DISS. ETH NO. 26682

WORKLOAD AND INTERCONNECTION NETWORK AWARE PERFORMANCE OPTIMIZATION

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

BOGDAN PRISACARI

Ing. Info. Dipl. EPFL

born on 09.03.1984

citizen of Romania

accepted on the recommendation of

Prof. Dr. Torsten Hoefler, examiner
Prof. Dr. Tor Skeie, co-examiner
Dr. Cyriel Minkenberg, co-examiner
Mitch Gusat, co-examiner

2020

A dissertation submitted to
ETH Zurich
for the degree of Doctor of Sciences

DISS. ETH No. 26682
Prof. Dr. Torsten Hoefler, examiner
Prof. Dr. Tor Skeie, co-examiner
Dr. Cyriel Minkenberg, co-examiner
Mitch Gusat, co-examiner

Examination date: March 6th, 2020.

Acknowledgements

Pursuing a doctoral degree has been a great opportunity for me to learn how to conduct state of the art research, to write high quality scientific publications, to collaborate with researchers across the world and in general to work with and meet some extraordinary intelligent people. Here I would like to express my appreciation to all those who helped me during this time.

First, I would like to express my sincere gratitude to Prof. Dr. Torsten Hoefler for giving me the opportunity to pursue my doctoral studies under his supervision. I highly appreciate his guidance, the discussions we have had over the years, the insights he has brought to my work, and most of all his unwaivering support.

I am also deeply grateful to Dr. Cyriel Minkenberg for his mentorship, for helping me grow professionally and develop my skills (be they building efficient large scale HPC simulation software, participanting in and driving research projects, or writing high quality publications and patents) and for his permanent encouragement and support.

Moreover, I would like to sincerely thank Prof. Dr. Tor Skeie and Mitch Gusat for accepting to be co-examiners for my doctoral defense and for investing significant time in reviewing my work and providing feedback.

I would also like to extend a special thanks to German Rodriguez who has been a constant source of support, productive research discussions and most of all for being a great friend during my time at IBM.

I also am very grateful to a whole group of former colleagues at IBM Research Zurich and IBM Research Yorktown, for the collaborations we have had over the years, the inspiring research-related and non-work-related discussions, and in general for providing a great work and research environment: Dr. Martin Schmatz, Georgios Kathareios, Dr. Wolfgang Denzel,

Acknowledgements

Philip Heidelberger, Yutaka Sugawara, Craig Stunkel, Dong Chen, Dr. Marina Garcia, Dr. Ana Jokanovic, Prof. Dr. Ramon Beivide, Maria Soimu, Dr. Maria Gabrani, Ronald Luijten, Dr. Celestine Duenner, Dr. Robert Birke, Dr. Daniel Crisan, Dr. Milos Stanisavljevic, Dr. Florian Auernhammer, Dr. Patricia Sagmeister, Dr. Evangelos Eleftheriou, Dr. Robert Haas, Charlotte Bolliger, Anne-Marie Cromack.

I would also like to thank my colleagues in the Scalable Parallel Computing Lab at ETH Zurich: Maciej Besta, Timo Schneider, Tobias Grosser, Tobias Gysi, Salvatore Di Girolamo, Grzegorz Kwasniewski.

I would like to thank my parents Vasile and Daria, my brother Ionut and his wife Andreea and my sister Corina for their moral support and patience, as well as my friends Laurentiu and Codrin for the great moments we have had together in Zurich these past years.

Most importantly, I am extremely grateful to my wife Andreea, for her love and permanent support during this entire time. Without her constant encouragement and optimism, this work would not have come to completion.

Zurich, 10.02.2020

Abstract

High-Performance Computing and datacenter systems are continuously growing in scale as technology evolves and increasingly larger problems are tackled. Looking back at the most powerful supercomputers in the world in the past years, computational power has steadily increased from crossing for the first time the 1 Petaflop threshold in 2008 to 200 Petaflops today. Often times, these supercomputers are used to run specific complex applications, at huge scale (with sometimes a single application running across the entire system), and correspondingly at a high cost both upfront (when building the system) and for the duration of the execution of the application. This, coupled with the fact that the variety of configurations (both from the hardware and application point of view) that are of practical importance is relatively limited, leads to there being an opportunity for system-workload co-optimization.

The focus of this thesis is in particular on the communication aspect, both from the point of view of the system architecture (e.g. topological configuration of compute nodes and switches, routing algorithms) and from that of the applications themselves (e.g. data exchange patterns between the individual processes, application deployment patterns). In this context, system-workload co-optimization translates into introducing adaptations at the different levels in the communication stack, adaptations that improve performance based on knowledge/assumptions of the characteristics of the other levels. Concretely, we investigate in this thesis to what extent improvements can be attained by:

- adapting the (communication behavior of the) application based on the topological and routing configuration of the underlying system;
- adapting the allocation of application processes to machines in the system based on the application itself as well as the topological and routing configuration of the underlying system;
- adapting the routing strategy depending on the application being run as well as the specifics of the system topology;
- adapting the topology itself to cater to specific applications.

For each of these, the optimization goal can be higher data throughput, faster overall application completion time, or even lower system cost (fixed cost (switches and links) and/or running cost (energy)).

Focusing on a combination of, on the one hand, architectures that are widely deployed in existing systems (Fat Trees and Dragonflies in particular) as well as more recent cutting-edge topologies (diameter two direct and indirect networks) and, on the other hand, communication patterns prevalent in High-Performance Computing (all to all exchanges, neareast neighbor exchanges, and permutation-like adversarial traffic), we propose several novel ways to improve performance by carefully orchestrating the data exchange across the stack.

To support the improvements presented in this thesis, we provide both formal proofs of the expected performance guarantees as well as state-of-the-art simulation results, each confirming that significant benefit can be gained from taking advantage of knowledge about the underlying system when configuring the application level and vice-versa.

Résumé

Les systèmes de calcul de Haute Performance et les centres de données continuent d'augmenter d'echelle, au fur et à mesure que la technologie évolue et que des problèmes de plus en plus complexes sont résolus. En regardant les superordinateurs les plus puissants du monde ces dernières années, leur puissance de calcul a régulièrement augmenté, passant du franchissement pour la première fois du seuil de 1 Petaflop en 2008 à 200 Petaflops aujourd'hui. Souvent, ces superordinateurs sont utilisés pour exécuter des logiciels complexes spécifiques, à grande échelle (avec parfois un seul type de logiciel exécuté sur l'ensemble du système), et en conséquence à un coût élevé à la fois lors de la construction du système et pendant l'exécution du logiciel. Ceci, couplé avec le fait que la variété des configurations (à la fois du point de vue matériel et logiciel) qui sont d'une importance pratique est relativement limitée, conduit à une opportunité de co-optimisation.

Cette thèse se concentre en particulier sur l'aspect communication, tant du point de vue de l'architecture du système (ex: configuration topologique des nœuds et commutateurs de reseau, algorithmes de routage) que de celui des logiciels eux-mêmes (ex: schémas d'échange de données entre les processus individuels, déploiement des processus du logiciel dans le système). Dans ce contexte, la co-optimisation se traduit par l'introduction d'adaptations aux différents niveaux de la pile de communication, adaptations qui améliorent les performances en fonction des specificitées des caractéristiques des autres niveaux. Concrètement, nous étudions dans cette thèse dans quelle mesure des améliorations peuvent être obtenues par:

- adapter (le comportement de communication du) logiciel en fonction de la configuration topologique et de routage du système sous-jacent;
- adapter l'allocation des processus du logiciel aux machines du système en fonction du logiciel soi-même ainsi que de la configuration topologique et de routage du système sous-jacent;
- adapter la stratégie de routage en fonction du logiciel en cours d'exécution ainsi que des spécificités de la topologie du système;
- adapter la topologie elle-même pour répondre à des logiciels spécifiques.

Pour chacun d'eux, l'objectif d'optimisation peut être un débit de données plus élevé, un temps d'exécution global des logiciels plus petit, ou même un coût du système plus faible (coût fixe (commutateurs et connecteurs) et / ou coût de fonctionnement (energie)).

En se concentrant sur une combinaison, d'une part, d'architectures largement déployées dans les systèmes existants (Fat Trees et Dragonfly en particulier) ainsi que des topologies de pointe plus récentes (réseaux directs et indirects à diamètre deux) et, d'autre part, de modèles de communication frequents dans le calcul haute performance (échanges de tous vers tous, échanges entre voisins les plus proches, ainsi que communication de type permutation), nous proposons plusieurs nouvelles façons d'améliorer les performances en orchestrant soigneusement l'échange de données à travers la pile.

Pour soutenir les améliorations présentées dans cette thèse, nous fournissons à la fois des preuves formelles des ameliorations de performance anticipées ainsi que des résultats de simulation de pointe, chacun confirmant qu'un avantage significatif peut être obtenu en tirant parti des connaissances sur le système sous-jacent lorsqu'on configure les logiciels et vice-versa.

Contents

1 Introduction

Seeking to build on the one hand high performance systems and on the other hand applications that are efficient in their utilization of those systems has always been a priority for hardware and software designers. At first, *scaling up* was sufficient. Single hardware units improved at a steady state in terms of higher processor frequency, higher IO and memory bandwidth and lower latencies while on the application side developers could afford to remain for the most part oblivious to the underlying system and still ride the performance increase wave.

Soon however, physical limits were hit that basically removed the possibility for further significant increase in performance on a single machine. The next approach was *scaling out*, using multiple processing / memory / IO units connected by some form of on-chip or off-chip interconnection network and leveraging application parallelism to increase efficiency. Although now software development had to change significantly to enable the shift from the single threaded to the multi-threaded model, it still remained mostly oblivious to the details of the underlying hardware (such as the exact topological structure of the network of nodes that host the application).

Nonetheless, for most standard applications, this approach proved to offer sufficient computational resources. However, certain classes of workloads needed even more. Among these, some of the most important ones are: high performance computing workloads, datacenter driven workloads, high data throughput workloads (such as particle accelerator and radio astronomy enablers) and data analytics workloads. For such applications, the need in computational power is in the *exa* range. The challenges in achieving such a level of performance are many and we will not address them in detail here. However, it becomes clearer and clearer that, in order to make such systems possible and feasible it is essential to take full advantage of any opportunities to get the most out of a certain application given a certain system hosting it. *Scaling in* seeks to remove obliviousness on both the hardware and the software side and, by jointly optimizing the workload and the system enabling it, achieve optimum efficiency.

In the context of high performance computing, where there are a relatively limited number of

workload patterns of interest and a very limited number of core system designs, all with a high degree of regularity, this idea seems all the more feasible.

The main focus of this thesis is to investigate ways of achieving increased performance on the one hand by assuming, a detailed knowledge of the system, and specifically, of the interconnection network, being available to the application and on the other hand by designing systems (mainly the interconnection aspect) tuned for specific applications, in that they are able to enable high performance of those application at improved cost.

This is a complex optimization space, spanning multiple dimensions:

- the topological layout of the interconnection network
- the routing algorithm used to move messages through the network
- the task placement approach used to assign processes to host nodes
- higher layer (such as library layer) aspects such as MPI collective operations optimization

Within this space, we start by first focusing on the Fat Tree topology, common in supercomputers today. Ideal fat tree networks have been introduced by Leiserson [60] and exhibit several good properties: deadlock freeness, full bisection bandwidth, contention free routing of any permutation traffic. They do however also require switches with a number of ports that is exponential in the number of tree levels, making it non scalable. Generalizations of the fat tree emerged however (k-ary n-trees [79, 80] and the even more general extended generalized fat trees or XGFTs [75]) that no longer have switch complexity issues at the expense of requiring more complex routing algorithms and are still problematic to scale. In support of our thesis that cost improvements can be achieved by tuning the workload in a harware-aware way, we show in **Chapter 2** how all-to-all communication can be performed at full speed without full bisection bandwidth by carefully orchestrating the data exchanges between processes. Further, in **Chapter 3**, we move from the workload axis to the routing axis, and introduce an algorithm for producing an optimal routing strategy that is tailored to both the workload and the topology parameters.

Next, in **Chapter 4** we continue focusing on the all-to-all communication, but extend our analysis to hierarchical topologies in general, meaning Dragonfly networks as well in addition to Fat Trees. Dragonflies have been introduced by Kim and Dally [53,54]. Their main advantage is the fact that they have a small constant diameter (3), leading to low latency. Their most notable uses are in the IBM PERCS system [10] and in the Cray Cascade [27]. In this chapter, we show once more that adapting the workload to suit the topology can lead to important performance improvements.

Fixing now the Dragonfly topology, but switching to a different class of workloads, nearest neighbor exchanges, we proceed to show in **Chapter 5** the extent of performance improve-

ments that can be achieved by tuning the task placement (axis 3) and routing strategy (axis 2) to suit the workload and topology parameters.

Chapter 6 completes our analysis of the Dragonfly, by now focusing on uniform and adversarial traffic patterns (bit complement, bit reversal), providing tight theoretical and simulation-backed performance estimates for each available routing algorithm, and implicitly providing guidelines as to how the system should be configured for maximum performance.

Finally, in **Chapter 7** we consider the more recent diameter two topologies: Slim Fly, Multi-Layer Full-Mesh, and Two-Level Orthogonal Fat-Tree and show the cost benefits that each can achieve as well as how to tune the routing algorithms for each for maximum performance.

2 Bandwidth-optimal All-to-all Exchanges in Fat Tree Networks

The personalized all-to-all collective exchange is one of the most challenging communication patterns in HPC applications in terms of performance and scalability. In the context of the fat tree family of interconnection networks, widely used in current HPC systems and datacenters, we show that there is potential for optimizing this traffic pattern by deriving a tight theoretical lower bound for the bandwidth needed in the network to support such communication in a non-contending way. Current state of the art methods require up to twice as much bisection bandwidth as this theoretical minimum. We propose a set of optimized exchanges that use exactly the minimum amount of resources and exhibit close to ideal performance. This enables cost-effective networks, i.e., with as little as half the bisection bandwidth required by current state of the art methods, to exhibit quasi optimal performance under all-to-all traffic. In addition to supporting our claims by mathematical proofs, we include simulation results that confirm their correctness in practical system configurations.

2.1 Motivation

Today's parallel computations are often arranged in a bulk synchronous programming (BSP) model [107] in which communication and computation steps alternate. For example, in the Message Passing Model as implemented by MPI [73], one can realize communication steps by sending messages between processes. Several common communication patterns, such as broadcast, are available as collective communications in MPI. This high-level specification enables a clean separation of concerns between the algorithm, that requires certain semantics, and the network architecture, that implements communication. This enables performance-portability of programs to a wide variety of different architectures and systems. Indeed, a

This chapter is based on the article [86] PRISACARI, B., RODRÍGUEZ, G., MINKENBERG, C., AND HOEFLER, T. Bandwidth-optimal all-to-all exchanges in fat tree networks. In *International Conference on Supercomputing, ICS'13, Eugene, OR, USA - June 10 - 14, 2013* (2013), ACM, pp. 139–148. https://doi.org/10.1145/2464996.2465434

collective programming model has enormous benefits for programmability as well as performance [36] and is thus also adopted by many other parallel programming environments [7,67].

Of those collective communications, all-to-all is certainly the most demanding operation for the network because it needs to move $\Omega(p)$ data between p processes; it is thus also often considered the least scalable of MPI's collective operations [13]. Yet, there are many applications that require this communication pattern: Spectral codes, such as Direct Numerical Simulation [71] perform a 3D Fast Fourier Transform, utilizing large all-to-all communications for transposing the y direction over the network together with shared-memory on-node transforms of the x direction [28]. Even at the other end of the spectrum of scientific applications, data-driven parallel graph computations, such as betweenness centrality, essentially perform full all-to-all exchanges over the network [108].

All-to-all has thus been the subject of multiple research efforts optimizing it for different theoretical network models, such as LogP [81] or Postal [20], or technologies, such as InfiniBand [64] or Myrinet [112]. Our approach, however, is mostly oblivious to the network technology and current theoretical network models because we optimize for a certain network *topology*.

One important topology today is the family of fat trees [60], which form the base of several Petascale computer architectures (e.g., [110]). Fat trees con be configured in terms of bandwidth, rack layout, and cost. At the heart of the cost/performance trade-off, and thus a critical decision in supercomputer design, is the number of resources towards the root, which determines the *bisection bandwidth*. As computations requiring all-to-all exchanges are important, and as it is commonly believed that all-to-all bandwidth is bounded by the bisection bandwidth (cf. [58]), most fat tree networks are designed with full bisection bandwidth.

In this chapter, we show that all-to-all only requires half bisection bandwidth due to its inherent locality. We then demonstrate how to design slimmed fat trees that have optimal cost while still providing full all-to-all bandwidth and how to schedule all-to-all communications on general trees optimally. The main contributions of our work are:

- A general discussion of locality in the all-to-all problem.
- A construction of cost-optimal fat trees delivering full all-to-all bandwidth.
- An algorithm to compute optimal exchange patterns for a given fat tree.

