Efficient Task Placement and Routing of Nearest Neighbor Exchanges in Dragonfly Networks

Bogdan Prisacari
IBM Research – Zurich
Säumerstrasse 4
8803 Rüschlikon, Switzerland
bpr@zurich.ibm.com

German Rodriguez
IBM Research – Zurich
Säumerstrasse 4
8803 Rüschlikon, Switzerland
rod@zurich.ibm.com

Philip Heidelberger
IBM Research – Watson
1101 Kitchawan Rd Rte 134
Yorktown Heights, NY 10598
philiph@us.ibm.com

Dong Chen
IBM Research – Watson
1101 Kitchawan Rd Rte 134
Yorktown Heights, NY 10598
chendong@us.ibm.com

Cyriel Minkenberg
IBM Research – Zurich
Säumerstrasse 4
8803 Rüschlikon, Switzerland
sil@zurich.ibm.com

Torsten Hoefler
ETH Zurich
Universitätstrasse 6
8092 Zürich, Switzerland
htor@inf.ethz.ch

ABSTRACT

Dragonflies are recent network designs that are one of the most promising topologies for the Exascale effort due to their scalability and cost. While being able to achieve very high throughput under random uniform all-to-all traffic, this type of network can experience significant performance degradation for other common high performance computing workloads such as stencil (multi-dimensional nearest neighbor) patterns. Often, the lack of peak performance is caused by an insufficient understanding of the interaction between the workload and the network, and an insufficient understanding of how application specific task-to-node mapping strategies can serve as optimization vehicles.

To address these issues, we propose a theoretical performance analysis framework that takes as inputs a network specification and a traffic demand matrix characterizing an arbitrary workload and is able to predict where bottlenecks will occur in the network and what their impact will be on the effective sustainable injection bandwidth. We then focus our analysis on a specific high-interest communication pattern, the multi-dimensional Cartesian nearest neighbor exchange, and provide analytic bounds (owing to bottlenecks in the remote links of the Dragonfly) on its expected performance across a multitude of possible mapping strategies.

Finally, using a comprehensive set of simulations results, we validate the correctness of the theoretical approach and in the process address some misconceptions regarding Dragonfly network behavior and evaluation, (such as the choice of throughput maximization over workload completion time minimization as optimization objective) and the question of whether the standard notion of Dragonfly balance can be extended to workloads other than uniform random traffic.

HPDC'14, June 23–27, Vancouver, BC, Canada.

ACM 978-1-4503-2749-7/14/06 ...$15.00.
http://dx.doi.org/10.1145/2600212.2600225.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.4 [**Performance of Systems**]

General Terms

Theory, Performance, Experimentation

Keywords

Dragonfly networks, Stencil computation, Nearest neighbor exchanges, Direct and indirect routing, Cartesian and random task placement

1. INTRODUCTION

High-Performance Computing (HPC) and datacenter networks are continuously growing in scale as larger problems are tackled, with several large scale systems already in use, especially in the HPC space. Technological advances in switch and cabling technology enabled Dragonfly, a new economic and efficient network topology developed independently by IBM (as the interconnection fabric of the PERCS system [2]) and Kim et al. [13]. Dragonfly networks combine high-radix switches and a mix of copper and electrical cables into a hierarchical two-tier topology. Both tiers are logically fully connected, a structure that guarantees low latency and high bisection bandwidth.

Dragonfly networks are scaled in practice to more than 5,200 nodes in Europe's most powerful supercomputer Piz Daint. Their theoretical scalability exceeds tens of thousands of nodes while achieving nearly full bandwidth for random uniform traffic patterns and shortest path routing. However, for deterministic patterns, a common characteristic of scientific applications, the bandwidth depends largely on the routing scheme employed (e.g., indirect random or adaptive) and the task-to-node mapping (e.g., random, blocked, or block-cyclic). The exact tradeoffs between routing and mapping for specific communication patterns are not well understood for Dragonfly topologies.

Cartesian *nearest neighbor exchanges* are very common in scientific computations [12]. They often represent a discretization of a physical system, which is modeled by a set

of elements that are arranged in a grid. A typical simulation, e.g., a heat propagation, then solves PDEs and ODEs on this grid to advance the simulated system time. The resulting computational structures are most often Cartesian structures called *stencils*. Stencils combine neighboring values of a grid point to compute its state in the next iteration. A distribution of this scheme requires communication in a Cartesian structure. This pattern is so typical that parallel programming schemes, such as the Message Passing Interface, provide explicit support [9].

Any deterministic traffic pattern may exhibit poor performance if the computation is mapped unfavorably to a Dragonfly [1, 3, 6, 13]. For example, we show in this paper that the completion time for randomly mapped stencil computations can be between 50% to 10 times larger than the best achievable completion time. Several related studies empirically analyzed the impact of routing [6, 13], topology [6] and task mapping [3] to support applications with deterministic communication patterns such as stencil. Those studies report significant improvements for random indirect routing or random task mapping. However, indirect routing reduces the available global network bandwidth by utilizing more links and random task mapping loses communication locality because neighborhoods are spread throughout the system. Both schemes increase the network load and the exact tradeoffs of the routing and mapping selections remain unclear in general.

In this work, we provide a clear set of guidelines how to configure the network for a given workload. For this, we derive a general theoretical model of communication performance of multi-dimensional stencil computations on arbitrary Dragonfly networks considering different domain decompositions and task placements. Our general model allows us to co-design the application decomposition for the class of Cartesian stencil applications with the ideal Dragonfly network configuration.

In summary, we guide the user to select the optimal values of the parameters

- domain decomposition,
- task placement (sparsity and randomization), and
- routing approach.

We also validate our theoretical results against detailed simulations of the targeted scenarios and conclude with a summary of practical recommendations on how to (1) configure an application to run on a Dragonfly network and (2) optimize the design of a Dragonfly network to achieve higher performance for the nearest neighbor exchange.

2. BACKGROUND AND RELATED WORK

Dragonfly topologies are highly scalable high-radix two-level direct networks with a good cost-performance ratio, used for example in the PERCS interconnect [2] and in the Cray Cascade [5] and likely to be one of the options chosen for many of the future Exascale systems.

A dragonfly is a two-level hierarchical network, where fully-connected groups of lower-radix switches at the first level form a virtual high-radix switch. These virtual high-radix switches form another fully-connected graph of groups [13]. The ports that the virtual high-radix switches use to connect to the other virtual switches are distributed across the physical switches that make up the virtual switch.

Dragonflies can be uniquely described by means of three parameters: p, the number of nodes connected to each switch, a, the number of switches in each first level group, and h, the number of channels that each switch uses to connect to switches in other groups. For certain values of these parameters it can be shown that close to ideal throughput can be achieved for uniform traffic.

For the Dragonfly networks studied here, shortest paths between pairs of nodes are unique[1]. The longest possible shortest path is made up of a traversal of a *local* (L) link in the first level group to get to the switch that has a *global* (R) link towards the destination group, a traversal of the R link and a second *local* link traversal in the destination group to get to the switch directly connected to the destination node. Figure 1 illustrates the topological layout of a Dragonfly network and the meaning of the (p, a, h) parameters.

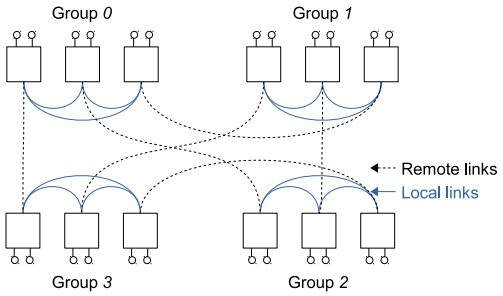


Figure 1: Example of a $(p = 2, a = 3, h = 1)$ Dragonfly topology. Every switch is connected to $p = 2$ nodes, there are $a = 3$ switches in every group and every switch has $h = 1$ links connecting it to switches outside its group.

However, this lack of shortest path diversity can lead to an extreme degradation in performance for certain adversarial traffic patterns [3, 6]. One option to alleviate this degradation is to use Valiant's algorithm [18]. This algorithm routes a packet to a randomly chosen intermediate node first, before routing it to the actual destination. The expectation is that, by using a different random intermediate node for each packet, the original nature of the traffic is shifted towards a uniform random distribution of traffic. However, this is done at the expense of longer paths and results in roughly a doubling of the load for an original traffic that is sufficiently dense (such as originally random uniform traffic [4]).

