S. Rajamanickam, V.J. Leung, K.T. Pedretti, S.L. Olivier, D.P. Bunde, Ü.V. Çatalyürek, and K.D. Devine. 2014. Exploiting Geometric Partitioning in Task Mapping for Parallel Computers. In *Proc. 28th IEEE Intern. Parallel and Distributed Processing Symp. (IPDPS)*.

[4] P. Faizian, M.A. Mollah, Z. Tong, X. Yuan, and M. Lang. 2017. A Comparative Study of SDN and Adaptive Routing on Dragonfly Networks. In *Proc. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*.

[5] N. Jain, A. Bhatele, X. Ni, N.J. Wright, and L.V. Kale. 2014. Maximizing throughput on a dragonfly network. In *Proc. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*.

[6] A. Jokanovic, B. Prisacari, G. Rodriguez, and C. Minkenberg. 2013. Randomizing task placement does not randomize traffic (enough). In *Proc. 2013 Interconnection Network Architecture: On-Chip, Multi-Chip (IMA-OCMC)*. 9–12.

[7] J. Kim, W. Dally, S. Scott, and D. Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Proc. 35th Ann. Intern. Symp. Comput. Arch. (ISCA)*. 77–78.

[8] B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, and T. Hoefler. 2014. Efficient Task Placement and Routing in Dragonfly Networks. In *Proc. 23rd ACM Intern. Symp. High-Performance Parallel and Distributed Computing (HPDC)*.

[9] Y. Zhang, O. Tuncer, F. Kaplan, K. Olcoz, V.J. Leung, and A.K. Coskun. 2018. Level-Spread: A new job allocation policy for Dragonfly networks. In *Proc. 32nd IEEE Intern. Parallel and Distributed Processing Symp. (IPDPS)*.