We start with a description of the locality within all-to-all and intuitive examples for mappings to fat trees. We then present generic lower bounds for all-to-all communications on fat trees followed by a construction of optimal all-to-all schedules. We conclude our study with a set of simulated real-world examples.

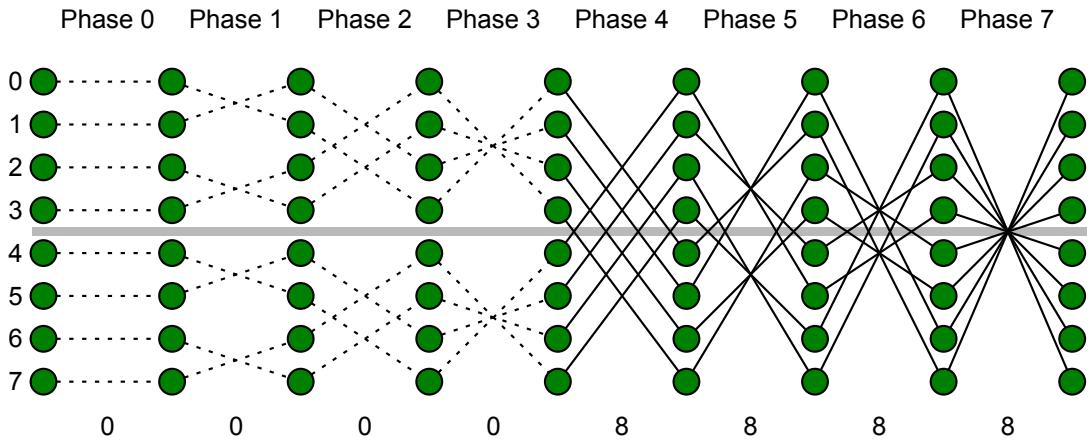


Figure 2.1 – XOR exchange pattern between 8 communicating tasks. The first half of the phases are local and no traffic crosses the network bisection (horizontal line), whereas during the last half of the phases, all the messages cross the bisection (continuous lines). This requires a bisection bandwidth equal to the aggregate injection bandwidth.
Figure included from [86]

2.2 The All-to-all Problem

In this section we will discuss the intuition behind the bounds for all-to-all. The full proof and the algorithm to compute the send permutations for a given fat tree will be presented in Sections 2.3.1 and 2.3.4.

All-to-all is often referred to as "bisection-bound" (cf.[58]). While this is true asymptotically, i.e., a full-bandwidth all-to-all requires a $\Omega(p)$ bisection bandwidth on p processes, we will show that it only requires half of the actual bisection bandwidth *if one can take advantage of the inherent locality of the all-to-all problem.*

Consider an all-to-all schedule where each process $p \in P$ exchanges a single message with each other process. Let P_1 and P_2 form an arbitrary bisection of the process set P, such that $b = |P_1| = |P_2| = \frac{|P|}{2}$. In an all-to-all, each process in the set P_1 has b peers in P_1 and b peers in P_2, such that only $b = \frac{|P|}{2}$ exchanges leave each set. If P_1 and P_2 were arbitrary bisections of processes mapped to endpoints in an arbitrary topology, then only half of the messages cross any bisection of this topology. Thus, intuitively, all-to-all exchanges require only half bisection bandwidth for arbitrary topologies.

The argument can be applied recursively, i.e., partitioning P_1 into P_{11} and P_{12} etc., yielding a bandwidth-minimal construction. We now demonstrate that current all-to-all algorithms do not take advantage of this observation and thus require full bisection bandwidth for a full-bandwidth all-to-all. We then show how to utilize our observation by optimally scheduling

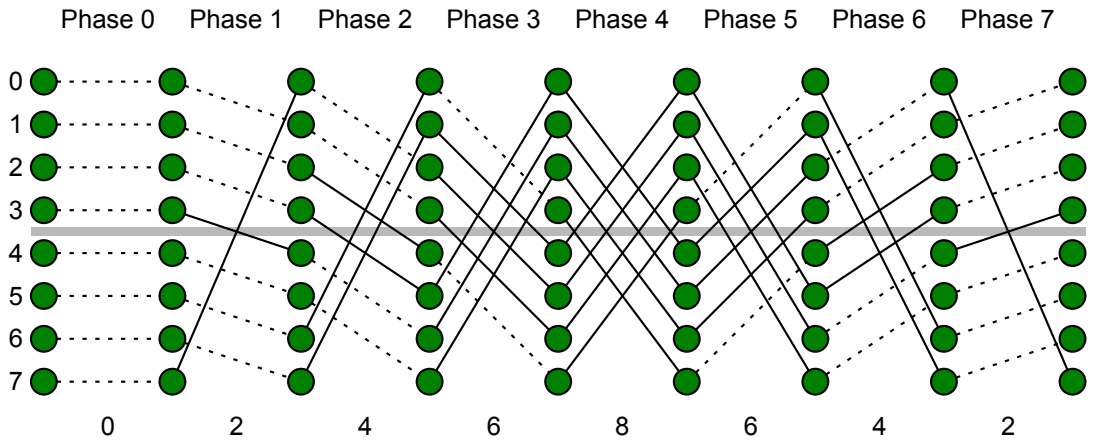


Figure 2.2 – Linear exchange pattern between 8 communicating tasks for shift = 0. The uneven distribution of the number of messages (continuous lines) crossing the network bisection requires a bisection bandwidth equal to the aggregate injection bandwidth to enable contention-free traffic in all phases.
Figure included from [86]

all-to-all communications such that they require only half bisection bandwidth and cause less congestion than today's algorithms.

A typical [113] way of performing an all-to-all exchange between N processes or tasks, especially when there is no possibility of message aggregation (such as is the case for large messages), is to schedule the exchange as a set of N phases. Every process sends a single message and receives a single message in each phase (we consider that a process sends a message to itself as well). In this case, the exchange pattern can be entirely defined by a function that maps a given phase p and source task s to a unique destination task d (all three numbered from 0 to $N-1$), meaning that during phase p, the message sent by task s is destined to task d. The two widely used exchange patterns are the binary XOR exchange and the linear shift exchange [113].

In the case of XOR (Figure 2.1), the function is $d_{\mathrm{XOR}} = s \mathrm{XOR} p$, where the exclusive or operation is performed bit by bit on the binary representations of s and p.

In the case of the linear shift exchange [90] (Figure 2.2), the function is $d_{\mathrm{LIN}} = (s+p+\mathrm{shift}) \bmod N$, where N is the total number of communicating tasks and shift is a parameter that takes a constant integer value between 0 and $N-1$.

If we partition the 8 nodes in the example all-to-all in the two equal halves separated by the horizontal median line it is obvious that both exchanges exhibit an uneven distribution of the amount of traffic traversing this bisection of the network across the different phases, with a peak required bisection bandwidth, in both cases, equal to the aggregate injection bandwidth.

Looking at the two examples, it is equally obvious that the average number of messages crossing the bisection as we have defined it in every phase is only half the total number of communicating tasks. Furthermore, there is a way to ensure that in every phase no more than this average number of messages crosses the bisection (Figure 2.3). This makes it possible to have uncontended per phase traffic with only half the full bisection bandwidth.

We will show in the following sections that the observations for this small example hold in general. We will prove tight lower bounds on the bandwidth required for uncontended all-to-all traffic. Furthermore, for fat tree networks in particular we will introduce a bandwidth-optimized exchange pattern that satisfies these bounds.

2.3 All-to-all optimization

First, we define the model of the interconnection network we are considering, the *fat tree* network. Fat trees are one of the most popular indirect network topologies used in current supercomputers.

The term fat tree has been used to refer to a broad class of different interconnection layouts. All fat trees can be described as a multi-stage tree-like topology where the width (bandwidth) of the connections increases towards the root of the tree.

An *ideal fat tree* would be a single binary tree interconnection network where the bandwidth of each link towards the root can always accommodate the aggregate capacity of the sub-tree connected to that link. These *ideal fat tree* networks are not realizable for large clusters, as the bandwidth needs to double at each level towards the root, which would require switches with either exponentially increasing number of ports or exponentially increasing bandwidth per link. The CM-5 [60] was the first machine to implement a variant of the ideal fat tree network.

However, it is possible to construct a tree-like network with an equivalent link bandwidth property by using fixed-radix switches, i.e., without increasing the number of connections per switch or their bandwidth [79, 80]. It is furthermore possible to construct networks [76] where the bandwidth towards the root can be tuned to almost arbitrary values, from the full-bisection bandwidth characteristic of ideal fat-trees to reduced bandwidth realizations that have a correspondingly reduced cost. We will discuss these and other practical implications of our method in Section 2.4.

For the time being, we will assume the following fat-tree model. We consider a layered single rooted tree structure that is homogeneous in that every node on a given layer has the same number of descendants. Using a notation similar to [76], we denote by $FT(L;M_1,M_2,...,M_L)$ a tree of $L+1$ consecutively numbered layers, with 0 being the layer of the leaves and L that of the single root. Given a layer l, M_l then denotes the number of descendants that every node on layer l has. The tasks that send and receive messages are located at the leaf nodes, whereas the nodes on the other levels serve as message routers. We will assume a single task per node

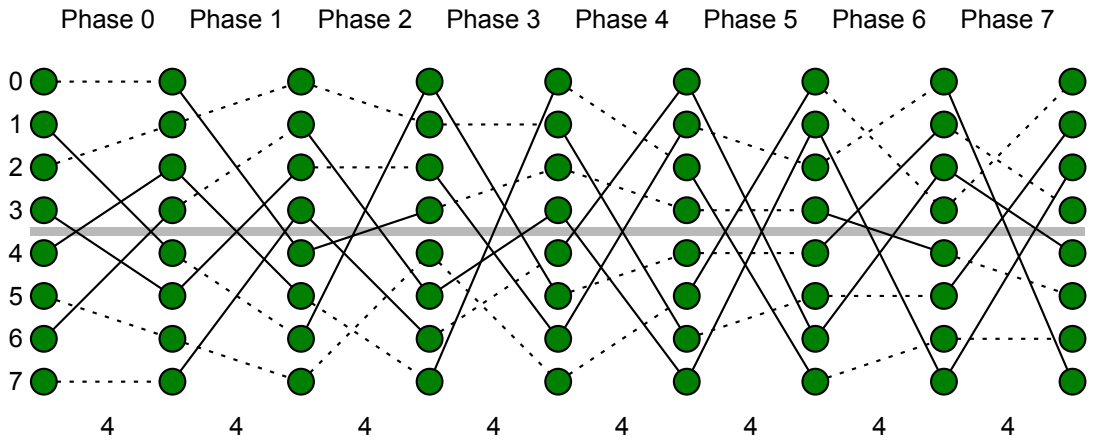


Figure 2.3 – Optimized exchange pattern obtained for a fat tree with 2 levels and with $M_2 = 2$ and $M_1 = 4$. The link occupation optimization ensures that in every phase exactly half of the messages (continuous lines) cross the network bisection, thus making contention free traffic possible with only half the full bisection bandwidth.
Figure included from [86]

(The case where several tasks are present on the same node is equivalent to extending the fat tree with an additional leaf level, where every leaf corresponds to a task). Furthermore, we will assume that every upward link starting on a node on a given layer has a fixed bandwidth dependent only on the layer index. In the context of this abstract model of the network, we will now prove that the requirement of uncontended all-to-all traffic induces a tight lower bound on the bandwidth of every fat tree link.

2.3.1 Bandwidth-optimal link occupation in all-to-all communication over a fat tree

Given a fat tree $FT(L;M_1,...,M_L)$ defined as above, we will show in this section that uncontended all-to-all traffic requires that every upward link originating on a layer l node as well as every downward link ending on a layer l node needs to be able to accommodate at least $B_{\min}(l)$ messages, with:

$$B_{\min}(l) = M_1 \cdot M_2 \cdot ... \cdot M_l - \left\lfloor \frac{M_1 \cdot M_2 \cdot ... \cdot M_l}{M_{l+1} \cdot M_{l+2} \cdot ... \cdot M_L} \right\rfloor . \tag{2.1}$$

To this end, we denote by Π_l the product $M_1 \cdot M_2 \cdot ... \cdot M_l$. Given that every leaf node corresponds to a single task, the total number of tasks is $N = \Pi_L$. Let us consider one of the nodes on level l, denoted by μ. The total number of messages originating from layer 0 descendants of this node during the all-to-all exchange is equal to the number of such descendants (which equals Π_l)

multiplied by the total number of destinations (which equals Π_L). Out of these Π_L destinations, Π_l are located in the subtree rooted in μ and thus messages destined to these "local" tasks need *not* be routed on the link going upward from μ. All messages destined to the other tasks, located outside the subtree rooted in μ, *must* be routed on μ's upward link. Consequently, during the complete all-to-all exchange, the number of messages passing through the upward link originating on any node on level l equals $U(l) = \Pi_l \cdot (\Pi_L - \Pi_l)$.

As the exchange comprises Π_L phases, the average per-phase number of messages passing through the upward link originating on a given node on level l therefore equals $\overline{U}(l) = \Pi_l \cdot \left(1 - \frac{\Pi_l}{\Pi_L}\right)$

Given an arbitrary exchange pattern and an arbitrary phase of that exchange, the maximum number of messages passing through an upward link at level l is equal to or greater than $\overline{U}(l)$. This is equivalent to saying that at least one link will have $\overline{U}(l)$ or more messages passing through it in that phase. Thus, a prerequisite for uncontended single-phase traffic is to provide every upward link with sufficient bandwidth to transfer at least this number of messages. As such, contention-free traffic imposes a lower bound of $\left\lceil \overline{U}(l) \right\rceil$ on the number of messages that every upward link starting on a layer l node must be able to accommodate, bound that is equal to $B_{\min}(l)$ as defined in Equation (2.1).

Through a similar analysis, we can compute the minimum bandwidth necessary on downward links. Considering again one arbitrary node μ on level l, the total number of messages destined to level 0 descendants of this node during the entire exchange is equal to $\Pi_l \cdot \Pi_L$. Out of the Π_L sources of these messages, Π_l are located in the subtree rooted in μ and thus none of the messages originating from them traverse the downward link ending in μ. Consequently, during the complete exchange, the number of messages passing through the downward link ending on any node on level l equals $D(l) = \Pi_l \cdot (\Pi_L - \Pi_l)$, which leads to an average per-phase number of downward messages equal to $\overline{D}(l) = \Pi_l \cdot \left(1 - \frac{\Pi_l}{\Pi_L}\right)$.

Applying the same argument as before, we conclude that every downward link ending on a layer l node must be able to accommodate at least $\left\lceil \overline{D}(l) \right\rceil$ messages, which is again exactly equal to $B_{\min}(l)$ defined in Equation (2.1).

Next, we will introduce an exchange pattern that requires no more than this minimum capacity, thus effectively proving that the bounds are tight. We will start by intuitively explaining the construction of the new pattern, then proceed to rigorously define it and prove its optimality.

2.3.2 Intuitive bandwidth-optimal exchange

Let us consider the example all-to-all exchange among 8 tasks presented in Section 2.2on a $FT(2;4,2)$ topology. For simplicity, let's start by optimizing the load induced by all tasks, on the two upward links ending in the root node.

As shown in Section 2.3.1, the key to optimizing bandwidth utilization is to balance the load

on each link across all phases. Obtaining a bandwidth utilization of $B_{\min}$ across all phases is only possible if every phase exhibits this utilization. In our specific example, we are interested in $B_{\min}(1) = M_1 - \lfloor M_1/M_2 \rfloor = 2$. Thus, we must find a way to ensure that an arbitrary upward link ending on the root is used by exactly two messages in each phase. One way to achieve this is for every source task s to choose in any two consecutive phases a destination that is in the same layer 1 subtree as itself and a destination that is in the other layer 1 subtree. However, this is not enough, as all 4 tasks in a subtree could all choose to send messages outside of the subtree in the same phases. Therefore it is also necessary that the source tasks belonging to the same subtree are desynchronized among themselves in their choice of the destination subtree. A simple subtree selection function that achieves this is $((s \bmod 2) + (p \bmod 2)) \bmod 2$. Once we have selected the subtree, we also need to ensure that the actual destination within that subtree is unique over the set of sources. This can be achieved by selecting among the 4 choices in a specific subtree via the formula $((s/2) + (p/2)) \bmod 4$, where / yields the quotient of integer division. The complete destination selection function can then be defined as: $d = (((s \bmod 2) + (p \bmod 2)) \bmod 2) * 4 + ((s/2) + (p/2)) \bmod 4$. This exchange is illustrated in Figure 2.3 and it is bandwidth optimal as exactly 4 messages cross the bisection in each phase.

We extend this to the general case of a $FT(L; M_1, ..., M_L)$ topology using the same principles. To balance load on the highest layer links, every source will send in M_L consecutive phases to M_L different subtrees. This approach is reused recursively from top to bottom.

The formalization and rigorous generalization of the approach follows in the next subsections. We first define a generic family of exchanges based on a *variable base representation* of numbers, then proceed to select among this set an exchange that we will prove to be bandwidth-optimal.