The longer paths in Valiant routing also have the disadvantage of requiring the use of additional virtual channels to guarantee deadlock freedom. In particular, the Valiant routing variant for dragonflies as described in [13], which we will call *Valiant [Kim:2008]*, requires 3 virtual channels (instead of 2 for shortest paths) for the L links and 2 virtual channels (instead of 1 for shortest path) for the R links, to guarantee deadlock freedom [2]. *Valiant [Kim:2008]* can be described as follows: when a source s in first-level group S sends a message to destination d in first-level group D, an intermediate misroute group is chosen (I). A minimal route (consisting of at most one L and one R hop) is taken to arrive to a switch in group I. Once the packet arrives to this intermediate group I, the packet follows the unique mini-

[1]More generally, there may be multiple such shortest paths, especially for networks that are smaller than the maximum possible scale.

mal route from the arriving switch at I to the destination d (requiring at most two L hops and one R hop).

The nearest neighbor exchange (NNE) is a common exchange pattern in many HPC applications. This pattern arises as the result of a decomposition of the domain of a problem into smaller elements. The computation for each element depends on a number of neighboring elements, where the neighborhood definition is dependent on both the specific problem and the specific way of performing the decomposition. In this work, we will consider Cartesian decompositions along a variable number of dimensions, and additionally we will consider that an element only exchanges data with elements with which it shares a (D-1)-dimensional contact plane. We will also make the assumption that the amount of exchanged data needed is proportional to the size of the (D-1)-dimensional contact plane.

A large fraction of HPC applications make use of the multi-dimensional nearest neighbor exchange pattern [12]. Relevant examples are Cactus [8], which uses a grid decomposition to solve Einstein's equations to study astrophysical phenomena, GTC [14] (Gyrokinetic Toroidal Code), which solves the gyrophase-averaged Vlasov-Poisson equations on a 3D toroidal domain, LBMHD [19] (Lattice Boltzmann methods for the problem of magneto-hydrodynamics (MHD)), which uses either a 2D or 3D lattice, or MILC, a 4D lattice QCD code. These represent only a small, but relevant fraction of the many codes which, with variations, rely on efficient nearest neighbor exchanges to achieve performance.

A recent paper [3] analyzed the performance achieved in dragonfly networks when executing such communication patterns with several task-placements and routing schemes (shortest path and Valiant). This work showed how a randomization of the task placement using shortest path routing achieved similar performance to a contiguous task placement using Valiant routing. Another work [11] also explored, for other patterns, the alternative of Valiant routing and of randomized task placement, and concluded that, while randomization improves performance over a naive contiguous placement, the performance achieved is still low when compared to the peak obtained for uniform random traffic, both with direct and indirect routing. Neither of these works studied in detail how the domain decomposition and a structured task placement influences the NNE performance on a dragonfly, and neither has provided a model that is able to predict NNE performance given the topology parameters, the domain decomposition, and the task placement onto a dragonfly.

3. THEORETICAL ANALYSIS

In order to explore the optimization opportunities for multi-dimensional near neighbor communication patterns in Dragonfly networks, we will provide theoretical estimates for the performance of applications executed as a set of concurrent tasks exhibiting arbitrarily shaped Cartesian multi-dimensional nearest neighbor communication patterns and being executed on a system interconnected via arbitrary Dragonfly networks. We will explore different task placement strategies and their effect on network utilization and performance. The end goal is to be able to determine guidelines as to how to map such applications onto the different nodes in the system such that communication performance is maximized.

This section provides the step by step derivation of these guidelines and is fairly technical. The reader that is interested mainly in the end result should read Section 3.2 where we introduce the main notations and then skip directly to Section 3.5.

3.1 Arbitrary workload performance in Dragonflies

We consider an arbitrary workload given by a traffic demand matrix T, with as many columns and rows as concurrent tasks perform the workload. Each element t_{sd} of the matrix represents the total number of bytes sent by task s to task d. We denote by $\hat{T}$ the normalized traffic demand matrix, where every element $\hat{t}_{sd}$ is the corresponding element in T divided by the total number of bytes sent by source task s.

$$\hat{t}_{sd} = t_{sd} / \sum_{d} t_{sd} \tag{1}$$

The goal of this subsection is to formalize the effective injection throughput that will be sustainable at the nodes by estimating the demand that the workload will impose on the network. We will denote this effective injection bandwidth for node n by $B^n_{\mathrm{P,eff}}$. If a link in the network receives more demand than it can sustain, then all sources whose messages periodically use that link will see their effective throughput eventually decreased, to the point at which the demand on the link no longer exceeds its capacity.

The exact distribution of demand to network links will intrinsically be linked to the routing approaches used in the network. Therefore, the routing approach will in turn be a significant factor that will determine the effective performance of the network. For the Dragonfly, we will consider two routing approaches:

1. Dragonfly direct routing, where messages are always sent through the network across shortest paths;

2. Dragonfly indirect routing, where messages are always sent through the network across indirect paths and the indirect paths between any given (source,destination) pair are used evenly via randomization. This includes the case where the source and destination belong to the same group.

Let us consider the entire duration of a workload, or, for ongoing workloads, a large enough amount of time such that all sources inject a statistically significant amount of traffic in the network, i.e., an amount of traffic that roughly obeys the distribution expressed by $\hat{T}$. We define the demand exercised on a given link as being the total number of bytes that need to cross that link in the considered amount of time.

Let us consider a remote link R_{ij} connecting the Dragonfly group G_i to the Dragonfly group G_j, $i \neq j$. The demand $\Delta^{\mathrm{R,direct}}_{ij}$ of that link under the direct routing approach will be equal to the total fraction of traffic sent by sources in G_i that have as a destination any of the tasks in G_j.

$$\Delta^{\mathrm{R,direct}}_{ij} = \sum_{s \in G_i} \sum_{d \in G_j} \hat{t}_{sd} \cdot B^s_{\mathrm{P,eff}}. \tag{2}$$

The demand $\Delta^{\mathrm{R,indirect}}_{ij}$ of the same link under the indirect routing approach has two distinct components. The first component consists of traffic sent by sources in G_i to any destination outside of G_j and G_i, using G_j as an intermediate routing point. The second component consists of

traffic sent by sources anywhere outside of G_i and G_j that is sent to destinations in G_j using G_i as an intermediate routing point.

$$\Delta_{ij}^{\text{R,indirect}} = \sum_{p\neq i,j}\sum_{s\in G_i}\sum_{d\in G_p}\hat{t}_{sd}\cdot P(i \xrightarrow{j} p)\cdot B^s_{\text{P,eff}} + \sum_{p\neq i,j}\sum_{s\in G_p}\sum_{d\in G_j}\hat{t}_{sd}\cdot P(p \xrightarrow{i} j)\cdot B^s_{\text{P,eff}} \quad (3)$$

where $P(i \xrightarrow{p} j)$ denotes the probability that the indirect route via group p is chosen when routing from group i to group j.

Concerning the demand imposed on the intra-group links, In contrast to the inter-group network structure, current Dragonfly systems do not exhibit a unique intra-group structure (e.g., the original Dragonfly design uses a standard full mesh, the PERCS interconnect uses a non-uniform (from the bandwidth and latency points of view) full mesh whereas the Cray Cascade interconnect uses a two dimensional Hamming graph). This, coupled to the fact that in practice it is generally the inter-group bottlenecks that determine overall system performance, would make thoroughly analyzing the intra-group demand an unnecessarily complex endeavor. Nonetheless, we will take into account the possibility of bottlenecks shifting inside the groups and will analyze this scenario in detail in Section 4.3.

We will assume that all nodes and switches in the network are identical and thus we can denote by:

- $B_P \rightarrow$ the injection bandwidth available to any node in the network,
- $B_L \rightarrow$ the bandwidth of each local link,
- $B_R \rightarrow$ the bandwidth of each remote link.