2.3.3 Permuted variable base exchanges

Given an ordered set of strictly positive integer numbers (Θ_1,Θ_2,...,Θ_K) and given a number x with $0 \le x < \Pi_{k=1}^{K} \Theta_k$, we define the following notation:

$$x = \Sigma_{k=1}^{K} \left(\theta_k^x \cdot \Pi_{k'=1}^{k-1} \Theta_{k'} \right), \ 0 \le \theta_k^x < \Theta_k, \ 1 \le k \le K. \tag{2.2}$$

Basically, this equation defines a way of representing positive integers in "variable base". Unlike a normal base-n representation where each digit can take any value between 0 and $n-1$, in this case each digit may have a different range. The θ_k digits define a representation in base (Θ_1,Θ_2,...,Θ_K), where the least significant digit's base is Θ_1. This representation, which

applies solely to non-negative integers strictly smaller than $\Pi_{k=1}^{K}\Theta_k$, can be easily proven to be unique and uniquely identifying a certain value. That is to say:

$$\begin{aligned} &\forall x,y \text{ s.t. } 0 \leq x,y < \Pi_{k=1}^{K}\Theta_k, \\ &x = y \Leftrightarrow (\theta_k^x = \theta_k^y, \ \ 1 \leq k \leq K). \end{aligned} \tag{2.3}$$

In the context of fat trees, we will assign a variable base representation to every leaf node, and consequently, to every task. For this, we will assign tasks with indices ranging from 0 to $\Pi_L - 1$ and choose appropriate variable bases to represent those indices in.

Given a fat tree $FT(L; M_1, ..., M_L)$, let $K = L$ and let $(\Theta_1,\Theta_2,...,\Theta_L) = (M_{\sigma(1)}, M_{\sigma(2)}, ..., M_{\sigma(L)})$, where σ is an arbitrary permutation of $(1,2,...,L)$. Additionally, we denote by α_k^x the digits of an arbitrary number x in the variable base $(M_1, M_2, ..., M_L)$. With these notations, we propose the following exchange pattern, which we will refer to as a *permuted-variable-base* exchange pattern. Let d be the destination of the message generated by source s in phase p. All three values s, d and p are numbers between 0 and $\Pi_{k=1}^{K}M_k - 1$. Then the function defining the pattern is:

$$\alpha_{\sigma(k)}^{d} = (\theta_k^s + \theta_k^p) \mod \Theta_k, \ \ 1 \leq k \leq K. \tag{2.4}$$

We will now show that this exchange model satisfies the two key properties of a valid all-to-all exchange pattern composed of phases: a source never sends to the same destination twice and no two sources send to the same destination in the same phase.

For the first property, let p_1 and p_2 be two distinct phases and let s be a certain source. We seek to prove that in this case, d_1 and d_2 obtained with Equation (2.4) are distinct.

Because p_1 and p_2 are positive and strictly smaller than $\Pi_{k=1}^{K}M_k$, they have a θ-representation. Furthermore, since they are distinct, according to Equation (2.3) there exists at least one $\tilde{k}$ with $1 \leq \tilde{k} \leq K$ such that $\theta_{\tilde{k}}^{p_1} \neq \theta_{\tilde{k}}^{p_2}$. According to Equation (2.4):

$$\theta_{\tilde{k}}^{s} + \theta_{\tilde{k}}^{p_1} = q_1 \cdot \Theta_{\tilde{k}} + \alpha_{\sigma(\tilde{k})}^{d_1}$$

where q_1 is an integer. Given that

$$0 \leq \theta_{\tilde{k}}^{s} + \theta_{\tilde{k}}^{p_1} < 2 \cdot \Theta_{\tilde{k}}$$

and

$$0 \leq \alpha^{d_1}_{\sigma(\tilde{k})} < \Theta_{\tilde{k}}$$

it follows that

$$-\Theta_{\tilde{k}} < q_1 \cdot \Theta_{\tilde{k}} < 2 \cdot \Theta_{\tilde{k}}.$$

As q_1 is an integer, it can only be 0 or 1. Similarly, q_2 has the same property. In the case where both are 0 or both are 1 ($q_1 = q_2 = q$), it immediately follows that $\alpha^{d_1}_{\sigma(\tilde{k})} \neq \alpha^{d_2}_{\sigma(\tilde{k})}$ from:

$$\theta^{s}_{\tilde{k}} + \theta^{p_1}_{\tilde{k}} - q \cdot \Theta_{\tilde{k}} = \alpha^{d_1}_{\sigma(\tilde{k})}$$

$$\theta^{s}_{\tilde{k}} + \theta^{p_2}_{\tilde{k}} - q \cdot \Theta_{\tilde{k}} = \alpha^{d_2}_{\sigma(\tilde{k})}$$

$$\theta^{p_1}_{\tilde{k}} \neq \theta^{p_2}_{\tilde{k}}$$

In the case where q_1 is 1 and q_2 is 0, we obtain

$$\theta^{p_1}_{\tilde{k}} - \theta^{p_2}_{\tilde{k}} = \alpha^{d_1}_{\sigma(\tilde{k})} - \alpha^{d_2}_{\sigma(\tilde{k})} + \Theta_{\tilde{k}}.$$

If $\alpha^{d_1}_{\sigma(\tilde{k})}$ were equal to $\alpha^{d_2}_{\sigma(\tilde{k})}$, then $\theta^{p_1}_{\tilde{k}} - \theta^{p_2}_{\tilde{k}} = \Theta_{\tilde{k}}$, which would lead to $\theta^{p_1}_{\tilde{k}} \geq \Theta_{\tilde{k}}$, which would constitute a contradiction. Therefore, in this case as well, $\alpha^{d_1}_{\sigma(\tilde{k})}$ must be different from $\alpha^{d_2}_{\sigma(\tilde{k})}$. The symmetric case ($q_1$ is 0 and q_2 is 1) is similar. As we proved that there exists a $\tilde{k}$ for which $\alpha^{d_1}_{\sigma(\tilde{k})} \neq \alpha^{d_2}_{\sigma(\tilde{k})}$, we proved that $d_1 \neq d_2$.

The proof of the second property is similar as the computation of the destination task as given by Equation (2.4) is symmetric with respect to the phase number p and the source number s.

This shows that every permutation σ of $(1,2,...,L)$ induces a valid exchange pattern. However,

not all of these patterns optimize the maximum traffic traversing the upward links in the fat tree. In the following subsection, we will prove that the specific exchange pattern corresponding to $\overline{\sigma} = (L, L-1, ..., 2, 1)$ is bandwidth-optimal.

2.3.4 Bandwidth-optimal exchange pattern

As explained above, let $(\Theta_1, \Theta_2, ..., \Theta_L) = (M_{\overline{\sigma}(1)}, M_{\overline{\sigma}(2)}, ..., M_{\overline{\sigma}(L)}) = (M_L, M_{L-1}, ..., M_1)$.

We will prove that the bandwidth requirements induced at layer l by the variable-base-exchange corresponding to this particular base is equal to $B_{\min}(l)$.

We will start with the per phase bandwidth requirement per upward link. First note that all the descendants of a given node on level $\tilde{k}$ share the same α_k values for all k such that $\tilde{k} < k \leq L$.

Let p be a given phase of the all-to-all exchange. Let μ be an arbitrary node at level $\tilde{k}$. The total number of messages originating in level 0 descendants of this node during phase p is $\Pi_{\tilde{k}}$. Among these messages, some will be destined to nodes that are descendants of μ and some to nodes that are not. For a message to be destined to a descendant of μ, as noted above, its destination d must have certain fixed values for some of its α coefficients, values that depend exclusively on the position of μ. Specifically:

$$\alpha_k^d = \alpha_k(\mu), \;\; \tilde{k} < k \leq L.$$

As d is obtained via Equation (2.4) this leads to:

$$(\theta^s_{L+1-k} + \theta^p_{L+1-k}) \mod M_k = \alpha_k(\mu), \;\; \tilde{k} < k \leq L.$$

As for a given p, the θ representation of p is fixed, and θ^s_{L+1-k}, θ^p_{L+1-k} and $\alpha_k(\mu)$ are all strictly smaller than M_k, this means that the θ representation of s needs to satisfy:

$$\theta^s_{L+1-k} = t_k(\mu, p), \;\; \tilde{k} < k \leq L$$

where $t_k(\mu, p)$ is a set of factors independent of s. So for a descendant s of μ to send a packet to another descendant of μ in phase p of the exchange, the θ coefficients of s from rank 1 to rank $L - \tilde{k}$ must all have values that depend exclusively on μ and p, but not on s. According to the θ representation definition in Equation (2.2), this is equivalent to:

$$s = r(\mu,p) + q(s,\mu,p) \cdot \Pi_{k=1}^{L-\tilde{k}} \Theta_k$$

where $0 \leq r(\mu) < \Pi_{k=1}^{L-\tilde{k}} \Theta_k$. Thus, the only sources s that send messages to other descendants of μ are those whose index yields a fixed remainder (dependent of μ and p) when divided by $\Pi_{k=1}^{L-\tilde{k}} \Theta_k$. Because the s values are consecutive among the descendants of μ and because in total there are $\Pi_{\tilde{k}}$ such descendants, the minimum number of sources that send to a destination that is also a descendant of μ is:

$$N_{min}(\tilde{k}) = \left\lfloor \frac{\Pi_{\tilde{k}}}{\Pi_{k=1}^{L-\tilde{k}} \Theta_k} \right\rfloor$$

which according to the definition of the base is equal to:

$$N_{min}(\tilde{k}) = \left\lfloor \frac{\Pi_{\tilde{k}}}{\Pi_{k=\tilde{k}+1}^{L} M_k} \right\rfloor. \tag{2.5}$$

The rest of the messages could potentially go outside of the subtree rooted in μ. Therefore, the maximum bandwidth (in terms of number of messages) necessary on the upward link starting on any node on level $\tilde{k}$ is:

$$B_{max}(\tilde{k}) = \Pi_{\tilde{k}} - \left\lfloor \frac{M_1 \cdot M_2 \cdot \ldots \cdot M_{\tilde{k}}}{M_{\tilde{k}+1} \cdot M_{\tilde{k}+2} \cdot \ldots \cdot M_L} \right\rfloor \tag{2.6}$$

which exactly equals the value we previously found for the minimum bandwidth theoretically necessary in a fat tree for contention-free all-to-all messaging.

The per-phase bandwidth requirement per downward link can be computed analogously. During a given phase, at most $\Pi_{\tilde{k}}$ messages in total are sent to the $\Pi_{\tilde{k}}$ level 0 descendants of a node μ of the fat tree on level $\tilde{k}$. It can be shown with an argument analogous to the one above that among these messages, a minimum number must originate from these same descendants, and that that minimum number is equal to $N_{min}(\tilde{k})$ in Equation (2.5). As such, by the same argument as before, we obtain the same upper bound as that given by Equation (2.6) for the bandwidth required on the downward links.

2.3.5 Discussion

In this section, we have derived tight lower bounds on the bandwidth requirement of the all-to-all exchange in fat tree networks. We have also introduced an optimal exchange pattern that uses no more than the minimal bandwidth.

This new exchange pattern allows for a better use of network resources, freeing bandwidth for other traffic if the tree was designed with bandwidth in excess of the minimal bound, or, alternatively, allowing for reduced-cost fat trees that perform optimally under the all-to-all.

Compared to a standard fat tree that offers full bisection bandwidth at every level, the amount of bandwidth that is left unused, or, alternatively can be removed altogether from the tree is given, for a layer l link, by $\Pi_l - B_{\min}(l)$. This bandwidth surplus is largest at the top of the tree and decreases rapidly with the layer index. Nonetheless, at the highest level we can leave unused as much as $\frac{1}{M_L}$ of the full bisection bandwidth. Given certain conditions (specifically $M_L = 2$), this means we can achieve optimal all-to-all traffic with only half the full bisection bandwidth.

Because the amount of redundant bandwidth at lower layers of the tree decreases exponentially, in practice only a few top levels will allow significant reduction of bandwidth usage while some lower levels will allow no reduction at all. This, coupled with the fact that the fully optimized pattern as described above is highly intricate, leads to an interest in performing the optimization only at the levels where reduction occurs. We have developed methods similar to the one presented above to tune the complexity of the pattern while maintaining optimality but these will be discussed in a subsequent work.

2.4 Practical considerations

In this section we will address some issues that arise from the fact that, in practice, fat tree networks are not implemented by means of ideal fat trees. This is because in practice the exponential increase in link bandwidth that is typical of multi-layered fat trees can only be achieved by means of having an exponentially increasing number of links between a single pair of nodes. This in turn leads to a necessity of switches with a very large port count, which are not feasible.

Therefore, in practice, fat tree topologies are approximated by means of k-ary n-trees [79, 80] or more generally by means of Extended Generalized Fat Trees (XGFTs) [76]. These network designs offer an alternative that only needs fixed small-radix switches and still provides many of the properties of ideal fat trees. However, the advantages come at the expense of reduced flexibility in bandwidth configuration and, more importantly, at the expense of needing more complicated routing algorithms.

The following two subsections briefly outline how the routing needs to be adapted to take full advantage of the optimal exchange pattern and provide an estimate of the cost saving that can

be achieved by reducing the bandwidth to the theoretical lower bound for optimal all-to-all exchanges.

2.4.1 Routing in XGFTs

In practical fat tree realizations, optimizing the exchange pattern is not sufficient to benefit from contention-free traffic. The optimal exchange pattern introduced in Section 2.3.4 only ensures that, in each phase, the *aggregate* bandwidth at each fat-tree-equivalent-node of the network matches the aggregate number of messages traversing that node.

This is a necessary but not a sufficient condition for contention-free message routing. In addition, the routing strategy must ensure that the messages indeed use the available bandwidth evenly, i.e., it must assign conflict-free paths for all messages in each phase.

In XGFT networks, the typical way to establish minimal deadlock-free paths is simple up*/down* routing [96]. Given a (source, destination) pair, paths are constructed by routing upwards to any of the Nearest Common Ancestor (NCA) nodes and then downwards to the destination. Once a specific NCA is chosen, the routing choices from source to NCA and from NCA to destination are uniquely determined. The specificity of a certain routing strategy then lies in the approach used to select a specific NCA for every (source, destination) pair.

Several traffic-pattern-oblivious approaches exist, from random [29, 37] or adaptive [33, 65] route selection to deterministic (e.g., modulo based) approaches such as *S*ource-*mod-k* [49, 60, 76] or *D*estination-*mod-k* [61, 91, 114].

For complex patterns such as ours however, oblivious routing can prove insufficient. A pattern-aware routing approach takes advantage of information about the communicating pairs in a network and information about the network itself and uses it to find an optimized set of routes that will minimize contention for the set of (source, destination) pairs. Pattern-aware schemes solve in some way or another a *max-flow* network problem. A variety of approaches to achieve this are presented by Kinsy et. al [55]. Among these approaches, we were able to derive optimal conflict-free routes for the bandwidth-optimal exchange that we are proposing by means of Mixed Integer-Linear Programming.

2.4.2 Cost effective fat tree networks

Since the optimized exchange exhibits contention free all-to-all phases in reduced bandwidth topologies, we consider the network cost savings that this exchange allows.

Given an XGFT($h : m_1, m_2, ..., m_h : w_1, w_2, ..., w_h$) [76], the number of switches necessary to build the layer l of the topology is $S(l) = w_1 \cdot ... \cdot w_l \cdot m_{l+1} \cdot ... \cdot m_h$. The number of bidirectional links that connect layer l to layer $l-1$ is $E(l) = S(l) \cdot m_l$.

To simplify the analysis, consider a simple cost reduction scenario. Let's start with a full

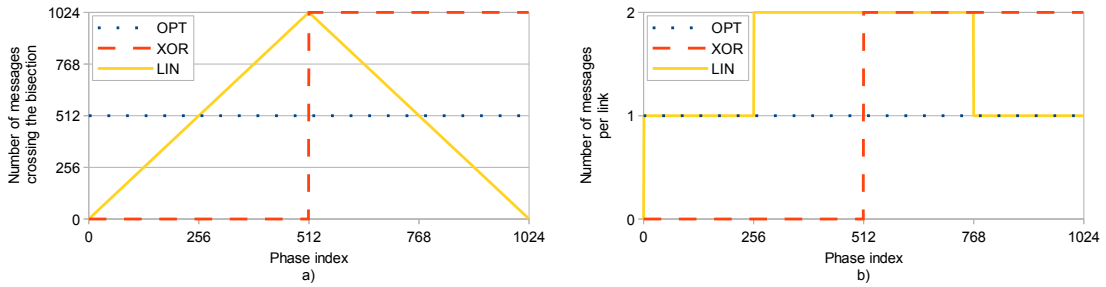


Figure 2.4 – Bisection (top level) occupation induced by XOR, LIN and OPT exchange patterns. The topology is an XGFT with $M_L = 2$ with half bisection bandwidth and total number of leaf nodes equal to 1024. Figure a) shows the total number of messages crossing the bisection in each phase, while Figure b) shows the best value for maximum link contention (in number of messages), or best contention level, that can be obtained in each phase.
Figure included from [86]

bisection bandwidth XGFT (i.e., where $w_{l+1} = m_l, 1 \leq l < h$ and $w_1 = 1$) that additionally has $m_h = 2$. This is the case we have shown to have the highest potential for bandwidth reduction. The reduced tree, XGFT$'(h : m_1, m_2, ..., m_h : w_1, w_2, ..., w'_h)$, has exactly the same parameters as the full bisection bandwidth tree, with the exception of w'_h which equals $w_h/2$.