We will further make a simplifying assumption that the workload does not induce or is not allowed to induce unfairness in the system, that is, every node will be able to effectively inject the same amount of traffic as any other node. Then, $B_{\text{P,eff}}$ can also be considered uniform across nodes and we can introduce the notion of relative demand δ as being the demand Δ defined above divided by the effective injection bandwidth $B_{\text{P,eff}}$.

$$\begin{aligned}\delta_{ij}^{\text{R,direct}} &= \Delta_{ij}^{\text{R,direct}}/B_{\text{P,eff}}\\ \delta_{ij}^{\text{R,indirect}} &= \Delta_{ij}^{\text{R,indirect}}/B_{\text{P,eff}}\end{aligned} \quad (4)$$

Given that the demand Δ on a link cannot exceed the bandwidth of that link, the achievable effective bandwidth is consequently upper bounded by the ratio between a link's bandwidth and the relative demand induced on the link. The throughput limitation that the network imposes on the nodes can thus be formally expressed by Eq. (5) for direct routing and Eq. (6) for indirect routing.

$$B_{\text{P,eff}}^{\text{direct}} \leq \min\left(B_P, \frac{B_L}{\max\limits_{\text{Llinks}}(\delta^{\text{L,direct}})}, \frac{B_R}{\max\limits_{\text{Rlinks}}(\delta^{\text{R,direct}})}\right) \quad (5)$$

$$B_{\text{P,eff}}^{\text{indirect}} \leq \min\left(B_P, \frac{B_L}{\max\limits_{\text{Llinks}}(\delta^{\text{L,indirect}})}, \frac{B_R}{\max\limits_{\text{Rlinks}}(\delta^{\text{R,indirect}})}\right) \quad (6)$$

Eq. (5) and (6) coupled to the load formulations (Eq. (2) and (3)) and to a particular traffic demand matrix allow us to predict network performance for arbitrary workloads in arbitrary dragonflies.

3.2 Formal description of targeted workloads

The workload we focus on in this work are is the multi-dimensional Cartesian nearest neighbor communication. We will assume that the concurrent tasks are solving a problem pertaining to a d-dimensional domain that is intrinsically split (along directions parallel to the coordinate axes) into equally sized d-dimensional elements in a way that is consistent with the domain's structure (e.g., if the domain is larger in one dimension, it will have proportionally more elements along that dimension). The number of elements in each dimension is given by a vector $\alpha \in \mathbb{N}^d$, for a total number of elements $|\alpha| = \Pi_{k=1}^d \alpha_k$. The assignment of the elements to computation tasks is also done along a Cartesian grid, such that the number of tasks in each dimension is given by a different vector $\beta \in \mathbb{N}^d$. Every task will thus be assigned one or several elements, the number of elements per dimension assigned to each task being α divided element by element by β. The total number of tasks is $|\beta| = \Pi_{k=1}^d \beta_k$. Any task can be naturally identified via a d-dimensional vector $x \in \mathbb{N}^d$ such that $0 \leq x_k < \beta_k, \forall 1 \leq k \leq d$. These concepts are illustrated in Fig. 2 for a two-dimensional application domain.

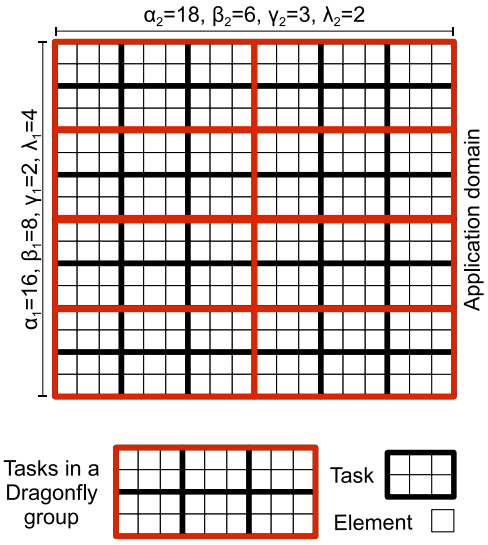


Figure 2: Application domain for a two-dimensional nearest neighbor exchange. The domain admits an intrinsic decomposition into identical elements (the small squares in the figure) along the axes of the Cartesian domain. This decomposition is characterized by the two-dimensional vector α which in the example figure takes the value $\alpha = (16, 18)$. The application is run as a set of concurrent tasks that is each assigned a Cartesian subset of the element grid (the 2×3 rectangles in the figure). Each subset assigned to a task has the same size along every dimension (in the figure, size 2 in the first dimension and size 3 in the second). The sub-decomposition is determined by a second vector β that determines how many tasks there are in each dimension (in the figure, $\beta = (8, 6)$).

The inter-task communication pattern considered is the nearest neighbor exchange of distance 1 on the d-dimensional Cartesian grid, where a task that is identified by the vector x communicates with a task identified by the vector y if and only if x and y share $d-1$ coordinates, while the remaining coordinate values x_k and y_k satisfy the relationship $(\beta_k + x_k - y_k) \bmod \beta_k = 1$. We will call two such tasks *neighbors along the k dimension*. We define the function $\nu^k_{x,y}$ which takes a value of 1 if tasks corresponding to x and y are neighbors along the k dimension and 0 otherwise, as follows:

$$\nu^k_{x,y} = \begin{cases} 1 & \text{if } (\beta_k + x_k - y_k) \bmod \beta_k = 1 \\ & \text{and } \forall i \neq k, x_i = y_i \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

A very important design choice is the exact manner in which to map the application domain to the network topology. We will consider two task-to-node mapping strategies. The first assumes a regular grouping of tasks in the groups of the Dragonfly. A set of tasks corresponding to a Cartesian application sub-domain is assigned to any given group such that this sub-domain will contain $\gamma \in \mathbb{N}^d$ tasks along each of the d axes. This leads to a total of $|\gamma| = \Pi^d_{k=1}\gamma_k$ tasks in each Dragonfly group. We further define the vector λ in $\mathbb{N}^d$ as the coordinate by coordinate ratio of β and γ: $\forall 1 \leq k \leq d, \lambda_k = \lceil \beta_k/\gamma_k \rceil$. λ will thus represent the d-dimensional decomposition of the application domain into Dragonfly groups. It entails that $|\lambda| = \Pi^d_{k=1}\lambda_k$ Dragonfly groups will be needed to host the entire set of tasks. The second strategy will assume a random placement of tasks within the Dragonfly, keeping however the task count per group to the same $|\gamma|$ value as in the Cartesian case, to allow for fair comparison.

This formalization of the traffic pattern allows us to explore two important performance affecting factors: the shape of the Cartesian intra-group sub-domains (in terms of number of dimensions and ratios between sizes along each dimension) and the level of sparsity (the proportion of nodes used to host the workload in every group). We consider sparse placements as they are becoming more and more common in practice, and are the default strategy in some of the more recent HPC systems [5]. Under a sparse allocation, the higher amount of network resources at the disposal of the application is balanced by an increased probability of interference with other simultaneously running applications. Our model estimates the impact of the placement decision under the assumption that interference from other applications is minimal. The resulting performance metric is a useful baseline for inter-application interference analysis, but the latter is out of the scope of this work.

We will also operate under the assumption that the assignment of tasks to nodes within a group is random (under the constraint that a node will be assigned at most one task), as is the assignment of Cartesian groups of tasks to specific Dragonfly groups. This is a reasonable choice to make as the structure of both the intra-group and inter-group networks is vertex-symmetric. Furthermore, such a placement can prove useful in practice as the randomization has the benefit of disrupting regular patterns that may otherwise cause load imbalance [16].

3.3 Performance evaluation metrics

Two typical metrics are used for evaluating the performance of a fixed-size workload (where a fixed amount of data needs to be exchanged across the network) such as the nearest neighbor exchange. One metric is the completion time, i.e., the time between the moment when the first message enters the network and the moment when the last message is delivered. The second is the average effective throughput, defined as the total number of bytes exchanged divided by the completion time and averaged across the total number of communicating tasks.

Finally, as we are dealing with different shapes of the application domain (as expressed by the α vector) and different ways of partitioning the domain into tasks (as expressed by the β vector), the messages exchanged between tasks will vary in size. Specifically, they will be proportional to the size of the surface separating the communicating tasks. This entails that messages exchanged along the k-th axis, when such an exchange takes place, will have a size μ_k given by

$$\mu_k = \mu \cdot \Pi^d_{i=1, i\neq k} \frac{\alpha_i}{\beta_i} = \mu \cdot \frac{|\alpha|}{|\beta|} \cdot \frac{\beta_k}{\alpha_k}. \quad (8)$$

3.4 Nearest neighbor communication performance in dragonflies

Two sets of factors determine the theoretical performance of a workload: the $\hat{t}_{sd}$ coefficients of the normalized traffic demand matrix and the $P(i \xrightarrow{p} j)$ routing probabilities in the case of indirect routing. Given the indirect routing assumption stated in 3.1 the value of the latter is

$$P(i \xrightarrow{p} j) = \frac{1}{ah-1}. \quad (9)$$

For the nearest neighbor exchange, we compute the traffic matrix as follows. We denote by m_k the proportion of messages (out of a task's entire communication workload) sent to a neighbor along the k-th axis. Due to the way we perform the decomposition of the domain into elements, the way we assign elements to tasks and the way we define the neighborhood of a task (7), it follows that a given task either does not exchange any messages along axis k (when $\beta_k = 1$), or it exchanges messages across two surfaces. If we denote by $\mathbb{1}_{condition}$ the indicator function that takes a value of 1 when the condition is true and a value of 0 otherwise, then the value of this ratio is

$$\begin{aligned} m_k &= \frac{\mu_k \cdot \mathbb{1}_{\beta_k > 1}}{2\sum^d_{i=1} \mu_i \cdot \mathbb{1}_{\beta_i > 1}} \\ &= \frac{\mu \cdot \frac{|\alpha|}{|\beta|} \cdot \frac{\beta_k}{\alpha_k} \cdot \mathbb{1}_{\beta_k > 1}}{2\sum^d_{i=1} \mu \cdot \frac{|\alpha|}{|\beta|} \cdot \frac{\beta_i}{\alpha_i} \cdot \mathbb{1}_{\beta_i > 1}} \\ &= \frac{\beta_k \cdot \mathbb{1}_{\beta_k > 1}/\alpha_k}{2\sum^d_{i=1} \beta_i \cdot \mathbb{1}_{\beta_i > 1}/\alpha_i}. \end{aligned} \quad (10)$$

These proportions are completely determined by the α and β domain characterization d-dimensional vectors.

Taking into account that if $\beta_k = 2$ for some k, then two tasks that are neighbors along axis k communicate across two surfaces, we obtain:

$$\hat{t}_{sd} = \sum^d_{i=1} m_i \cdot \nu^i_{s,d} \cdot (1 + \mathbb{1}_{\beta_i = 2}) \quad (11)$$

Cartesian placement.

The case where the tasks assigned to any fixed group form a Cartesian sub-domain is of interest because it allows keep-

ing as much of the communication local as possible. For this strategy, we defined the γ vector characterizing the mapping of tasks to groups. For a fixed γ, we have all the information necessary to compute the demand that the workload induces in the network as described by Eqs. (2) and (3).

Indeed, in the case of direct routing, we would need to estimate the sum of the individual demands induced by sources in a Dragonfly group i sending to destinations in another Dragonfly group j on the remote link connecting the groups. Similarly to how we assigned coordinate vectors to tasks, we can now assign coordinate vectors to groups of Cartesian sub-domains of tasks. A group g will be assigned a coordinate vector x^g in $\mathbb{N}^d$, where every coordinate x^g_k satisfies $0 \leq x^g_k < \lambda_k, \forall 1 \leq k \leq d$. Similarly to the neighborhood relationship for tasks defined by (7) we can define a neighborhood relationship for groups as

$$\tilde{\nu}^k_{i,j} = \begin{cases} 1 & \text{if } (\lambda_k + x^i_k - x^j_k) \bmod \lambda_k = 1 \\ & \text{and } \forall l \neq k, x^i_l = x^j_l \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

For two neighboring groups along direction k, the traffic exchanged across the common boundary is the aggregate traffic sent to one neighbor along dimension k by all the tasks forming that boundary. Given the shape of the task domain, there are exactly $(1+\mathbb{1}_{\lambda_k=2}) \cdot |\gamma|/\gamma_k$ tasks on the boundary. Eq. (13) shows the resulting direct routing demand.

$$\begin{aligned} \delta^{\text{R,direct}}_{ij} &= \Delta^{\text{R,direct}}_{ij} / B_{\text{P,eff}} \\ &= \sum_{s \in G_i} \sum_{d \in G_j} \hat{t}_{sd} \\ &= \sum_{k=1}^{d} (1 + \mathbb{1}_{\lambda_k=2}) \cdot |\gamma|/\gamma_k \cdot m_k \cdot \tilde{\nu}^k_{i,j} \end{aligned} \tag{13}$$

In the case of indirect routing, we start by expressing $\delta^{\text{R,indirect}}$ from Eq. (3) and (4).

$$\begin{aligned} \delta^{\text{R,indirect}}_{ij} =& \sum_{p \neq i,j} \sum_{s \in G_i} \sum_{d \in G_p} \hat{t}_{sd} \cdot P(i \xrightarrow{j} p) \\ &+ \sum_{p \neq i,j} \sum_{s \in G_p} \sum_{d \in G_j} \hat{t}_{sd} \cdot P(p \xrightarrow{i} j) \\ =& \frac{1}{ah-1} \cdot \left(\sum_{p \neq i,j} \sum_{s \in G_i} \sum_{d \in G_p} \hat{t}_{sd} \right. \\ &\left. + \sum_{p \neq i,j} \sum_{s \in G_p} \sum_{d \in G_j} \hat{t}_{sd} \right) \end{aligned} \tag{14}$$

The two triple sums each evaluate to the same value for reasons that are linked to the symmetry of the communication pattern. Indeed, the amount of traffic sent by tasks in a group i to tasks in all groups $p \neq i, j$ (first triple sum) is equal to the aggregate external nearest neighbor traffic of i minus the traffic that i sends to group j if i and j are neighbors along some dimension. Similarly, the amount of traffic sent by all groups $p \neq i, j$ to tasks in group j (second triple sum) is equal to the aggregate external nearest neighbor traffic received by j minus the traffic that i sends to group j if i and j are neighbors along some dimension. Due to the fact that from the point of view of any individual sub-domain of tasks mapped to a group, the communication pattern is symmetric (same messages received and sent along each dimension), the two triple sums are equal.

The aggregate nearest neighbor traffic sent (or received) by any group is equal to $2\sum_{k=1}^{d}(|\gamma|/\gamma_k \cdot m_k \cdot \mathbb{1}_{\lambda_k>1})$ where the indicator function only serves to take into account the corner case where the intra-group domain would be so large along dimension k that it would completely cover the entire application domain along that direction and thus eliminate any inter-group traffic along that dimension. Eq. (15) shows the resulting indirect routing demand.

$$\begin{aligned} \delta^{\text{R,indirect}}_{i,j} =& (4 \sum_{k=1}^{d} (|\gamma|/\gamma_k \cdot m_k \cdot \mathbb{1}_{\lambda_k>1}) \\ &- 2\delta^{\text{R,direct}}_{i,j})/(ah-1) \end{aligned} \tag{15}$$

To be able to now derive performance estimations (via Eq. (6)) we maximize the demand. For direct routing, using the fact that Eq. (12) implies that task domains mapped to two groups can only be neighbors in at most one direction and Eq. (13), we obtain the bound in Eq. (16)

$$\max_{i,j}(\delta^{\text{R,direct}}_{i,j}) = \max_{k}((\mathbb{1}_{\lambda_k>1} + \mathbb{1}_{\lambda_k=2}) \cdot |\gamma|/\gamma_k \cdot m_k) \tag{16}$$

Similarly, Eq. (17) shows the bound obtained when maximizing indirect routing induced demand

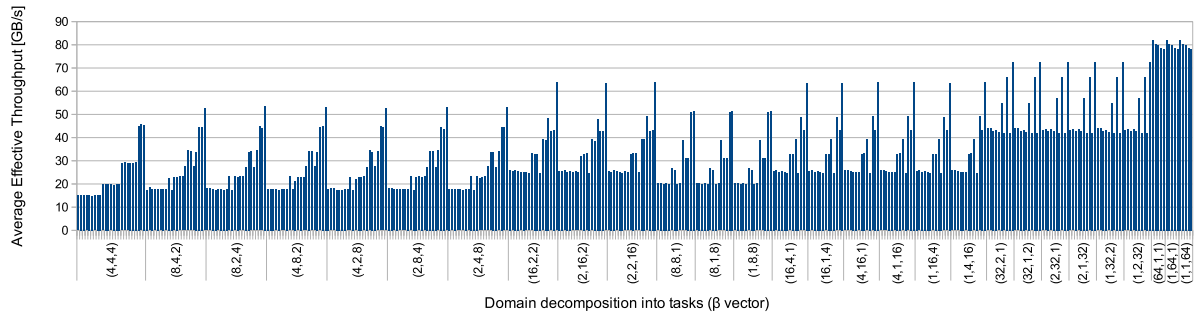
$$\max_{i,j}(\delta^{\text{R,indirect}}_{i,j}) = 4 \sum_{k=1}^{d} (|\gamma|/\gamma_k \cdot m_k \cdot \mathbb{1}_{\lambda_k>1})/(ah-1). \tag{17}$$

As expressed by Eq. (6), maximizing performance is equivalent to minimizing the maximum demand. The previous equations express the maximum demand when the choice of a γ vector defining the mapping of tasks to the system topology has already been made. The same equations can however also be used to reason about what the mapping vectors γ, β themselves should be such that performance is maximized. This can be achieved by shifting the maximization domain to include the domain of possible values for γ and β as well (in addition to the values of k).