Given the properties shown above, we can see that, for $1 \leq l < h$, $E(l) = E'(l) = E = w_1 \cdot m_1 \cdot ... \cdot m_h$ and $S(l) = S'(l) = E/m_l$. For $l = h$ we have $E(h) = E$, $E'(h) = w_1 \cdot m_1 \cdot ... \cdot m_h/2 = E/2$, $S(h) = E/m_h$ and $S'(h) = E/(2m_h)$.

The saving potential in the number of switches equals

$$1 - \frac{\sum_{l=1}^{h} S'(l)}{\sum_{l=1}^{h} S(l)} = 1 - \frac{1/(2m_h) + \sum_{l=1}^{h-1} 1/m_l}{1/m_h + \sum_{l=1}^{h-1} 1/m_l}$$

and in the number of links equals

$$1 - \frac{\sum_{l=1}^{h} E'(l)}{\sum_{l=1}^{h} E(l)} = 1 - \frac{(h - 1/2) \cdot E}{h \cdot E} = \frac{1}{2h}.$$

For typical XGFTs interconnecting 16 to 1024 leaf nodes, we can thus *achieve a reduction in the number of switches of* 25% – 30% *and a reduction in the number of necessary links of* 12% – 16%.

2.5 Experiments

The following results were obtained by means of a simulation framework that is able to accurately model custom networks (including XGFTs) at a flit level [69]. The simulator provides a high level of customization and modularity, allowing the configuration of the desired model

in detail. Several tests have been performed on XGFTs of 16, 32, 64, 128, 256, 512 and 1024 leaf nodes and up to 4 non-leaf levels. All these fat tree topologies had the M parameter of the topmost level equal to two ($M_L = 2$) and half bisection bandwidth. Table 2.1 shows the exact configurations used.

N	Network topology
16	$XGFT[3:4,2,2:1,4,1]$
32	$XGFT[3:4,4,2:1,4,2]$
64	$XGFT[3:8,4,2:1,8,2]$
128	$XGFT[3:8,8,2:1,8,4]$
256	$XGFT[4:8,4,4,2:1,8,4,2]$
512	$XGFT[4:8,8,4,2:1,8,8,2]$
1024	$XGFT[4:8,8,8,2:1,8,8,4]$

Table 2.1 – Benchmarked topological configurations, using the notation from [76].
Table included from [86]

The simulations were performed using output-buffered switches with 4 kbytes of buffer space per port. The links had a bandwidth of 10 Gbit/s while their latency was either set to an ideal value of zero (to validate theoretical estimations) or to a realistic value of 100 ns (to estimate performance on real-life systems). In this same realistic latency scenario, switch traversal was considered to have a latency of 50 ns while adapter latency (the time between a message is issued in the MPI library and the time its first bit leaves the adapter) was set to 500 ns. Credit based flow control was used as well as wormhole switch traversal.

The messages have a constant size which, depending on the scenario, took values between 64 bytes (a single flit) and 32 kbytes, while the flit size was fixed to 64 bytes. Each acknowledgement consisted of a single flit.

The traffic pattern is that of a single all-to-all collective exchange performed by one task on each of the leaf nodes of the XGFT. The exchange is split into consecutive phases in each of which every node sends a single message and receives a single message and acknowledges it. No explicit synchronization or separation of the phases is enforced. The exact structure of each phase (the set of (source, destination) pairs) is defined according to one of three exchange models, all defined in Sections 2.2 and 2.3.4:

- XOR: the binary XOR model,
- LIN: the linear shift model with a shift value of 0,
- OPT: the optimized exchange given by Equation (2.4).

The metric we used is that of the time necessary for the traffic pattern to complete. However, these completion times can amount to very different values when looking at different sized

networks and even at the same network with messages of different size. We are not interested in how these parameters influence the completion time, but rather in the influence of the exchange pattern within each given (network size, message size) configuration. As such, all results show the relative completion time compared to an ideal completion time that we will define in the following.

2.5.1 Ideal all-to-all performance

We use the following model to compute a best-case expectation for the completion time of the all-to-all exchange. Ideally, the messages travel through the network with no contention and simply incur the cumulative latency of the links, switches and adapters that they traverse. During the exchange, a given source sends $M_1 \cdot M_2 \cdot ... \cdot M_{l-1} \cdot (M_l - 1)$ messages to sources in its layer l subtree but not in its layer $l-1$ subtree, for every l. These messages need to first arrive at the root of the subtree, passing through 1 adapter, l links and $l-1$ switches, then cross the root, then travel again through 1 adapter, l links and $l-1$ switches to get to their destination. The cumulative latency for the entire path is thus $t_{\text{path}}(l) = 2 \cdot t_{\text{adapter}} + (2 \cdot l - 1) \cdot t_{\text{switch}} + 2 \cdot l \cdot t_{\text{link}}$.

Given a message that is made up of F flits, each of size S, and a link bandwidth of B, uncontended wormhole switch traversal ensures that the time it takes end-to-end, from message generation to message delivery, equals $t_{\text{path}}(l) + F \cdot S/B$.

The message needs to be acknowledged, which is equivalent to an extra one-flit message being sent along the reverse path. This leads to the total message time being: $T(l) = 2 \cdot t_{\text{path}}(l) + (F + 1) \cdot S/B$.

Given the message distribution we described above, the ideal exchange time is given by

$$T = \sum_{l=1}^{L} M_1 \cdot M_2 \cdot ... \cdot M_{l-1} \cdot (M_l - 1) \cdot T(l)$$

2.5.2 Ideal latency analysis and measurements

To validate our simulation framework, we analyze the network contention induced by the three approaches when other performance limiting factors such as latency and latency-induced phase overlapping are factored out.

The XOR exchange exhibits no contention in half of its phases, namely in the phases where messages do not reach the top level of the fat tree. In the other half, N messages cross the top level, but have only half bisection bandwidth available, leading to a contention level of exactly two conflicts in every one of these $N/2$ phases. We defined the contention level as being the maximum number of messages that need to use a common link.

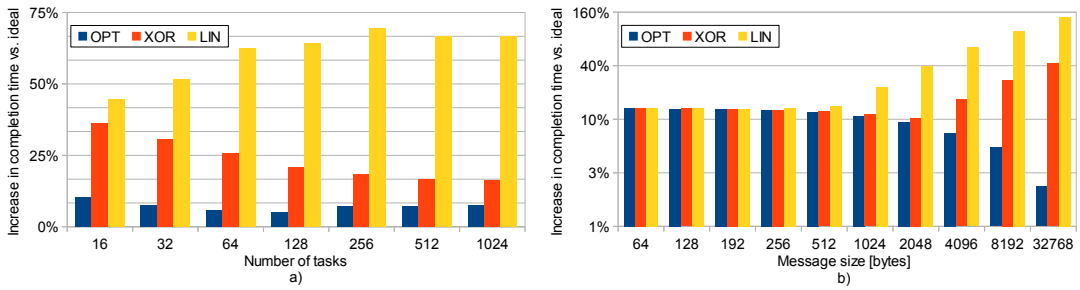


Figure 2.5 – Completion time comparison between the three exchange patterns for varying networks sizes under fixed 4 KB message size (Fig. a) and for varying message size under fixed 512 leaf nodes network size (Fig. b). Please note the linear scale for the y axis in a) and the logarithmic scale for the y axis in b).
Figure included from [86]

In the case of the LIN exchange, exactly $N - |2p - N|$ messages cross the top level of the tree in phase p, leading to a contention level of 2 yet again in exactly $N/2$ phases (the phases where $N - |2p - N| > N/2$).

In the case of the OPT exchange, exactly $N/2$ messages cross the top level of the tree in every phase, leading to uncontended traffic in every phase (Figure 2.4).

This implies that, in absence of latency and latency-induced phase overlapping, we can estimate theoretically that OPT will have a completion time close to the optimum, while LIN and XOR will incur a doubling of the completion time of half the phases, leading to at least a 50% increase in all-to-all completion time. This is confirmed accurately by our simulator, which yields completion times for OPT within 1% of the ideal time and completion times for XOR 50% – 55% longer than ideal, both irrespective of the message size. LIN performs even worse, with completion times between 70% and 140% larger than ideal (the performance loss increases with both the network and the message size). This is due to additional phase overlapping induced by intra-phase differences in message delivery time caused by different flows exhibiting different levels of contention (which is not the case for OPT and XOR).

2.5.3 Realistic latency results

We performed two sets of benchmarks to predict OPT performance on real systems. The first one is aimed at studying the influence of the network size on the performance gain induced by the optimized exchange pattern. The second on the other hand studies the influence of the size of the exchanged messages on the same performance gain.

Figure 3.4 a) illustrates the relative completion time of the three exchange patterns for a fixed message size of 4 kbytes and for different network sizes. We observed that in the realistic latencies configuration OPT managed to achieve completion times less than 10% higher than ideal, surpassing XOR (which is between 15% and 35% worse than ideal) and LIN (which is

between 50% and 70% worse than ideal). The advantage of the optimized exchange versus XOR was especially strong for smaller topologies.

Figure 3.4 b) illustrates the relative completion time of the three exchange patterns for a fixed number of 512 communicating tasks (and a fixed network size) and for different sizes of the exchanged messages. We observed that for messages smaller than approximately 512 bytes, where the completion time is latency dominated, the optimization of the exchange doesn't help too much, because the impact of contention was small. For messages larger than this threshold, where the completion time was congestion dominated, OPT achieved an increasingly closer to ideal level of performance (reaching only a 2% – 5% difference for very large messages), while the non-optimized exchanges became progressively worse as the message size increased, reaching as much as a 40% performance decrease for XOR and as much as 140% for LIN, for very large messages. This confirms that our approach is asymptotically bandwidth optimal.

2.6 Related work

Collective optimizations are increasingly important in large-scale high-performance computing. Thus, numerous research works deal with the optimization of collective operations in general and all-to-all in particular. Most traditional approaches assume theoretical network models, such as the Postal model or the LogP model. Well-tuned algorithms exist for both models [6, 20]. However, all theoretical models (with the notable exception of LoGPC [72], which has not been used to model all-to-all algorithms due to its complexity), do not consider the network topology and thus only optimize endpoint congestion. Our work, however, uses a simple topology-specific model for optimizing all-to-all communications. Our model can be combined with existing network models to model endpoint congestion more accurately and can also be used as a blue-print for optimizing other collectives on fat trees. However, our contribution lies in the scheduling algorithm rather than the model.

Topology becomes more important with growing system size and several researchers acknowledge the fact that algorithms purely based on the mentioned theoretical models do not perform well on real systems. For example, Chan et al. [21] provide a general abstract framework for topology-aware collective communication. However, the abstract nature of this framework makes it hardly applicable to the recursive structure of fat trees. Zahavi et al. [113, 115] show that a particular routing function (a variant of up*/down*) guarantees under certain placement conditions full-bandwidth all-to-all exchanges implemented using typical collective permutation sequences on full bisection bandwidth fat trees. We improve upon this result by a factor of two, in that our methods applied to networks with half the full bisection bandwidth exhibit no or negligible decrease in performance by comparison. Sack and Gropp present a small set of topology-aware collectives for fat tree and torus networks in [95], however, in this latest work, they do not consider all-to-all and their approach of adding communication does not lead to improvements for all-to-all. In previous work [103],

targeted at the implementation of optimized collectives in MPICH, the authors proposed the selection among a pool of implementations of different collective algorithms depending on message size and number of communicating processes, but not on any specific topology.

Other approaches for optimizing collectives, for example, improving the lower-level send/receive primitives [64], non-blocking collectives [38, 39], or hierarchical exchanges [43, 99] are completely orthogonal to the scheduling of messages. Those techniques can use our message-scheduling algorithm for improving the bandwidth of all-to-all.

2.7 Summary

The goal of this chapter was to show that the current state of the art exchange patterns for the personalized all-to-all collective message exchange can be optimized in fat tree topologies to the point of using half of the bisection bandwidth required by previous proposals.

To this end, we have derived a theoretical bound on the network link occupation that is intrinsically necessary and sufficient for any all-to-all exchange and we have shown that current approaches require as much as twice that proven optimally minimum. Furthermore, we introduced a message exchange pattern that we demonstrated to be bandwidth-optimal and that thus uses exactly the minimum required amount of network resources.

Finally, by conducting network simulations benchmarking the proposed method against established approaches, we confirmed that in practical scenarios the optimizations we proposed achieve important reductions in network utilization, or alternatively network complexity and cost (by reducing the top level of the network such that only half the bisection bandwidth remains available), with little impact on performance.

Furthermore, we demonstrated that in the half-bisection network scenario, where a cost-effective network with only the minimum amount of necessary resources was used, current methods incurred significant performance penalties, (in the 12% to 40% range for XOR and in the 12% to 140% range for LIN) whereas our method only diverged by at most 12% from optimal performance.

In conclusion, our method enables new design options for fat tree networks and offers a viable way of maintaining close to ideal application performance levels with important reductions in cost.

3 Fast Pattern-Specific Routing for Fat Tree Networks

In the context of extended generalized fat tree (XGFT) topologies, widely used in HPC and datacenter network designs, we propose a generic method, based on integer linear programming (ILP), to efficiently determine optimal routes for arbitrary workloads. We propose a novel approach that combines ILP with dynamic programming, effectively reducing the time to solution. Specifically, we divide the network into smaller subdomains optimized using a custom ILP formulation that ensures global optimality of local solutions. Local solutions are then combined into an optimal global solution using dynamic programming. Finally, we demonstrate through a series of extensive benchmarks that our approach scales in practice to networks interconnecting several thousands of nodes, using a single-threaded, freely available linear programming solver on commodity hardware, with the potential for higher scalability by means of commercial, parallel solvers.

3.1 Motivation

The suitability of an interconnection network design is typically quantified via simple metrics such as link bandwidth/latency and bisection bandwidth. However, these metrics lead to optimistic estimations on the achievable performance, the latter depending on the exact combination of communication patterns and routing algorithm being used. Inefficient routing approaches can induce significant levels of network congestion, which will negatively impact network performance, in terms of both throughput and latency.

Several characteristics of network topologies determine the options for routing algorithms. An important feature of any interconnection network is path diversity. Path diversity denotes the number of different ways to route messages between a given source and destination and consequently quantifies the breadth of the space of possible routing algorithms and the

This chapter is based on the article [87] PRISACARI, B., RODRÍGUEZ, G., MINKENBERG, C., AND HOEFLER, T. Fast pattern-specific routing for fat tree networks. *ACM Transactions on Architecture and Code Optimization (TACO) 10*, 4 (2013), 1–25. https://doi.org/10.1145/2541228.2555293

difficulty of choosing a good algorithm. An intelligent choice of the path that each message will follow will have important benefits, e.g. load balancing and congestion reduction, and will translate to an increase in workload performance. On the other hand, a suboptimal choice can introduce artificial performance limitations that are neither inherent to the workloads being accommodated nor to the network itself.

Some of the most popular choices for the interconnection fabric of large scale systems are variants of the fat tree architecture: fat trees [60], k-ary n-trees [80] or generic extended generalized fat trees (XGFTs) [76]. In the enterprise datacenter space, fat trees are the network architecture of choice [4] whereas in the high performance computing (HPC) space they are by far the dominating topology employed in clusters using InfiniBand™ technology, clusters which account for more than 40% of the systems in the latest Top 500 list [1, 18]. A few prominent examples of current HPC installations using fat tree interconnects are TACC Stampede [102], LRZ SuperMUC, and Pangea, the world's fastest private supercomputer, which are ranked as numbers 6, 9 and 11 in the latest Top 500 list [1], respectively.

All fat tree architectures are characterized by an amount of path diversity that increases roughly linearly with the size of the network, making the route selection problem both very important and very challenging. Currently, there is—to the best of our knowledge—no general solution to the problem of determining optimal routes for arbitrary workloads running on arbitrary fat trees of reasonable size. In practice, approximate solutions obtained either heuristically or through dynamic or adaptive approaches, where the routing decision is dependent on the current state of the network, are used to approximate optimal behavior and performance. However, for certain workloads such approaches can prove to be insufficient.

This work was originally motivated by the work described in detail in Chapter 2. There, we tackled the problem of optimizing the communication pattern of the *all-to-all exchange* collective operation, encountered in many parallel programming models, for XGFT networks with less then full bisection bandwidth. Having established a theoretically optimal way of performing such an all-to-all exchange, we found that optimizing a communication pattern for a specific network topology was not sufficient: although the workload and the network guaranteed optimal performance, none of the existing routing strategies were able to sustain it. In other words, the need arose for a practical and scalable method capable of finding optimal routes for a given communication pattern and a given (XGFT) topology. This chapter fills this gap, providing a solution for arbitrary communication patterns on arbitrary XGFT networks.