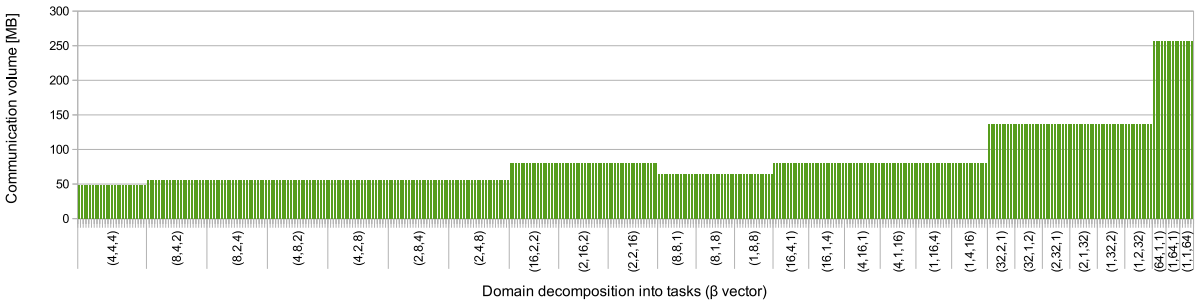
Random placement.

The second placement strategy we analyze is one where elements are assigned at random to nodes in the system under the constraints of having at most one element assigned to a given node and exactly 0 or $|\gamma|$ elements per group. This leads to there being exactly $|\lambda|$ groups hosting tasks. This strategy sacrifices communication locality in exchange for more uniform-like traffic that is amenable to direct routing.

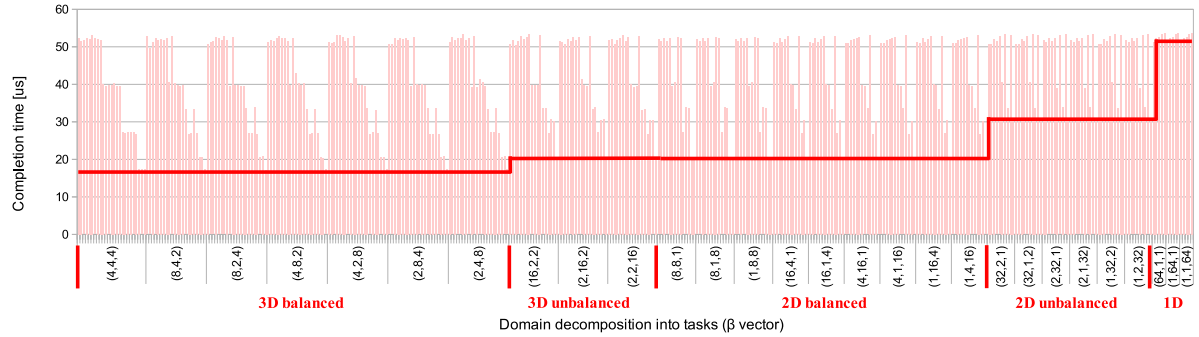
Given an arbitrarily chosen group to which tasks are assigned, and given a certain dimension k and direction in which the tasks exchange messages, there will be exactly γ messages that the tasks in the group need to exchange. Due to the placement strategy described above, each of these γ messages have an equal chance to be destined to tasks in each of the $|\lambda|$ groups, including the source group itself. In the case of direct routing, according to Eq. (5), what we are interested in is the expected maximum remote link demand. To estimate it, we note that the problem of assigning messages to destination groups is an instance of the balls-and-bins problem [10], where messages are the balls ($n_{\text{balls}} = |\gamma|$) and the groups are the bins ($n_{\text{bins}} = |\lambda|$) What we are interested in is the expected number of balls that the bin that



(a) Throughput (higher is better)



(b) Communication Volume



(c) Completion time (lower is better)

Figure 3: Comparison of the measured effective throughput, communication data volume, and completion time for a balanced dragonfly topology under indirect routing for a collection of all possible application domain decompositions into 64 tasks. The x axis of every subfigure shows the β vector defining the decomposition. For a fixed β vector, several task-to-node placement strategies (shown on the charts from denser to sparser left to right) are benchmarked to illustrate the variability of achievable performance. For figure c), the best completion time per domain decomposition is highlighted by means of horizontal lines. This best case performance is shown to be highly correlated with the decomposition type, and as such the same figure shows the clusters of decompositions that share similar characteristics.

has the most balls will have. As proven in [7, 17], for large balls-to-bins ratios, this is given by Eq. (18).

$$\frac{n_{\text{balls}}}{n_{\text{bins}}} \cdot \frac{\log n_{\text{bins}}}{\log \log n_{\text{bins}}} \tag{18}$$

This leads to the expected maximum demand induced by messages exchanged in the chosen direction of the chosen dimension k to be:

$$\frac{|\gamma|}{|\lambda|} \cdot \frac{\log |\lambda|}{\log \log |\lambda|} \cdot m_k. \tag{19}$$

Summing over all dimensions and both directions we obtain

$$\begin{aligned} \max_{\text{Rlinks}} (\delta^{\text{R,direct}}) &= 2 \sum_{k=1}^{d} \frac{|\gamma|}{|\lambda|} \cdot \frac{\log |\lambda|}{\log \log |\lambda|} \cdot m_k \\ &= \frac{|\gamma|}{|\lambda|} \cdot \frac{\log |\lambda|}{\log \log |\lambda|} \cdot 2 \sum_{k=1}^{d} m_k \\ &= \frac{|\gamma|}{|\lambda|} \cdot \frac{\log |\lambda|}{\log \log |\lambda|}. \end{aligned} \tag{20}$$

This applies when the ratio of messages to groups is large. Since this is not necessarily the case in several of the configurations we take into account, we can expect the accuracy of

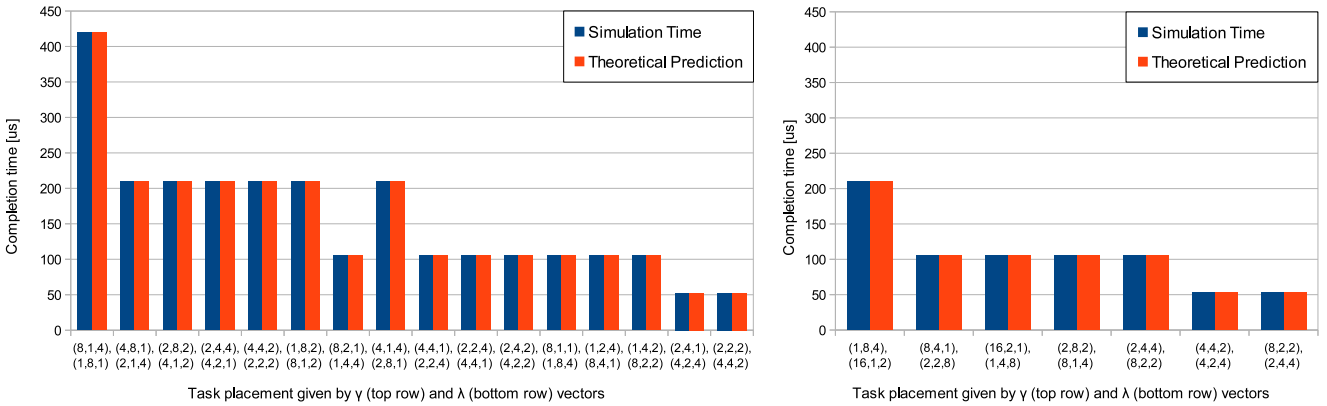


Figure 4: Comparison of the completion time (estimated and measured) for a balanced dragonfly topology under direct routing for a collection of all possible task-placements γ for a 3-dimensional $\beta = (8, 8, 4)$ nearest neighbor exchange of 256 tasks (Figure a)) and a $\beta = (16, 8, 8)$ nearest neighbor exchange of 1024 tasks (Figure b)). The x axis lists the task-placements from least-sparse (each group is fully occupied) to most-sparse (each group contains the minimum number of nodes that still allows for the workload to be scheduled on the Dragonfly system) and within each sparsity level, every possible γ placement whose $|\gamma|$ corresponds to the sparsity level. The first row of the x axis labels defines γ while the second row defines λ.

the random placement model to be somewhat poorer than that of the Cartesian placement model.