The core idea of our method is to perform offline a highly optimized search over all possible routing algorithms, using Integer Linear Programming (ILP). Although this is not a new idea, previous attempts have failed in applying it to real systems and concluded that the method is NP-complete and impractical for all but the smallest network sizes [25]. This chapter presents a set of key contributions that enabled us to find optimal routes for networks interconnecting several thousands of nodes in a practically feasible amount of time (in the range of hours).

After an initial survey of related work already performed in the field (Section 5.2), we start

by introducing a novel topological characterization of XGFT networks (Section 3.3) that will enable us to construct an efficient ILP formulation of the optimal routing assignment problem, for arbitrary workloads (Section 5.3). We proceed to explain how this solution can be effectively used in practice and present a set of experimental results that illustrate the benefits that using this approach can bring in a practical scenario (Section 3.5). Finally, we demonstrate (Section 3.6) by means of a set of extensive benchmarks that our approach is able to generate optimal routing assignments using only commodity hardware and an open source, free, single-threaded linear programming solver, in a reasonable amount of time, for networks interconnecting thousands of nodes. Thus, although a polynomial time-to-solution is not guaranteed, we show that through a series of acceleration strategies (introduced in Section 5.3) our approach reduces the completion time to very manageable values (e.g., in the order of one hour for a network interconnecting a thousand nodes), making, to the best of our knowledge, the current work the first to achieve rendering an ILP-based optimal routing assignment generation approach practical for arbitrary communication patterns in production systems.

3.2 Previous work on linear programming driven route optimization

Finding an optimal routing for a specific communication matrix and network topology is equivalent to the well-known maximum flow problem, or its generalization, the multi-commodity flow problem [59, 100]. For the general problem, considering any directed graph, and allowing for *fractional* flows, non-integral linear programming formulations exist that are solvable in polynomial time. However, for integer flows the problem is NP-complete [26].

Several works exist that do not seek to obtain the optimum integer solution but rather use the fractional solution as a basis for close-to-optimal routes and then provide upper bounds on the expected difference to an actual optimal solution. Räcke [89], for example, showed that it is possible to find an *oblivious* routing for any symmetric network (undirected and with equal capacities in both directions), such that any traffic pattern would only experience a maximum of a poly-logarithmic increment in contention with respect to the optimum for that specific traffic pattern. Azar [11] later corrected the bound to $O(N^{0.5})$, with N being the number of communicating nodes, and developed a linear programming formulation that can be solved with the Ellipsoid algorithm to obtain a solution in polynomial time. The proposed LP formulation, however, grows exponentially [9] with respect to the network size. Applegate [9] later provided an LP formulation that is polynomial in size with respect to the network size. Building on Räcke's and Azar's work, Applegate further showed that a robust routing can be obtained with little knowledge of the traffic demands. For the sample Internet Server Provider (backbone) networks studied, optimizing the oblivious case (where all source-destination pairs are considered with equal probability) was shown to be significantly more robust than optimizing for specific traffic patterns.

A more recent work [56] takes an additional step towards the integral max-flow problem by

using a Mixed Integer LP formulation to find a set of routes for *unsplittable* flows, i.e., flows that cannot be separated across different paths. However, the solution is still not completely integral, as the flows, although unsplittable, may still be fractional. Furthermore, the author himself states that even this partial integer optimization is only feasible from a practical point of view for "problems of small size".

However, none of these works solve the actual integral max-flow problem. It might be argued that the fractional solution can be made to approximate the integral solution by using a combination of segmentation (breaking the message into smaller pieces) and *dynamic* (sprayed) routing when the messages are large enough to be subdivided into a sufficient number of smaller units. However, such a solution will still only be an approximation of the optimal routing and it will require additional hardware support (such as larger routing tables and re-sequencing queues) and introduce new problems (such as out of order arrivals and segmentation overheads). Furthermore, when a *contention-free* routing solution exists for a certain topology and traffic pattern, an approximate routing solution that has even a single unnecessary conflict will incur a noticeable performance degradation with respect to the optimum, as the communications competing for the same link will share the bandwidth of that link [92, 93]. These contention-caused delays will propagate throughout the entire execution, because subsequent transfers, especially in HPC applications, may depend—directly or through a chain of dependencies—on a given delayed transfer. This causes increased jitter, a well-known cause of severe performance degradation [41, 78].

Other works do attempt to solve the ILP problem, but they generally conclude that the approach is not feasible for practical network sizes. A notable work that falls in this category and provides an ILP formulation specifically for multi-stage networks is due to Elmallah [25]. The paper shows that, although a polynomial-time algorithm can be found for multi-stage networks having up to three levels, the problem is NP-complete for networks with more than three levels. Hence, no attempts were made to find ways of making the optimization process practically feasible.

Other works have steered away from the NP-complete integer linear programming and have concentrated on trying to find optimal routes with polynomial-time algorithms for specific networks and/or traffic patterns. For example, in the context of Clos networks there exist optimal simple routing algorithms when very specific conditions are met [44]. However, no such results have been obtained for extended generalized fat trees.

The method we propose is radically different from previous approaches in that:

- the optimum is computed outside of the linear optimization process and the optimality is embedded in the feasibility conditions (LP constraints), such that the ILP solver serves only as a feasible solution identifier as all feasible solutions are optimal;
- the optimal routes are not computed for the entire network at once; instead, the problem is split up into smaller sub-problems that can be solved efficiently using ILP, and the

optimal local solutions are subsequently combined using dynamic programming into an optimal global solution.

Finally, this is to our knowledge the first work presenting a method to solve the integral optimal routing problem for arbitrary workloads that can claim practically feasible completion times for XGFT networks with thousands of nodes and up to 6 levels.

3.3 Extended generalized fat trees

As we have shown in the motivation section, fat trees are one of the most popular indirect network topologies used in the design of current HPC systems as well as datacenter networks.

The fat tree family encompasses a broad class of different interconnection layouts which can all be described as multi-stage tree-like topologies where the bandwidth of the links may increase towards the root of the tree.

An *ideal fat tree* is a k-ary tree interconnection network where the bandwidth of each link towards the root increases in powers of k. The CM-5 [60] was the first machine to implement a variant of the ideal fat tree network. More generically, a *full-bisection bandwidth* fat tree is a tree interconnection network in which every link can always accommodate the aggregate capacity of the sub-tree connected to that link. Such fat tree networks are not realizable for large clusters, as the bandwidth needs to increase exponentially at each layer towards the root, which would require switches with either exponentially increasing number of ports or exponentially increasing bandwidth per link.

However, it is possible to construct a tree-like network that exhibits an equivalent full bisection bandwidth property and uses fixed-capacity switches, i.e., does not require increasing either the number of ports per switch or the port bandwidth [79, 80].

Furthermore, it is possible to construct networks in which the bandwidth towards the root can be tuned to almost arbitrary values, from the full-bisection bandwidth of ideal fat trees to reduced bandwidth realizations that have a correspondingly reduced cost. These are called *Extended Generalized Fat Trees* or *XGFTs*. Öhring et al. [76] provide a generic framework for compactly describing arbitrary XGFT topologies. In this framework, an XGFT network is completely described by two values for each tree layer, namely an M value, describing the number of direct descendants of a node at a specific layer, and a W value, describing the number of direct ancestors of a node at a specific layer. As such, the network is completely described by a set of parameters of the form XGFT($H : M_1,...,M_H; W_1,...,W_H$), where the parameter H corresponds to the number of switch layers of the tree. An XGFT($H : M_1,...,M_H; W_1,...,W_H$) is conceptually equivalent to a standard fat tree with downward branching factors ($M_1,...,M_H$) and upward bandwidths available to every conceptually equivalent fat tree node at every layer equal to ($W_1, W_1 \cdot W_2, ..., W_1 \cdot ... \cdot W_H$).

3.3.1 A new XGFT representation

First, we introduce an alternative way of describing the XGFT topology that will enable an easier formulation and understanding of the routing optimization approach in Section 5.3. Instead of viewing the XGFT as before as a single, monolithic network layout, we dissociate it into two complementary aspects:

- On the one hand an XGFT is meant to conceptually mirror a regular tree where every node has a single ancestor, with the exception of the unique root, which has no ancestor.
- On the other hand, an XGFT is meant to provide a practical way of ensuring increasing bandwidth towards the root using (relatively) small radix switches. As the number of ports per switch is limited, the large fat-tree-specific bandwidth that is required of upper layer switch-to-switch connections cannot be achieved via a multitude of links. The XGFT design solves this problem by replacing the conceptual exponential-radix fat-tree switch with an exponential (towards the root) multitude of constant-radix switches. We will show that this separation of the conceptual switches can be described by a second tree that is regular but *inverted* (the root layer of this second tree corresponds to the leaf layer of the XGFT).

In the remainder of this section we will show how these two aspects are completely described by a single regular tree each, and how the XGFT itself can be described via this dual view.

Given an XGFT($H : M_1,...,M_H; W_1,...,W_H$) as described above, we consider two simple trees $MT(H : M_1,...,M_H)$ and $WT(H : W_H,...,W_1)$, each with the same number of layers $H+1$ as the original XGFT. For the former, we number the layers starting with 0 from the leaf layer in increments of 1 (Figure 3.1b), whereas for the latter we number the layers starting with 0 from the root layer also in increments of 1 (Figure 3.1c). The parameters M_l of MT represent the number of descendants of each MT node on layer l, whereas the parameters W_l of WT represent the number of descendants of each WT node on layer $l-1$.

The bijective correspondence between the standard XGFT characterization and this new dual characterization is the following. A layer l node in the XGFT is equivalent to a pair of nodes, one at layer l of MT and one at layer l of WT. The MT node is reached from the unique root of the MT tree by choosing the same links moving towards lower index layers as the links that need to be chosen to reach the XGFT node when starting on any root ancestor of that node. Similarly, the WT node is reached from the unique root of the WT tree by choosing the same links moving towards higher index layers as the links that need to be chosen to reach the XGFT node when starting on any leaf descendant of that node.

Similarly to how every XGFT node on layer l is assigned a set of identifying coordinates $(l : m_H, ..., m_{l+1}, w_l, ..., w_1)$, with $0 \le w_i < W_i$ and $0 \le m_i < M_i$, we assign to every node on layer l in the trees MT and WT a set of coordinates $(l : m'_H,...,m'_{l+1})$ and $(l : w'_1,...,w'_l)$ respectively, by simply making a list of the branching choices required starting on the root of the tree

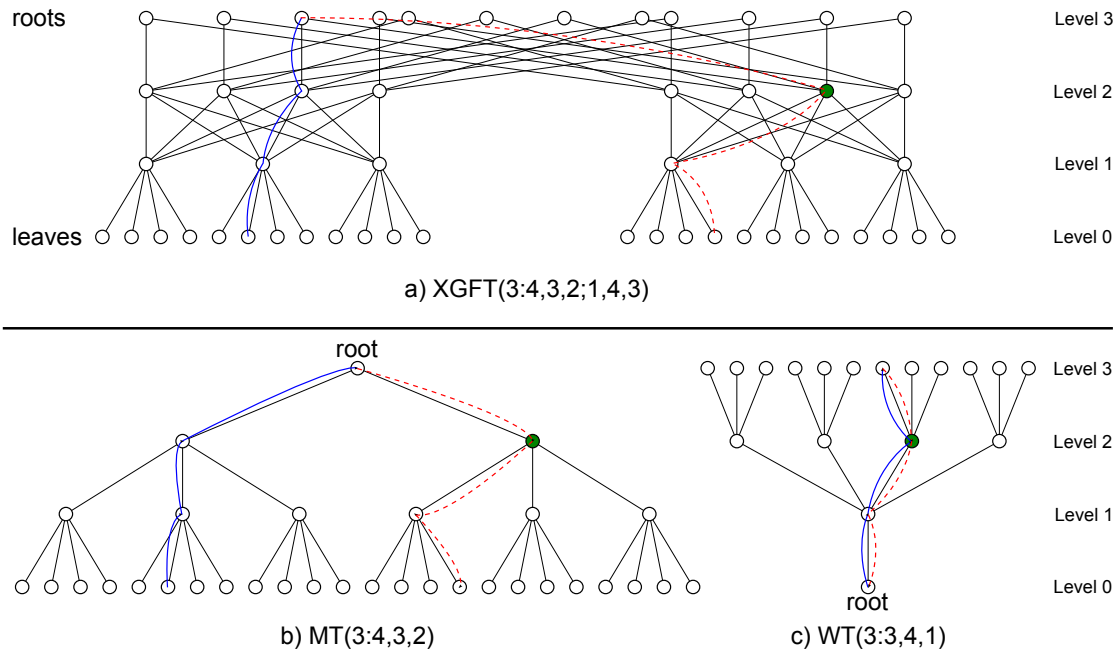


Figure 3.1 – Standard (Figure a) and dual (Figures b and c) representation for a 3-layer network: XGFT(3:4,3,2;1,4,3). The dual representation is composed of two simple trees, the MT tree (Figure b) and the WT tree (Figure c). The layers of the WT tree are numbered in the reverse order of that of the XGFT and MT tree. The MT tree captures the scalability properties of the fat tree that the XGFT is conceptually modeling, i.e., it captures the downward branching factors of the fat tree, with no information on the bandwidth available at each layer. The WT tree expresses how many links each node has available to the next layer (in numbering order), which equals the number of upward links for a given node in the original XGFT. Every XGFT node on layer l, n^l, corresponds to a unique pair of nodes, one in the MT tree and one in the WT tree, also at layer l in their corresponding trees: (mt^l, wt^l). The filled nodes exemplify such a correspondence. The figures also show an example path between a source-destination pair as they are reflected by both representations. The upward path is shown in a continuous line (in blue), whereas the downward path is shown dashed (in red). In the MT tree, the path between a fixed source and destination is unique. In the WT tree, the upward path and the downward path are identical, but any path connecting the root to any node on the layer where the downward path begins is a valid one.
Figure included from [87]

to get to the node in question. Then formally, the pair of (MT,WT) nodes corresponding to an XGFT node with coordinates $(l : m_H, ..., m_{l+1}, w_l, ..., w_1)$ are the nodes with coordinates $(l : m_H, ..., m_{l+1})$ and $(l : w_1, ..., w_l)$ in their respective MT and WT trees.

Figure 3.1 illustrates the correspondence via the nodes that are filled. The layer 2 XGFT node that is filled corresponds to two filled nodes in the dual representation. To reach the XGFT node starting from the root layer, one would choose the rightmost link of any ancestor root, while to reach it starting from the leaf layer, one would choose first the only link available, then the second rightmost link. This choice is unique and allows the determination of the MT and WT corresponding nodes.

A link in the XGFT is represented in the dual representation via a pair of links, one belonging to MT and one belonging to WT, which are the exact links that connect the pairs of MT and WT nodes that form the dual representations of the end points of the original link in the XGFT.

Since the bijection applies to links as well as nodes, it also applies to the paths taken by messages in the XGFT. Routing can thus be seen as a synchronized movement in the two trees of the dual representation, between the MT components of the source and destination on one hand and between the WT components of the source and destination on the other. An example path in both representations is shown in Figure 3.1, with the upward portion of the path being shown as a continuous blue line and the downward portion as a dashed red line.

The dual representation clearly exposes the XGFT routing-related properties. Specifically, whereas in the MT tree there exists a unique route between any given source and destination, routing in the WT tree is equivalent to starting from the root, descending into the tree for a number of layers, then returning to the root. Thus, it is obvious that when moving away from the root of WT (equivalent to moving "up" in the XGFT), the number of possible ways we have to descend into the WT tree is equal to $W_1 \cdot W_2 \cdot ... \cdot W_l$, where l is the maximum descent layer, while when moving back towards WT's root (equivalent to moving "down" in the XGFT), there are no choices to be made, i.e., the path is completely determined by the choices made while descending. This property of the downward path completely being determined by the upward path is one of multiple intrinsic properties of XGFT routing which become immediately obvious in the dual representation, but are not exposed explicitly by the standard characterization.

Another example of such a property, which we will use in our approach and which, with the standard XGFT model, can require significant effort to prove [24] while being obvious in the dual representation, is the following. If during the upward part of an up-down path, on a node on layer l, the choice was made to route on the upwards link w_l, then on the downward path the message will enter the node on layer l on link w_l as well. In the dual representation, because the upward link choice is determined solely by the WT tree trajectory, and because in that tree we are taking the same path first moving away from the root and then in reverse, it is immediate that the same links will be used.