Random placement and random indirect routing are two solutions to similar issues arising in Dragonfly networks, each with its advantages and disadvantages. Using the two techniques in conjunction would yield very little benefit beyond the benefits already attainable by employing one or the other separately, and additionally incur the drawbacks of both strategies. Thus, there is little motivation for modeling performance for indirectly routed randomly placed workloads and therefore we will restrict our analysis of randomly placed nearest neighbor exchanges to the direct routing case.

3.5 Summary

In this Section we have introduced a formal performance model for nearest neighbor communication over Dragonfly networks, under i) Cartesian and ii) random task placement and using a) direct or b) *Valiant [Kim:2008]* indirect routing. The results of this analysis are the following.

For Cartesian placement and direct routing, we have shown (Eq. (16) and (5)) that effective injection bandwidth is limited by

$$B_{\mathrm{P,eff}} \leq \frac{B_R}{\max_{k,\gamma,\beta}((\mathbb{1}_{\lambda_k>1} + \mathbb{1}_{\lambda_k=2}) \cdot |\gamma|/\gamma_k \cdot m_k)}. \quad (21)$$

By substituting m_k using Eq. (10) we obtain the optimization criterion for this configuration: optimal performance is obtained by minimizing

$$\max_{k,\gamma,\beta}\left[(\mathbb{1}_{\lambda_k>1} + \mathbb{1}_{\lambda_k=2}) \cdot \frac{|\gamma|}{\gamma_k} \cdot \frac{\beta_k \cdot \mathbb{1}_{\beta_k>1}}{\alpha_k}\right]. \quad (22)$$

Strictly speaking this optimizes the performance bound, but we expect it to also optimize performance since the bounds should be tight given our assumption that other limitations on performance are removed.

For Cartesian placement and *Valiant [Kim:2008]* indirect routing, we have shown (Eq. (17) and (6)) that effective injection bandwidth is limited by

$$B_{\mathrm{P,eff}} \leq \frac{B_R}{\max_{\gamma,\beta}(4\sum_{k=1}^{d}(|\gamma|/\gamma_k \cdot m_k \cdot \mathbb{1}_{\lambda_k>1})/(ah-1))}. \quad (23)$$

By substituting m_k using Eq. (10) we obtain the optimization criterion for this configuration: optimal performance is obtained by minimizing

$$\max_{\gamma,\beta}\left[\sum_{k=1}^{d}\left(\mathbb{1}_{\lambda_k>1} \cdot \frac{|\gamma|}{\gamma_k} \cdot \frac{\beta_k \cdot \mathbb{1}_{\beta_k>1}}{\alpha_k}\right)\right]. \quad (24)$$

For random placement and direct routing, we have shown (Eq. (20) and (5)) that effective injection bandwidth is limited by

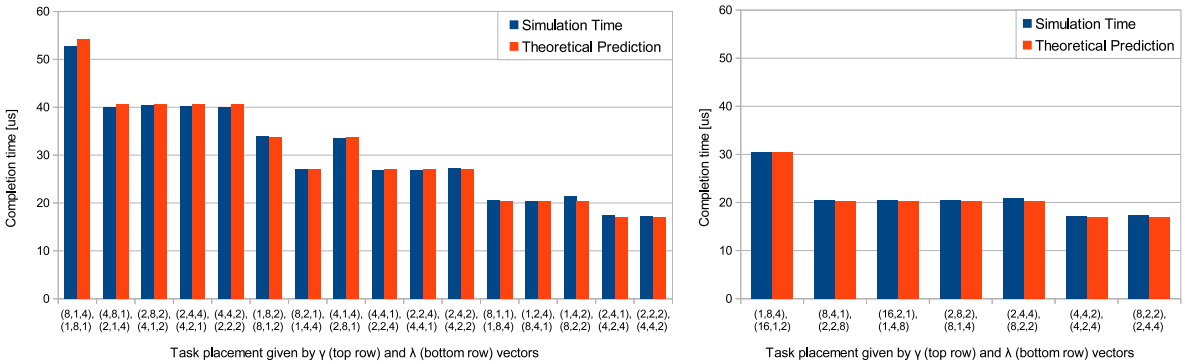
$$B_{\mathrm{P,eff}} \leq \frac{B_R}{\max_{\gamma,\beta}\left[\frac{|\gamma|}{|\lambda|} \cdot \frac{\log|\lambda|}{\log\log|\lambda|}\right]}. \quad (25)$$

Optimal performance is thus obtained by minimizing

$$\max_{\gamma,\beta}\left[\frac{|\gamma|}{|\lambda|} \cdot \frac{\log|\lambda|}{\log\log|\lambda|}.\right] \quad (26)$$

In all cases, given that the parameter space is not very large, minimization can be achieved by exhaustive exploration.

Thus, the analytical model we introduce has a two-fold use. First, in the context of one of the three routing and placement strategies described, it provides straightforward criteria allowing the selection of the most efficient assignment of domain elements to tasks (β) and assignment of tasks to network nodes (γ,λ). Second, due to its capability to estimate not only the circumstances in which performance is maximized but also actual expected performance, the model allows selecting the (routing, placement) strategy itself. Thus, overall, it allows for the identification of the complete configuration of a workload such that that workload completes in a minimum amount of time.



a) 256-Tasks NNE; β = (8,8,4); *Valiant [Kim:2008]* routing

b) 1024-Tasks NNE; β = (16,8,8), *Valiant [Kim:2008]* routing

Figure 5: Comparison of the completion time (estimated and measured) for a balanced dragonfly topology under *Valiant [Kim:2008]* indirect routing for a collection of all possible task-placements γ for a 3-dimensional $\beta = (8, 8, 4)$ nearest neighbor exchange of 256 tasks (Figure a)) and a $\beta = (16, 8, 8)$ nearest neighbor exchange of 1024 tasks (Figure b)). The x axis lists the task-placements from least-sparse (each group is fully occupied) to most-sparse (each group contains the minimum number of nodes that still allows for the workload to be scheduled on the Dragonfly system) and within each sparsity level, every possible γ placement whose $|\gamma|$ corresponds to the sparsity level. The first row of the x axis labels defines γ while the second row defines λ.

4. SIMULATION RESULTS

The results that we present in this section were obtained by means of a simulation framework that is able to accurately model custom networks (including Dragonflies) at a flit level [15]. The simulator is characterized by a high level of customization and modularity, allowing the configuration of the desired model in detail. Given that the parameter space to explore was already very large, we have chosen a fixed representative system scale of 1056 end nodes. Specifically, we have chosen a system interconnected by a balanced $DF(4, 8, 4)$ Dragonfly network, where groups are made up of $a = 8$ switches interconnected in a fully connected mesh and each switch has $p = 4$ nodes attached and $h = 4$ ports towards other groups. The bandwidth of every link was chosen to be 40 Gbit/s. The routing approaches used were direct routing and *Valiant [Kim:2008]*.

The traffic pattern is that of a single, 1D, 2D or 3D, nearest neighbor exchange as described in Section 3.2. The application domain was considered to be made up of 2^{27} elements shaped according to the vector $\alpha = (512, 512, 512)$. The per-element per-neighbor message size was chosen to be 1 KB. This means that if a task shares with a neighboring task a surface made up of E elements, the total amount of data sent by the task to that neighbor will be E KB.

4.1 Optimal application domain to task mapping

We will start by exploring the trade-offs that are associated with the selection of the β vector, expressing the mapping of application domain elements to computational tasks. The selection is done along two dimensions:

1. The total number of tasks.

2. The shape of the element sub-domain assigned to a task. By shape we refer to whether the sub-domain is one, two or three-dimensional as well as to the ratios between the elements of β.

As Eq. (8) shows, the larger the number of tasks the smaller the communication footprint per task will be. Also, regarding the sub-domain shape, similar dimensions of the task sub-domain lead to similar contact surfaces in each dimension, and therefore to more balanced sizes of the messages exchanged along each axis. Given the broad range in which the workload size can vary when choosing different element to task mappings, we will consider for this subsection both the effective completion time of the communication pattern, which is the most relevant performance indicator, and the effective throughput, which is often used as a performance indicator in practice.