Summarizing, we have the following properties:

Property 1. *In an XGFT, given a message that moves from its source upwards into the tree up to layer l then descends to its destination, the total number of available paths that message can choose equals $W_1 \cdot W_2 \cdot ... \cdot W_l$.*

Property 2. *In an XGFT, given a message that moves from its source upwards into the tree up to layer l then descends to its destination, the message having already performed the upward portion of its path, then the downward path of the message is unique (i.e., completely determined).*

Property 3. *In an XGFT, given a message from a source leaf node to a destination leaf node, the message having advanced from layer l to layer $l+1$ using the w-th upwards link from the node it was on at layer l, then on its downward path, the message will descend from layer $l+1$ to layer l using, in reverse direction, the w-th upwards link on the node it will reach on layer l.*

3.4 Linear programming optimized routing

The exact problem we are solving is the following. We are considering a fixed communication load that needs to be delivered using the network as a medium. In the most general case, this means that every source node S needs to send a message to every destination in a set (with repetition) D_S. The set D_S can be empty or can contain the same destination multiple times, the latter case corresponding to the situation where source S sends multiple messages to a given destination.

Messages traveling through the network can use a variety of paths. Given an assignment of a single path to each message in a workload, we define the contention level induced by the workload on an arbitrary link as the total number of assigned paths that contain that link. The problem we aim to solve is finding a path-to-message assignment that ensures that the maximum contention level across all links in the network is minimized.

This is a problem that has relevance regardless of the network type. However, for arbitrary workloads and arbitrary networks, finding such an optimum assignment is typically not possible via a heuristic approach, making it typically necessary to perform an exhaustive search of the assignment space, which is large enough to make such an approach impractical. Furthermore, even for specific classes of networks but arbitrary workloads, exhaustive search might ultimately still be necessary to obtain an optimal solution.

Nonetheless, in the latter case, an optimized exhaustive search that takes advantage of the structure of the network has potential to yield a solution in a practical time.

In the remainder of this section, we will present an approach to achieve just that in the case of arbitrary XGFT networks. We will use an integer linear programming solver as the means to perform the search (thus benefiting from the large amount of optimizations done in this field) and provide it with a specification of the routing problem that ensures rapid convergence.

3.4.1 Layer-by-layer route optimization

The main problem in performing an exhaustive search on the path-to-message assignment space is that the space in question is extremely large. Indeed, given a workload composed of N messages and a path diversity for any given source-destination pair of P, then the total number of possible assignments is P^N.

In an XGFT with the specification $XGFT(H : M_1,...,M_H; W_1,...,W_H)$, we have that $P_l = W_1 \cdot ... \cdot W_l$ for messages needing to go up in the tree up to layer l before descending back to the destination. As an example, in a small 1024-node XGFT ($XGFT(4 : 16,8,4,2; 1,16,8,4)$) where every node needs to send a single message to another node across the root (which is a case that is typical in practice, for example in many of the phases of an all-to-all exchange), the size of the search space is $512^{1024} \approx 10^{2700}$.

For this reason, our approach divides the network into smaller optimization domains and propagates solutions from one domain to the next, by using a dynamic programming approach. This reduces the assignment space that needs to be explored, which in turn reduces search time. However, to allow the method to work correctly, as with any dynamic programming approach, we must ensure that any solution that is found for a sub-domain and is propagated as a starting point for the next sub-domain is a partial solution of an optimum total solution.

In the case of the XGFT network, a natural way of partitioning the topology is on a per-layer basis. Specifically, messages will first be assigned routing decisions that allow them to move from layer 0 to layer 1 of the tree. Then, all messages that need to go up past layer 1 will be propagated in the network according to the previously chosen assignment and a new assignment will be chosen that allows them to go from layer 1 to layer 2, and so on up to the root.

With such an approach, the total search space is reduced to a succession of per-layer search spaces each of a size of approximately W_{l+1}^N, where N is the number of messages needing to go up past layer l and W_{l+1} is the XGFT W parameter at layer l. Going back to our previous example, this reduces the search space to a maximum size of $16^{1024} \approx 10^{1200}$, significantly smaller than for the original approach. The difficulty, once more, lies in ensuring the global optimality of the union of sub-solutions.

3.4.2 Single-layer problem formulation

The problem we want to solve is the following. Given a workload as described above, a layer l of the XGFT and a set of routing assignments for the messages in the workload for layers 0 through $l-1$, we want to generate a set of routing assignments for layer l.

The first thing to notice is that we can use the lower layer routing assignments to generate a "position" in the network for every message reaching layer l. Thus, every node at layer l will be assigned a set of messages, some of which need to be routed downwards, others routed

upwards. Property 2 ensures that messages that are routed downwards have their entire path already decided. Therefore, routing assignments need only be generated for messages routed upwards.

The output of our algorithm must thus be, for every message that is to be routed upwards of layer l, a choice among the W_{l+1} upward links available to that message on the node it is on. This choice needs to:

1. fulfill a local optimality criterion for the links going upwards at the current layer;

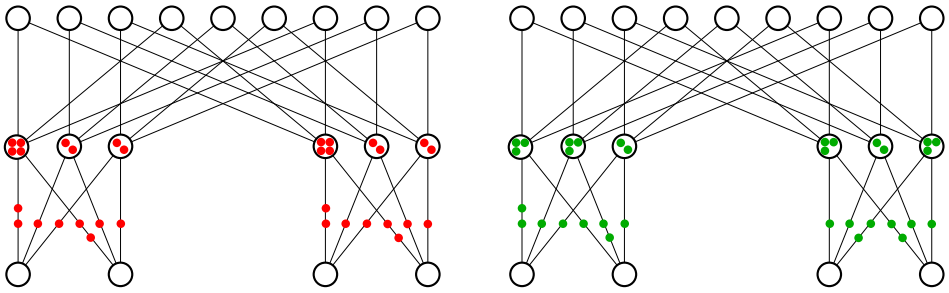
2. ensure that the locally optimal solution is part of a globally optimal solution;

3. since the choices made going upwards also permanently determine the downward path and since we know what links those messages will use going down, at the same layer (according to Properties 2 and 3) the current choice must also ensure Properties 1 and 2 for the downward path.

The latter property actually requires that two optimization criteria be pursued simultaneously (one for messages on their upward path and the other for the same messages on their downward path) and this is actually what makes the optimization process very difficult to achieve by other means than an exhaustive search.

3.4.3 Upward local optimality criterion

Given a set of n messages present on a given node of the tree at layer l that need to move to layer $l+1$ and given W_{l+1} choices in terms of links those messages can move on, the minimum achievable contention level across the links is clearly $\lceil n/W_{l+1} \rceil$. To ensure not only a minimum maximum contention level across links but also a complete load balancing of traffic, we also need to impose a lower limit on the number of messages routed on any given link of $\lfloor n/W_{l+1} \rfloor$. To understand why this is necessary, let us take the example of $n = 7$ messages needing to be distributed across $W = 3$ links. The minimum achievable contention level across the links is $\lceil n/W \rceil = 3$. However, without the strict load balancing lower limit, one acceptable assignment would be 3,3,1, that is, having two links assigned 3 messages each and the third link assigned a single message. This would cause the third link to receive much fewer messages than the others. A more balanced assignment would be 3,2,2. In general, a perfect load balancing of messages across links is ensured by any assignment where the difference (in number of messages assigned) between the lowest utilized link and the highest utilized link is at most 1 message. The lower bound mentioned above guarantees this property. Enforcing it is done however at the expense of reducing the density of feasible solutions in the search space. This is the only local optimality criterion.



a) Non-optimal routing b) Optimal routing

Figure 3.2 – Example of two route assignments for an $XGFT(2:2,2;3,3)$ network and a workload consisting of 4 messages, shown as red dots in Figure a) and as green dots in Figure b), sent from every leaf node source to leaf node destinations across the root. Both assignments are optimal for the lowest layer of links, however the assignment in Figure a) is non-optimal for the second layer (the leftmost second layer node for example has 4 messages to distribute among 3 links) while the assignment in Figure b) is optimal.
Figure included from [87]

3.4.4 Global optimality preserving criteria

First, let us illustrate through an example why the local optimality conditions are insufficient, in that a locally optimal partial solution to the route assignment problem can prove to not constitute a part of a globally optimal solution. Let's take the $XGFT(2:2,2;3,3)$ illustrated in Figure 3.2 and let's assume a workload where every leaf node needs to send 4 messages across the root. The locally optimal upper bound for the number of messages that cross each upwards link at layer 0 is $\lceil n/W_1 \rceil$ which, given that $n = 4$ and $W_1 = 3$ is 2. One solution (illustrated in Figure 3.2a) that is locally optimal at layer 0 is to assign, on every node, 2 of the 4 messages to link 0 and 1 message to each of the other two upward links. Propagating the messages to layer 1, this would lead to 4 of the 6 layer 1 nodes receiving 2 messages while the remaining 2 nodes at that layer would receive 4 messages each. Given that the number of upward links at layer 1 is $W_2 = 3$, this leads to a contention level of at least 2 (the latter two nodes have to distribute 4 messages across 3 links). However, there exists a different, optimal route assignment (illustrated in Figure 3.2b) that induces a contention level on layer 1 upward links of only 1 (this can be achieved for example by having leaf node i, where nodes are numbered consecutively from left to right, assign 2 messages to upward link $i \mod W_1$ and 1 message to each of the remaining 2 upwards links). We have thus shown that the local optimality criterion is insufficient.

Let us now derive in the general case the optimality preserving criteria for the global solution. Given two messages at layer l, on different XGFT nodes, four conditions need to be fulfilled in order for those two messages to contend for a link at layer $l + \lambda$:

1. the messages need to start their downstream portion of the path at a layer strictly larger than $l + \lambda$

2. the messages need to eventually propagate to the same node in the MT tree at layer $l+\lambda$, which is equivalent to them being on layer l nodes in the MT tree that have a common ancestor at layer $l+\lambda$

3. the messages need to eventually propagate on the same node in the WT tree at layer $l+\lambda$, which is equivalent to them being on the same layer l node in the WT tree and them picking exactly the same w-s at layers l through $l+\lambda-1$

4. the messages need to pick the same link $w_{l+\lambda+1}$ at layer $l+\lambda$

However, we do not know the routing assignments of the messages at layers above l. Hence, we must ensure that in the case of an optimum assignment for layers $l+1$ through $l+\lambda$, our current layer l assignment still allows for optimality for layer $l+\lambda$. The messages at layer l that, with the knowledge we currently have, can potentially contend on layer $l+\lambda$ are those messages that descend at a layer strictly greater than $l+\lambda$, that have a common ancestor in the MT tree at layer $l+\lambda$, and that are on the same WT tree node at layer l. Let's denote, for every pair $(mt^{l+\lambda}, wt^l)$ of an MT tree ancestor at layer $l+\lambda$ and a WT tree node at layer l, the number of messages that can potentially contend at layer $l+\lambda$ as being $c^u_{\lambda,mt^{l+\lambda},wt^l}$ (the "u" standing for "upwards").

An assignment at layer l separates these messages into W_{l+1} different groups, where contention can only appear within each individual group (i.e., among messages for which the routing assignment allocated the same upward link index). The local restriction ensures only that the number of messages that have potential to contend post-assignment (i.e., the number of messages in every group) is upper bounded by $\sum_{\{mt^l|\text{ancestor}(mt^l)=mt^{l+\lambda}\}} \lceil c^u_{0,mt^l,wt^l}/W_{l+1} \rceil$. The optimal assignment however is one where the $c^u_{\lambda,mt^{l+\lambda},wt^l}$ messages that will reach past layer $l+\lambda$ are assigned evenly between groups. The condition that enforces the optimum assignment is thus that the number of messages in a group be upper bounded by $\lceil c^u_{\lambda,mt^{l+\lambda},wt^l}/W_{l+1} \rceil$.

Once more, we can additionally impose a load balancing restriction by lower bounding the same number of messages by $\lfloor c^u_{\lambda,mt^{l+\lambda},wt^l}/W_{l+1} \rfloor$. These optimality preserving restrictions need to be enforced for every λ up to the root and for each given λ, for every $mt^{l+\lambda}$.

3.4.5 Improving search convergence

Strong restrictions, such as the ones above, allow in principle a more aggressive pruning of invalid partial solutions during the optimization process. However, in practice we have found that past a certain point they will also diminish the density of feasible solutions in the restriction of the search space they define, making the overall completion time of the search higher.

Hence, there is potential to improve the completion time by providing slightly weaker restrictions, that are nonetheless optimality preserving (they ensure the same maximum contention level) at the expense of inducing a load across links that might be less than ideally balanced.

The local criterion is not only sufficient but also necessary, so no restriction relaxation is possible in its case. However, weaker conditions can be formulated when ensuring global optimality.

The strong restriction states that the size of a group of messages choosing the same upward link must be upper bounded by $\lceil c^u_{\lambda,mt^{l+\lambda},wt^l}/W_{l+1}\rceil$. This is equivalent to ensuring that messages that have potential to contend at layer $l+\lambda$ will be split as evenly as possible across the W_{l+1} upward links available at layer l. Looking at the WT tree, one can clearly see that once having reached layer $l+1$, each of these groups of messages will be split further, again evenly (due to the local optimality criterion that will be enforced in the next layer), into W_{l+2} groups, and so on up to layer $l+\lambda$. So all in all, the set of messages in one of the original layer l groups will have potential to be split into $W_{l+2}\cdot\ldots\cdot W_{l+\lambda+1}$ groups until contending at layer $l+\lambda$. Therefore, if that group contained n messages, the contention level those messages will induce at layer $l+\lambda$ will be $\lceil n/(W_{l+2}\cdot\ldots\cdot W_{l+\lambda+1})\rceil$.

The strong restriction imposes an upper limit on what n can be. However, we can see that increasing n from that limit to the first multiple of $(W_{l+2}\cdot\ldots\cdot W_{l+\lambda+1})$ actually has no influence on the contention level induced at layer $l+\lambda$.

Consequently, we can replace the strict upper bound derived above by $\lceil c^u_{\lambda,mt^{l+\lambda},wt^l}/W_{l+1}/(W_{l+2}\cdot\ldots\cdot W_{l+\lambda+1})\rceil\cdot(W_{l+2}\cdot\ldots\cdot W_{l+\lambda+1})$. We will call this acceleration technique *restriction relaxation*.

A second method to improve the completion time of the optimized exhaustive search is to remove superfluous restrictions. We mentioned above that the local restriction ensures by itself an upper bound of $\sum_{\{mt^l|\text{ancestor}(mt^l)=mt^{l+\lambda}\}}\lceil c^u_{0,mt^l,wt^l}/W_{l+1}\rceil$ for the number of messages in a group. If this limit is lower than or equal to either the strong upper bound or the upper bound obtained through restriction relaxation, then we can eliminate the strong/relaxed restriction altogether. We will call this acceleration technique *restriction elimination*.

The applicability of these techniques is dependent on both the structure of the topology and that of the workload so it might not always be possible to accelerate the search using these approaches. However, when it is possible, the resulting routing assignment ensures an optimal maximum contention level across links, at the possible expense of having less-than-perfect balancing of load across links.

3.4.6 Downward path restrictions

Until now we have only discussed about enforcing the load induced by messages moving towards the root of the tree. However, the routing decisions taken going upwards completely determine the downward path as well due to Properties 2 and 3). Ensuring an optimal maximum contention level on the downward links is equivalent to considering them as upwards links in a time-reversed message flow, where messages depart from their destination and finish at their source, and then applying the same restrictions to the exhaustive search as above.

Within the optimization step of a single layer, both the downward and upward conditions must be enforced simultaneously to ensure an overall optimum maximum link contention level.

The remainder of this section formalizes the approach described thus far.

3.4.7 Formal description of the optimization constraints

A message $m(id,s,d,\delta,l,mt_u,wt_u,mt_d,wt_d,w_{l+1}^{id})$ at layer l is characterized by:

- a unique id id
- its source node index s
- its destination node index d
- the layer at which is starts its downward path δ
- the current layer l
- the MT node and WT node indices at layer l on the upward path mt_u and wt_u
- the MT node and WT node indices at layer l on the downward path mt_d and wt_d
- the choice at layer l w_{l+1}^{id}

The first 4 fields are computed at the beginning and remain unchanged for the duration of the routing assignment generation across layers. The rest of the fields are initialized and updated as follows.

Initialization:

- $mt_u = s$
- $wt_u = 0$
- $mt_d = d$
- $wt_d = 0$
- $l = 0$

Update when moving to the next optimization layer, for messages that need to move upwards at that layer:

- $mt_u = ancestor(mt_u)$

- $wt_u = descendant(wt_u, w_{l+1}^{id})$
- $mt_d = ancestor(mt_d)$
- $wt_d = descendant(wt_d, w_{l+1}^{id})$
- $l = l + 1$

The optimization step at an arbitrary layer l receives as input the set of messages M that have advanced to that layer (each message missing the w_{l+1}^{id} parameter) and needs to output for each message $m(id,...)$ the link choice w_{l+1}^{id}.

We introduce the following notations. For a given l, a given λ, a given node $mt^{l+\lambda}$ in the MT tree at layer $l+\lambda$ and a given node wt^l in the WT tree at layer l, we denote by $M^u_{\lambda,mt^{l+\lambda},wt^l}$, independent of the routing choice at layer l, the set of messages that:

- reached layer l moving upwards,
- begin their downward path at a layer strictly greater than $l+\lambda$,
- have propagated moving upwards to layer l on a node that has node $mt^{l+\lambda}$ as an ancestor in the MT tree,
- and have propagated moving upwards to layer l on node wt^l in the WT tree.