We will start by choosing a fixed number of tasks, $|\beta| = 64$, and analyzing all possible domain decompositions. For a given domain decomposition, we will include multiple configurations with respect to parameters such as task-to-node mapping strategy, but we will not analyze them in detail in this subsection. What we will focus on is what performance can be obtained for a fixed domain decomposition (β vector) in the best case. The measured performance is illustrated in Fig. 3; the measured effective throughput is shown in Figure 3a the total volume of exchanged data is shown in Figure 3b and the effective completion time is shown in Figure 3c.

Several conclusions can be drawn from this experiment. First, we notice that there is a clear correlation between the best achievable completion time and the domain decomposition (Figure 3c). Indeed, decompositions that preserve both a higher number of dimensions and a symmetric distribution of tasks across dimensions perform consistently better than decompositions that are at the other end of the spectrum. The completion time of the best decomposition is more than twice as small as the completion time of the worst decomposition, with several performance levels for intermediate decompositions.

Second, the most widely used metric to measure network performance is the throughput that the network can sustain. However, if we were to have based our performance analysis

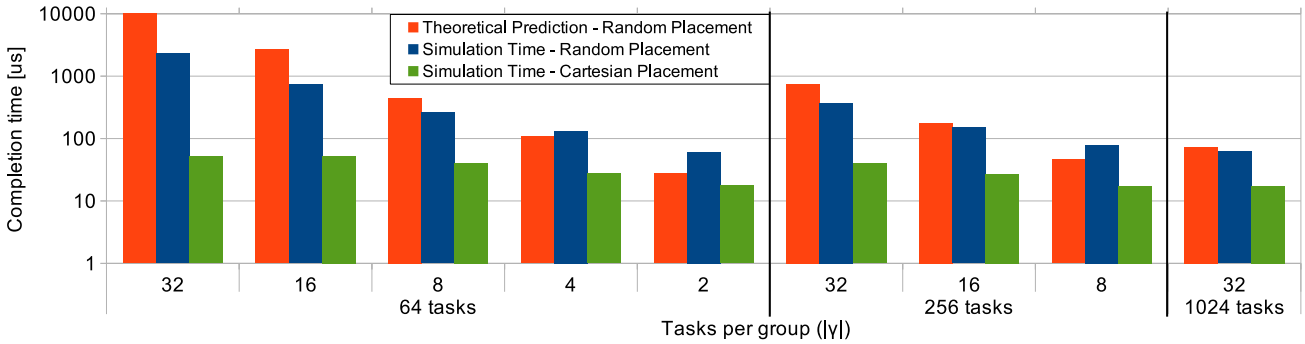


Figure 6: Comparison of the completion time (estimated and measured) for a balanced dragonfly topology under direct routing and random task placement for three 3-dimensional nearest neighbor exchanges ($\beta \in \{(4,4,4),(8,8,4),(16,8,8)\}$). The x axis lists the 3 exchanges from left to right (identifying them by the total number of tasks $|\beta|$), and within each exchange lists several levels of sparsity from least-sparse (each group is fully occupied) to most-sparse (each group contains the minimum number of nodes that still allows for the workload to be scheduled on the Dragonfly system). For each of these configurations, the completion time of the same workload, but this time Cartesian-mapped and indirectly routed, is also shown to allow comparison between the two strategies. The y axis has a logarithmic scale.

on this measure, our conclusions would have been the exact opposite. Indeed, as shown in Figure 3a, judging strictly by maximum achievable throughput, the best decomposition is, by a significant margin, exactly the one that actually takes the longest to complete. Using throughput to measure performance is thus questionable in this context, where the high throughput is actually a direct consequence of the decomposition requiring significantly larger volumes of exchanged data (Figure 3b) to account for the significantly larger contact surfaces between communicating tasks.

Finally, although the correlation between best case completion time and domain decomposition (β choice) is clear, the variability in the performance achieved for a fixed decomposition is extremely high. In fact, it is much higher than the variability across decompositions. This makes the choice of the intra-group task-to-node mapping (γ choice) of the utmost importance.

4.2 Optimal task to network topology mapping and validation of theoretical estimates

Across tested decompositions of the application domain, the best performance was obtained for a domain that is as close to a cube as possible (a β vector with elements as close to equal as possible). Furthermore, from the point of view of the variability induced by the intra-group placement (γ-choice), the different domain decompositions exhibited a similar behavior. Thus, in this subsection we will set the decomposition of the application domain to the decomposition closest to a cube and benchmark all possible intra-group placements (γ choices) under the fixed β vector.

We will consider, instead of the 64 task decomposition, a 256 and a 1024 task decomposition of the same domain to be able to examine the scale dependence of the results we obtain. For the former, we will consider $\beta = (8,8,4)$ while for the latter we will consider $\beta = (16,8,8)$.

For the vector γ, which completely defines a Cartesian task-to-node mapping, we will limit our analysis to values for which the individual elements divide the corresponding β elements. For each choice, we measure the completion time of the workload and compare it against the theoretical predictions of the model introduced in Section 3. In addition to validating our theoretical framework, by analyzing all possible γ values, we are able to study the impact of the shape of the domains chosen to be mapped to individual groups, as well as the impact of workload sparsity. The measured and predicted performance is shown in Figure 4 for direct routing and in Figure 5 for *Valiant [Kim:2008]* indirect routing.

In order to be able to compare the predicted effective bandwidth induced by remote link bottlenecks to measured performance, for this experiment we consider a high enough bandwidth for the local links, such that the bottleneck does not shift towards them, especially for the sparser mappings.

The main conclusion we can draw from the simulation results is that the theoretical framework is able to accurately capture the behavior of the system. This is particularly true for Cartesian task placements under both direct (Figure 4) and indirect (Figure 5) routing, where the predictions of the model are practically indistinguishable from the measurements. An immediate consequence of this fact is that the model can be used in a standalone fashion not only to select the best configuration to run a particular workload but can also to produce accurate estimates of the absolute performance of arbitrary configurations.

For random task placement (Figure 6), the predictions follow well the evolution of the performance across tested configurations, but the model experiences nonetheless fairly large deviations compared to the absolute measured values. This is due to the fact that the statistical analysis the model is based on in this case relies on the assumption of exchanging a large number of messages relative to the number of occupied Dragonfly groups $|\lambda|$ and this is not always the case in all tested configurations. That being said, the performance trends are captured faithfully and the decision of which configuration suits a particular workload best can still be taken based solely on the model.

Finally, Figure 6 also shows that, for all tested configurations, random task placement with direct routing is consistently outperformed by Cartesian task placement with indirect routing. Indeed, on the configurations that we benchmarked, the former took between 3 and 15 times more time to complete for a fixed β and γ configuration.

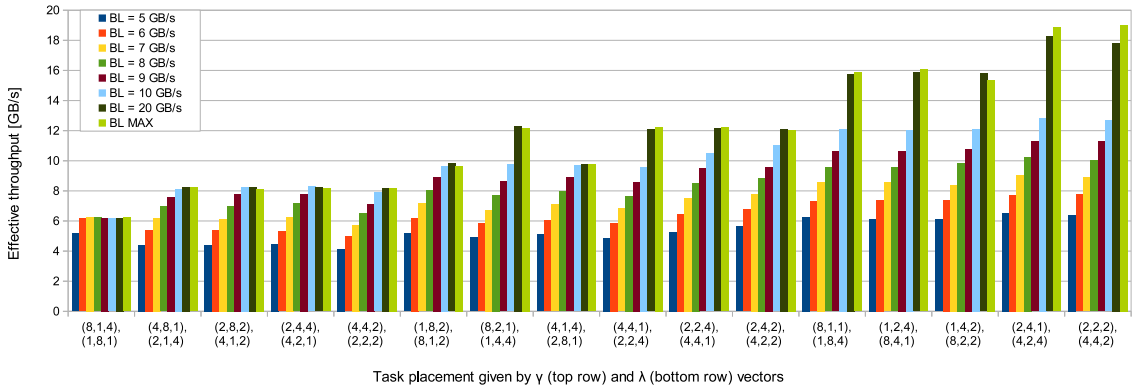


Figure 7: Effective throughput upper bound for nearest neighbor exchange among $|\beta| = 256$ tasks mapped on a 3-dimensional domain of size $\beta = (8, 8, 4)$ for increasingly higher local link bandwidths. The routing used is indirect routing and the placement is Cartesian: the first line of the x axis shows the vector γ while the second line shows the vector λ. By increasing the local link bandwidth, bottlenecks in the local links become less important, up to a point where a balance is reached between the limitations caused by remote links and limitations caused by local links. Further increasing the L link bandwidth will yield practically no further increase of the effective throughput.