Furthermore, we denote the subset of messages in $M_{\lambda,mt^{l+\lambda},wt^l}$ that have their routing choice equal to w_{l+1} where $0 \leq w_{l+1} < W_{l+1}$, by $M^u_{\lambda,mt^{l+\lambda},wt^l,w_{l+1}}$.

We introduce analogous notations for the downward path: $M^d_{\lambda,mt^{l+\lambda},wt^l}$ and $M^d_{\lambda,mt^{l+\lambda},wt^l,w_{l+1}}$.

Formally:

$$M^u_{\lambda,mt^{l+\lambda},wt^l} = \{m(id,s,d,\delta,l,mt_u,wt_u,mt_d,wt_d,w_{l+1}^{id}) | \\ \delta > (l+\lambda), ancestor^{\lambda}(mt_u) = mt^{l+\lambda}, wt_u = wt^l\} \quad (3.1)$$

$$M^u_{\lambda,mt^{l+\lambda},wt^l,w_{l+1}} = \{m(id,s,d,\delta,l,mt_u,wt_u,mt_d,wt_d,w_{l+1}^{id}) | \\ m \in M^u_{\lambda,mt^{l+\lambda},wt^l}, w_{l+1}^{id} = w_{l+1}\} \quad (3.2)$$

$$M^d_{\lambda,mt^{l+\lambda},wt^l} = \{m(id,s,d,\delta,l,mt_u,wt_u,mt_d,wt_d,w_{l+1}^{id}) | \\ \delta > (l+\lambda), ancestor^{\lambda}(mt_d) = mt^{l+\lambda}, wt_d = wt^l\} \quad (3.3)$$

$$M^{d}_{\lambda,mt^{l+\lambda},wt^{l},w_{l+1}} = \{m(id,s,d,\delta,l,mt_u,wt_u,mt_d,wt_d,w^{id}_{l+1}) | \\ m \in M^{d}_{\lambda,mt^{l+\lambda},wt^{l}}, w^{id}_{l+1} = w_{l+1} \quad (3.4)$$

With these notations, we then have that:

$$c^{u}_{\lambda,mt^{l+\lambda},wt^{l}} = |M^{u}_{\lambda,mt^{l+\lambda},wt^{l}}|. \quad (3.5)$$

and the strong condition for the upward path then is:

$$\forall \lambda \geq 0, \forall mt^{l+\lambda}, \forall wt^{l}, \forall 0 \leq w_{l+1} < W_{l+1}, |M^{u}_{\lambda,mt^{l+\lambda},wt^{l},w_{l+1}}| \leq \lceil c^{u}_{\lambda,mt^{l+\lambda},wt^{l}} / W_{l+1} \rceil, \quad (3.6)$$

where the local optimality criterion corresponds to $\lambda = 0$ and the global optimality conditions correspond to $\lambda > 0$.

Similarly, given the notation:

$$c^{d}_{\lambda,mt^{l+\lambda},wt^{l}} = |M^{d}_{\lambda,mt^{l+\lambda},wt^{l}}|, \quad (3.7)$$

the strong condition for the downward path is:

$$\forall \lambda \geq 0, \forall mt^{l+\lambda}, \forall wt^{l}, \forall 0 \leq w_{l+1} < W_{l+1}, |M^{d}_{\lambda,mt^{l+\lambda},wt^{l},w_{l+1}}| \leq \lceil c^{d}_{\lambda,mt^{l+\lambda},wt^{l}} / W_{l+1} \rceil \quad (3.8)$$

Should we wish to enforce strict load balancing, two more conditions need to be added: one for the upward path:

$$\forall \lambda \geq 0, \forall mt^{l+\lambda}, \forall wt^{l}, \forall 0 \leq w_{l+1} < W_{l+1}, |M^{u}_{\lambda,mt^{l+\lambda},wt^{l},w_{l+1}}| \geq \lfloor c^{u}_{\lambda,mt^{l+\lambda},wt^{l}} / W_{l+1} \rfloor, \quad (3.9)$$

and one for the downward path:

$$\forall \lambda \geq 0, \forall mt^{l+\lambda}, \forall wt^{l}, \forall 0 \leq w_{l+1} < W_{l+1}, |M^{d}_{\lambda,mt^{l+\lambda},wt^{l},w_{l+1}}| \geq \lfloor c^{d}_{\lambda,mt^{l+\lambda},wt^{l}} / W_{l+1} \rfloor \quad (3.10)$$

Should we wish to enforce the relaxed restrictions, the conditions change as follows. For the upward path:

$$\forall \lambda \geq 0, \forall mt^{l+\lambda}, \forall wt^{l}, \forall 0 \leq w_{l+1} < W_{l+1},$$
$$|M^{u}_{\lambda, mt^{l+\lambda}, wt^{l}, w_{l+1}}| \leq \lceil c^{u}_{\lambda, mt^{l+\lambda}, wt^{l}} / W_{l+1} / (W_{l+2} \cdot ... \cdot W_{l+\lambda+1}) \rceil \cdot (W_{l+2} \cdot ... \cdot W_{l+\lambda+1}), \quad (3.11)$$

and for the downward path:

$$\forall \lambda \geq 0, \forall mt^{l+\lambda}, \forall wt^{l}, \forall 0 \leq w_{l+1} < W_{l+1},$$
$$|M^{d}_{\lambda, mt^{l+\lambda}, wt^{l}, w_{l+1}}| \leq \lceil c^{d}_{\lambda, mt^{l+\lambda}, wt^{l}} / W_{l+1} / (W_{l+2} \cdot ... \cdot W_{l+\lambda+1}) \rceil \cdot (W_{l+2} \cdot ... \cdot W_{l+\lambda+1}), \quad (3.12)$$

3.4.8 Routing as a linear programming problem

As explained earlier in this section, we model the routing problem as an exhaustive search in the routes-to-messages assignment space. The size of the space we need to explore is immense and as such an optimized approach that is able to prune infeasible partial solutions very early is critical. In order to benefit from existing work, we model the routing problem as a linear optimization problem with binary variables and use an independently optimized linear programming solver to produce the solution.

To ensure as aggressive a pruning of the search space as possible, our method embeds the optimality of the solution not in the optimization function of the linear programming specification but in the feasibility constraints themselves. This way, any feasible solution is an optimum solution and any non-optimum solution is eliminated implicitly.

The problem formulation for layer l uses multiple binary variables named using the form $var_{l,id,w_{l+1}}$. If the solution contains a value of 1 for $var_{l,id,w_{l+1}}$, it signifies that the message with the identifier id should be routed on the upward link with index w_{l+1} at layer l. For a given linear optimization problem, l is fixed, id is the identifier of any message with $\delta > l$ and w_{l+1} takes values between 0 and $W_{l+1} - 1$.

In order to express the restrictions defined above as linear programming constraints, it is sufficient to express the values of

- $M^{u}_{\lambda, mt^{l+\lambda}, wt^{l}}$
- $M^{d}_{\lambda, mt^{l+\lambda}, wt^{l}}$
- $|M^{u}_{\lambda, mt^{l+\lambda}, wt^{l}, w_{l+1}}|$

- $|M^{d}_{\lambda,mt^{l+\lambda},wt^{l},w_{l+1}}|$

in terms of the variables introduced above.

The former two are not dependent on the optimization process and their exact values can be precomputed in a straightforward fashion from the distribution of messages across nodes at layer l, which is an input to the optimization process. Once these sets are established, the number of elements in each set is also readily available. The latter two can be expressed as:

$$|M^{u}_{\lambda,mt^{l+\lambda},wt^{l},w_{l+1}}| = \sum_{m(id,\ldots)\in M^{u}_{\lambda,mt^{l+\lambda},wt^{l}}} var_{l,id,w_{l+1}} \tag{3.13}$$

$$|M^{d}_{\lambda,mt^{l+\lambda},wt^{l},w_{l+1}}| = \sum_{m(id,\ldots)\in M^{d}_{\lambda,mt^{l+\lambda},wt^{l}}} var_{l,id,w_{l+1}} \tag{3.14}$$

This allows us to express all restrictions previously mentioned as an ILP problem. However, none of these restrictions actually enforce a property that is not related to the optimization but is intrinsic to routing itself: a message cannot be routed on more or less than one link at any given layer. This is equivalent to:

$$\forall id, \sum_{0\le w_{l+1}<W_{l+1}} var_{l,id,w_{l+1}} = 1 \tag{3.15}$$

Finally, as the optimality of the solution is embedded in the feasibility constraints, no optimization function is provided as part of the ILP formulation.

3.5 Practical relevance

In this section, we delve into some of the practical aspects and relevance of our approach. In the first part, we describe the exact steps that need to be undertaken to effectively use the method in practice. In the second part, we present how we successfully applied our approach to obtain optimal routes for an actual, relevant and challenging HPC workload that was provably optimum for the target topology, but for which existing state-of-the-art routing strategies for fat tree networks were not able to sustain the optimal level of performance. We demonstrate that in conjunction with our proposed pattern-aware route optimization, the workload was able to achieve the expected levels of performance and overall system performance was improved beyond the state of the art.

3.5.1 Usage of the method

The method is designed to be used in practice as follows: To achieve maximum performance, the routing optimization algorithm needs to be made aware of the exact network configuration that is targeted as well as of the workload or mix of workloads to be optimized for. While the structure of the network is typically fixed and known before-hand, a workload characterization step might be necessary to extract the patterns in the communication steps of the targeted applications. Alternatively, synthetic traffic patterns that are known to approximate typical application mixes can also be used.

The workload and network specification are then utilized to generate the routing decisions, using an offline iterative process with as many steps as layers there are in the network, as follows. For each layer of the network, starting with the lowest, an ILP formulation is generated according to the procedure detailed in Section 5.3. The resulting problem specification is supplied as input to an ILP solver (in our benchmarks, detailed in Section 3.6, we used the solver included in the freely available open-source framework GLPK [63]). The solution produced by the solver is translated into a routing assignment for the current layer using once more the procedure detailed in Section 5.3. The routing assignment is then used to generate the workload as it propagates to the next layer and we move on to the next iteration.

The end result of this process is the complete set of routing decisions. This set is then transferred to the network, typically via programming of the routing tables.

3.5.2 Practical use case

Number of leaf nodes	Topology specification
16	$XGFT(2:8,2:1,4)$
32	$XGFT(3:4,4,2:1,4,2)$
64	$XGFT(3:8,4,2:1,8,2)$
128	$XGFT(3:8,8,2:1,8,4)$
256	$XGFT(4:8,4,4,2:1,8,4,2)$
512	$XGFT(4:8,8,4,2:1,8,8,2)$
1,024	$XGFT(4:16,8,4,2:1,16,8,2)$
2,048	$XGFT(5:8,8,4,4,2:1,8,8,4,2)$
4,096	$XGFT(5:8,8,8,4,2:1,8,8,8,2)$
8,192	$XGFT(5:8,8,8,8,2:1,8,8,8,4)$

Table 3.1 – Benchmarked XGFT configurations. Each topology is specified using the the $XGFT(H:M_1,M_2,...,M_H:W_1,W_1,W_H)$ notation introduced by [76]. $H+1$ is the number of network layers, the list of M parameters defines the number of descendants a node has at every layer, while the list of W parameters characterizes the bandwidth available at each layer in the tree and is equal to the number of ancestors a node has on every layer.
Table included from [87]

The need for a generic optimal routes generator emerged in the process of developing and

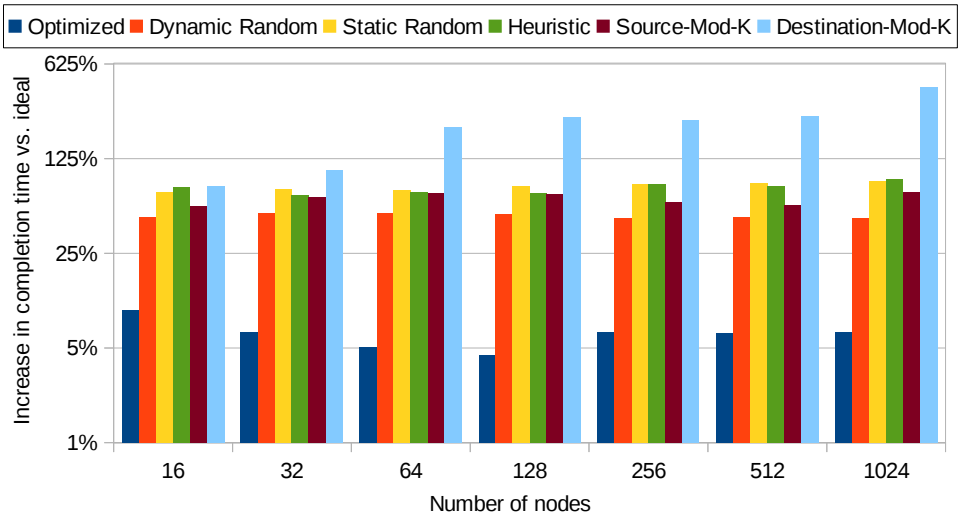


Figure 3.3 – Completion time comparison for the optimized all-to-all communication pattern between the optimized routing algorithm and five other routing approaches commonly used in XGFTs for varying networks sizes under fixed 4 KB message size. The scale for the y axis is logarithmic.
Figure included from [87]

evaluating theoretically optimal communication patterns for all-to-all message exchanges over slimmed fat tree networks.

The all-to-all message exchange is a communication pattern performed by concurrent tasks where every task needs to exchange a single distinct message with every other task. Due to the large amount of traffic it generates (the number of messages scales quadratically with the number of communicating tasks), it is one of the most challenging communication patterns to optimize and scale. The fact that it is the core message exchange in many relevant applications [101] makes its optimization all the more important. Indeed, all-to-all is the core message exchange in two of the NAS Parallel Benchmarks kernels [12]: FT (FFT) and IS (Integer Sort), but more notably required in several physics simulations working in the spectral domain requiring distributed transpose operations, such as in GYRO (time-dependent, non linear Gyro-Kinetic-Maxwell (GKM) equations solver) [5], as well as full physics simulations of Global Atmospheric Models using a Double Fourier Series (DFS) dynamical core [77], which while increasing precision and efficiency with respect to the state-of-the-art, still "suffers from the inherent scalability problem of the spectral model; namely, the transpose operation (or all-to-all communication) required for transforming wave space to grid space or vice versa" [77].

In Chapter 2 we addressed all-to-all optimization in the context of fat tree networks. We considered a case where each leaf node of an XGFT network is assigned a single task. Furthermore, we considered the case where messages are not aggregated and where a given task only sends its own messages (it cannot act as a proxy task for another message source). Therefore, the

load-optimization is performed only via the order in which every source chooses its destinations. The basic idea behind the exchange is to spread traffic as evenly as possible across the different subtrees at every layer, such that any given link of the fat tree conceptually modeled by the XGFT will be traversed in any of the phases by a minimum number of messages (if the total number of messages traversing that link in the complete exchange is n, and there are p phases, then in any given phase there are exactly $\lfloor n/p \rfloor$ or $\lceil n/p \rceil$ messages traversing that link).

This resulted in a set of highly intricate per-phase workloads that are provably bandwidth-optimal for the conceptual fat tree. Despite this optimality, when evaluated in practice, the performance of the approach showed significant degradation compared to the optimum level. The cause was the fact that the routing algorithms were not able to sustain the theoretically ideal workload performance.

In Chapter 2 we also derived the minimum amount of bandwidth required at each fat tree layer to ensure contention-free traffic in each exchange phase. The higher the layer, the more potential for bandwidth reduction exists compared to the full bisection design. In an XGFT, the bandwidth is tunable via the W parameters. The topologies that we consider all have exactly the minimum amount of bandwidth at the top layer (the one where the most reduction is possible), whereas every other layer has the full bisection bandwidth. The resulting topologies are XGFTs with as little as 16 and as many as 8192 leaf nodes, with a number of layers between 3 and 6. The exact configurations are detailed in Table 4.1.

Taking the ideal completion time as the baseline, Figure 3.3 shows the temporal overhead induced by several routing approaches compared to that induced by the optimal routing for a network interconnecting 512 nodes that exchange $4KB$ messages. The routing approaches that were used are some of the commonly employed strategies for fat tree networks [30,91,113]:

1. Static random: On the upwards path, the upward link is chosen at random, but the choice is fixed for every message between the same (source,destination) pair.
2. Dynamic random: Similar to the previous approach, but with a choice that changes dynamically for every message.
3. Heuristic routing: An optimized heuristic approach detailed in [93].
4. Source-mod-K [49,76].
5. Destination-mod-K [61,94,114].

In all cases, the optimized routing showed a performance benefit of at least 40% compared to the best performance achieved across all other routing strategies.