4.3 Dragonfly balance for nearest neighbor exchanges

To conclude this section, we revisit the notion of Dragonfly balance in the context of nearest neighbor exchanges. A Dragonfly system is considered balanced if the limitation on effective injection bandwidth imposed by the three system bandwidth parameters B_P, B_L and B_R is the same under uniform random all-to-all traffic. Ensuring this property translates into a set of constraints on what the bandwidth per each type of link should be.

For traffic patterns that do not have a uniform random all-to-all structure and that exhibit poor performance in balanced Dragonfly systems under direct routing, it is generally assumed that a uniform distribution of load in the network can be achieved via indirect routing. It is further assumed that for this new load distribution, the balance of the Dragonfly is similar to that of indirectly routed uniform traffic.

In the case of the nearest neighbor exchange, we study the balance between the local and remote link bandwidth. We look at two extreme cases, one where the bandwidth of the local links is set to a very high value, such that the throughput limitation is the effect of solely a remote link bottleneck, and the other where the dragonfly is balanced for uniform traffic (at 5 GB/s with direct routing). Figure 7 shows that in the latter case, the bottleneck has clearly shifted towards the local links. As such, we benchmarked several L bandwidth values to identify what the tipping point is in terms of bandwidth, and implicitly what the balance of the Dragonfly is when considering nearest neighbor traffic.

We observe that, while for the mapping strategies that exhibit low sparsity, the balance of the system is very close to the uniform traffic balance, as we move toward sparser and more efficient mappings, the local bandwidth required to balance the traffic pattern increases as well, becoming up to a factor of 4 larger in the sparsest case.

4.4 Guidelines for application-network joint configuration and design

In Section 3.5 we have summarized the analytical performance model introduced in this work, which has the ability to i) determine the configuration options that would enable a Cartesian nearest neighbor exchange to achieve optimum performance on a Dragonfly network and ii) estimate that level of optimum performance. In the current section, we have shown this model to be accurate in both aspects.

Thus, this model can be used in practice as follows. For each of the three routing and placement strategies presented, i.e., Cartesian placement with direct routing, Cartesian placement with indirect routing, and random placement with direct routing, one would use the model (namely Eq. (22), Eq. (24) and Eq. (26) respectively) to determine the configuration of the workload parameters β and γ. Then, one would use these parameters to determine for each (routing,placement) strategy (via Eq. (21), Eq. (23) and Eq. (25) respectively) the expected completion time for the workload, and select the strategy with the lowest one.

Concerning the optimization of the network design, the previous subsection has shown that the generally accepted guidelines for designing balanced, near optimal performance Dragonfly networks [13] (derived for uniform traffic), do not typically hold in the context of Cartesian nearest neighbor exchanges. Indeed, the intra-group aggregate bandwidth should be over-provisioned (relative to the balanced case), for optimal performance by as much as a factor of 4.

5. CONCLUSIONS

In this work we analyze communication workload performance in systems where the interconnect fabric is a Dragonfly network. We introduce a theoretical framework that is able to identify the bottlenecks that appear in the network under arbitrary workloads (specified via their traffic demand matrix), assuming either direct or indirect routing approaches, as well as to determine how those bottlenecks impact the effective injection throughput of the nodes.

With the help of this framework, we analyze Cartesian multi-dimensional nearest neighbor exchanges, a communication pattern that is prevalent in multiple high performance computing applications. Using the resulting theoretical estimates, as well as a wide array of simulations results that validated and augmented the analytical model, we quantify the performance of different nearest neighbor workloads coupled to a variety of mapping strategies. We are able to

pinpoint mapping-related performance trends such as the advantages of workload fragmentation and of assigning convex application sub-domains with low surface-to-volume ratios to Dragonfly groups. This enables us to *co-design* application decomposition, routing, and mapping in order to achieve *optimal* overall performance.

Finally, we were able to unveil common misconceptions regarding Dragonfly network design and evaluation. We showed that optimizing for throughput and not workload completion time is often misleading. Furthermore, the notion of system balance that is often cited as a Dragonfly design parameter is not directly applicable to all workloads.

We present a network-application co-design effort between one of the most promising topologies from a scalability and cost point of view, the Dragonfly, and one of the most widely used communication patterns in scientific applications, the Cartesian nearest neighbor exchange. Our theoretical models capture important application and network characteristics and can be solved optimally. We showed substantial performance improvements of up to 10x and expect that our model will soon become a standard technique. For example, a batch system could inform a self-optimizing application about the task mapping and a solver could automatically determine the best decomposition and routing strategy.

6. ACKNOWLEDGMENTS

This work was supported and partially funded by Lawrence Livermore National Laboratory, on behalf of the US Department of Energy, under Lawrence Livermore National Laboratory subcontract number B601996.

7. REFERENCES

[1] M. Alvanos, G. Tanase, M. Farreras, E. Tiotto, J. N. Amaral, and X. Martorell. Improving performance of all-to-all communication through loop scheduling in PGAS environments. In *Proc. of the* 27^{th} *International Conference on Supercomputing*, ICS '13, pages 457–458, New York, NY, USA, 2013. ACM.

[2] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony. The PERCS High-Performance Interconnect. In *Proc. of the* 18^{th} *IEEE Symposium on High Performance Interconnects*, HOTI '10, pages 75–82, Washington, DC, USA, 2010. IEEE Computer Society.

[3] A. Bhatele, N. Jain, W. D. Gropp, and L. V. Kale. Avoiding hot-spots on two-level direct networks. In *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 76:1–76:11, New York, NY, USA, 2011. ACM.

[4] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[5] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard. Cray Cascade: a scalable HPC system based on a Dragonfly network. In *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 103:1–103:9, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[6] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, G. Rodriguez, J. Labarta, and C. Minkenberg. On-the-fly adaptive routing in high-radix hierarchical networks. In *Proc. of the* 41^{st} *International Conference on Parallel Processing (ICPP)*, pages 279–288, 2012.

[7] G. H. Gonnet. Expected length of the longest probe sequence in hash code searching. *J. of the ACM (JACM)*, 28(2):289–304, 1981.

[8] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus framework and toolkit: Design and applications, 2003.

[9] T. Hoefler, R. Rabenseifner, H. Ritzdorf, B. R. de Supinski, R. Thakur, and J. L. Traeff. The Scalable Process Topology Interface of MPI 2.2. *Concurrency and Computation: Practice and Experience*, 23(4):293–310, Aug. 2010.

[10] N. L. Johnson and S. Kotz. *Urn models and their application: an approach to modern discrete probability theory*. Wiley New York, 1977.

[11] A. Jokanovic, B. Prisacari, G. Rodriguez, and C. Minkenberg. Randomizing task placement does not randomize traffic (enough). In *Proc. of the* 7^{th} *Interconnection Network Architecture: On-Chip, Multi-Chip*, IMA-OCMC '13, pages 9–12, New York, NY, USA, 2013. ACM.

[12] S. Kamil, J. Shalf, L. Oliker, and D. Skinner. Understanding ultra-scale application communication requirements. *Proc. of the Workload Characterization Symposium*, pages 178–187, Oct. 2005.

[13] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-driven, highly-scalable dragonfly topology. *SIGARCH Comput. Archit. News*, 36(3):77–88, June 2008.

[14] Z. Lin, S. Ethier, T. S. Hahm, and W. M. Tang. Size scaling of turbulent transport in magnetically confined plasmas. *Phys. Rev. Lett.*, 88(19):195004–, 2002.

[15] C. Minkenberg, W. Denzel, G. Rodriguez, and R. Birke. End-to-end modeling and simulation of high-performance computing systems. *Springer Proc. in Physics: Use Cases of Discrete Event Simulation: Appliance and Research*, page 201, 2012.

[16] B. Prisacari, G. Rodriguez, M. Garcia, E. Vallejo, R. Beivide, and C. Minkenberg. Performance implications of remote-only load balancing under adversarial traffic in dragonflies. In *Proc. of the* 8^{th} *International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*, INA-OCMC '14, pages 5:1–5:4, New York, NY, USA, 2014. ACM.

[17] M. Raab and A. Steger. Balls into bins - a simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science*, pages 159–170. Springer, 1998.

[18] L. G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 11(2):350–361, 1982.

[19] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick. Lattice Boltzmann simulation optimization on leading multicore platforms. In *Proc. of the International Conference on Parallel and Distributed Computing Systems (IPDPS)*, 2008.