Using the approach presented in this chapter, the artificially induced routing bottleneck was eliminated, and the measured performance matched ideal estimates in the case of large

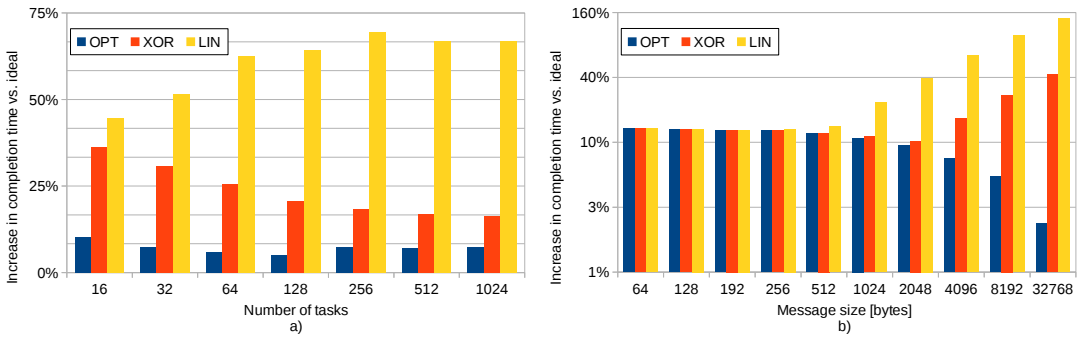


Figure 3.4 – Completion time comparison between the optimized communication pattern (OPT) with optimal routing and two other commonly used patterns – XOR (binary exclusive or) and LIN (linear shift) – for varying networks sizes for fixed 4 KB message size (Fig. a) and for varying message size under fixed 512 nodes network scale (Fig. b). Note the linear scale for the y-axis in a) and the logarithmic scale for the y-axis in b).
Figure included from [87]

message exchanges to within a 3% difference, whereas previous approaches would be between 40% and 140% less efficient. Figure 3.4 shows more detailed results for the different networks shown in Table 4.1 and for different message sizes.

3.6 Algorithm performance evaluation

In benchmarking the efficiency of the approach, we will use the workload and network configurations presented in the previous section. This particular workload is especially well suited for the benchmark for several reasons. First of all, we look at the bandwidth-optimal all-to-all exchange as a set of multiple independent phases which we optimize separately. This gives us the opportunity of assessing the variability in completion time that is to be expected from the route optimization algorithm when dealing with multiple workloads. Second, each individual phase constitutes a particularly difficult to optimize traffic pattern, as showed by the performance that other routing approaches induce, and as such they will test the ability of the route optimization process of finding solutions that are very rare in the search space. Finally, the workloads in this particular set satisfy the conditions necessary to make the application of the acceleration techniques described in section 5.3 possible. As such, we will be able to also measure the impact of the techniques on the completion time of the algorithm.

3.6.1 Benchmarking system description

The benchmarks were conducted using a commodity mid-level system with an Intel® Core™ i7-2600 CPU @ 3.40GHz and 16 GB of RAM. The linear programming solver used is the freely available open-source glpsol solver that is included in the GNU Linear Programming Kit v4.50 [63]. An important limitation of this solver is that it is single-threaded, so although the CPU

we are using has significant potential for parallel acceleration, we do not benefit from it (using a commercial parallel solver will in all likelihood produce the ideal routing in a significantly smaller amount of time). The glpsol solver's behavior has a high degree of tunability via the numerous command line parameters it takes. Among other things, one can tune

- the selection of the branching variable (via the parameters --first, --last, --mostf, --drtom, --pcost)
- the backtracking strategy (via the parameters --dfs, --bfs, --bestp, --bestb)
- the type of heuristic cuts to be applied to the search space (via the parameters --gomory, --mir, --cover, --clique, --cuts)

For a given combination of the above parameters, it might be the case that some workloads on a given topology are a worst case for the solver, in that they take a particularly long time to optimize (much longer than the average completion time across other workloads on the same topology). However, it is often, if not always, the case that changing the parameters, and thus the heuristic of the solver, will lead to different worst case scenarios, such that across a set of parameter combinations the worst cases are eliminated altogether. As such, we analyzed several combinations of the parameters above.

Furthermore, we configured the solver:

- to use the MIP presolver (parameter --intopt),
- to use exact arithmetic (parameter --exact) and
- to consider the integer variables binary (parameter --binarize).

Lastly, we enabled the feasibility pump heuristic (parameter --fpump) which greatly accelerated the finding of feasible solutions, which in our case were also optimal solutions (the optimality is embedded in the feasibility conditions, not in any optimization function).

3.6.2 Results

Before presenting the results for the individual per-phase workloads making up the bandwidth-optimal all-to-all exchange, it is important to specify that the process of optimizing the entire exchange is highly parallel. The serial portion of the process (that can only be further accelerated by a parallel solver) is the single phase optimization. Given a reasonable number of execution threads and/or machines with a small amount of memory each, the determining of the route for the entire exchange can be done in an amount of time in the same order of magnitude as the time necessary to optimize a single phase. A similar rationale stands behind using different combinations of parameters for the solver for those few phases that might

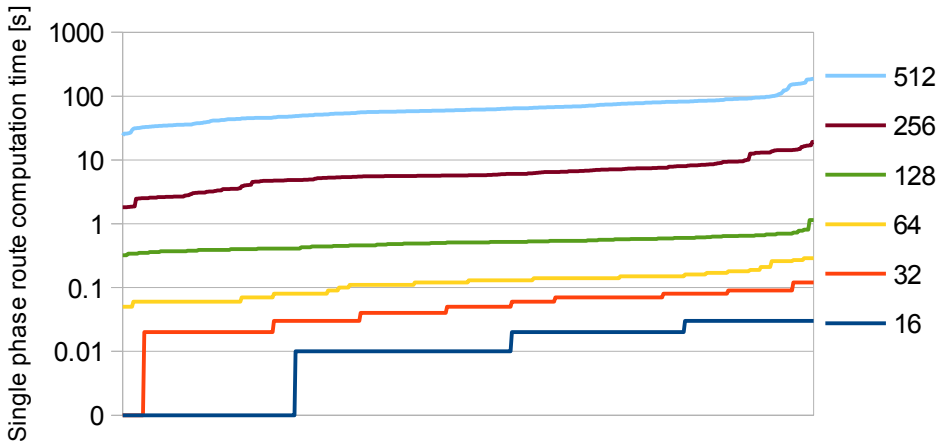


Figure 3.5 – Duration of the route optimization process for a workload constituting of a single phase of the bandwidth-optimized all-to-all exchange. Every line corresponds to a fixed network size (the legend shows the number of communicating nodes corresponding to that network). The completion time of the process was measured for every all-to-all phase then sorted in increasing order and plotted left-to-right to show their distribution. To enable using the same X-axis interval for all network sizes, given that the number of phases is equal to the number of communicating nodes, we use a step-wise constant graph, where the width of each step is constant for a given network size: 1 for the 512 leaf nodes XGFT, 2 for the 256 one, 4 for the 128 one and so on up to 32 for the 16 leaf nodes XGFT. The increase in completion time with the network size is faster than linear and the distribution of the completion time for a fixed network size sees significant variability, with the slowest phase to complete being up to 3 times slower than the average.
Figure included from [87]

be more problematic to optimize. In practice, when the optimization of a workload would exceed a certain amount of time, we would in parallel start optimization processes for the same workload but with different sets of ILP solver parameters and halt the execution of all these parallel processes once a single one of them completes.

We will start with presenting the completion times achieved with the strongest restriction method, which, as we will see, is the slowest to complete and consequently limited in its application to larger size networks. Nonetheless, the strongly restricted approach is able to successfully generate optimal routes for XGFT networks with 1024 leaf nodes in under 30 hours for a single phase of the all-to-all workload. In order to show how the completion time of the algorithm varies with the network size, we ran the optimization process on every one of the phases of the complete all-to-all exchange, for several XGFTs with as few as 16 and as many as 512 leaf nodes. Figure 3.5 shows the (sorted) completion times of the optimization processes across the different phases of the all-to-all exchange. For smaller network sizes we have less phases to benchmark (the number of phases is equal to the number of communicating nodes), so in order to allow for the same X axis interval to be used for all curves, the completion times

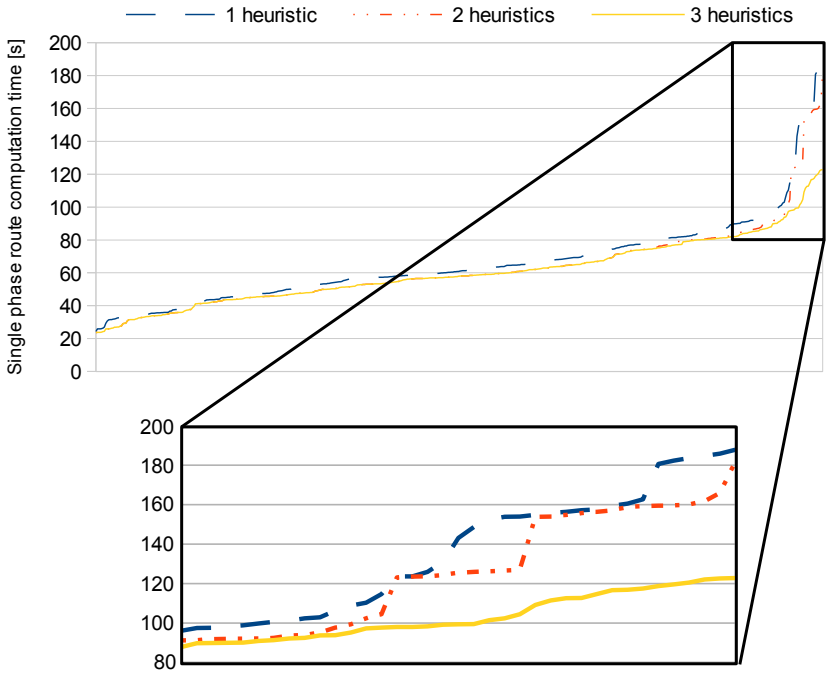


Figure 3.6 – Impact of varying the heuristics of the linear programming solver. The chart on top shows the completion times of the optimization process for every phase of an optimized all-to-all exchange on a 512 leaf nodes XGFT, for one, two and three combinations of solver parameters. When using more than one combination, the completion time of a phase is computed as being the minimum completion time among the different combinations of parameters used. Similarly to before, the X-axis does not represent the phases in their natural order, but instead in increasing order of optimization process completion time, so that the distribution of the completion times becomes apparent. The chart on the bottom zooms in on the tail of the distribution, i.e. the phases that take the most time to optimize. By using only two additional heuristic approaches, we are able to significantly flatten the tail, reducing the ratio between the slowest to complete optimization and the average optimization completion time from 3 to 2 - effectively a 30% reduction in the completion time of the slowest optimization process.
Figure included from [87]

are plotted as step-wise constant graphs, with a step width inversely proportional to the total number of phases. The conclusion is that the completion time of the optimization process scales faster than linear with the network size. As we progressively double the number of leaf nodes in the XGFT, starting with 32, the average completion time for the generation of optimized routes for a single phase increases first 2.5 times, then approximately 4 times, then 10 and then another 10 times. When moving to 1024 (from 512) leaf nodes, the completion time will actually increase 100-fold, due to limitations of the ILP solver.

In the same figure one can observe that, while most workloads (each corresponding to a phase of the all-to-all exchange) have a completion time close to the average, there are a few phases

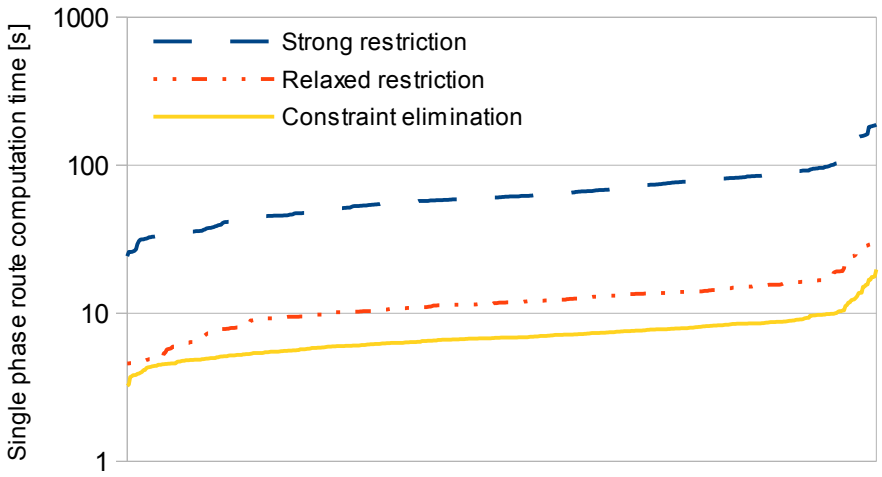


Figure 3.7 – Reduction in completion time of the optimization process achievable when using restriction relaxation (in red) and, in addition, restriction elimination (in yellow) compared to the standard strong restriction formulation (in blue). The benchmarked system is a 512 leaf node XGFT and the curves show the completion time of the optimization process for each of the phases of an optimized all-to-all exchange, ordered from left to right.
Figure included from [87]

for which the exchange structure seems to pose significantly more problems in terms of finding the optimized routes. There might be an interest, particularly in the case where a parallel setup would be available that would allow optimization of a large number of workloads simultaneously, to reduce the completion time of these outliers, as it is these difficult to optimize workloads that would actually limit the parallel completion time. As explained above, this is possible without changing the problem formulation, simply by varying the heuristics of the linear programming solver through the usage of a slightly different combination of command line parameters.

Figure 3.6 shows how the distribution evolves when using two and three heuristics compared to the single heuristic case. What we can see is that by using only two additional heuristics we are able to progress from having the slowest optimization process being 3 times slower than the average to it being only twice as slow. This is particularly of interest in the case of larger networks where this decrease in completion time can mean obtaining the results hours or even days earlier.

For some types of networks and workloads, we cannot improve the completion time further. However, for other cases, where the accelerations techniques described in Section 5.3 are applicable, we can simplify the job of the linear solver and consequently achieve lower optimization durations. The workload we chose for our benchmarks falls in this latter category and as such we can use it to evaluate the impact of the acceleration techniques.

The first acceleration technique we introduced, constraint relaxation, consisted in imposing

weaker conditions in the ILP formulation, all the while preserving the optimum maximum contention level on any link property, at the expense of load not being entirely evenly balanced. The second acceleration technique, constraint elimination, consisted in removing optimality propagation conditions when they were not needed due to the relaxed limits that the first technique introduced. Figure 3.7 illustrates the evolution of the completion time for the 512 leaf node topology when applying constraint relaxation alone and then in conjunction with constraint elimination. We also show on the same figure the original, non-accelerated completion time for reference. What we observe is that relaxing the constraints and reducing the number of constraints brings a significant improvement. The first acceleration technique reduces the average and maximum completion time by a factor of approximately 5 while the second acceleration technique further reduces the completion time by a factor of approximately 2.

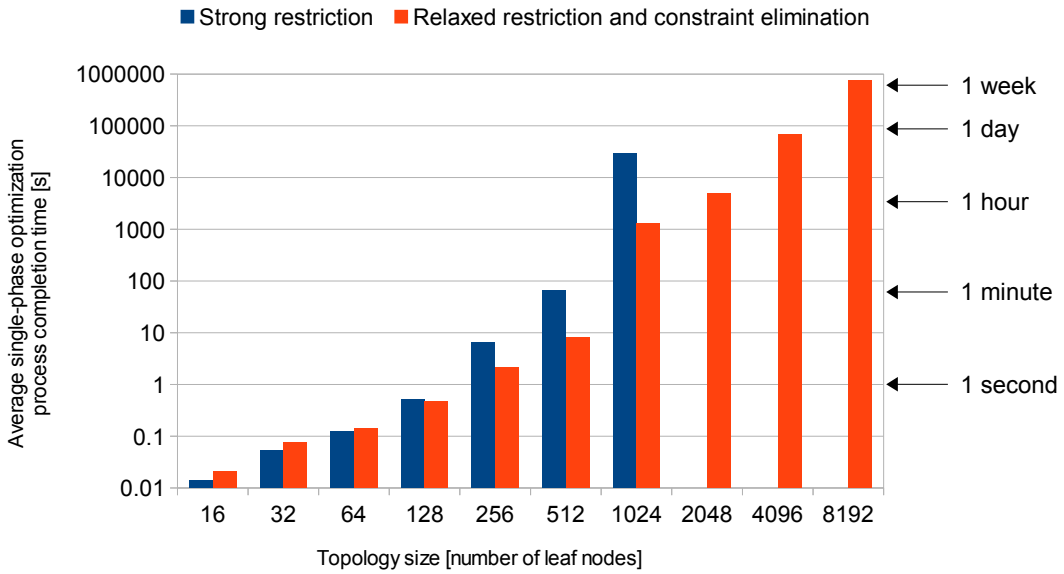


Figure 3.8 – Average completion times for the optimization process across the different phases of the all-to-all exchange on multiple XGFT topologies. The columns in blue show the completion time in the case where strong restrictions are enforced, while the columns in red show the completion time when both restriction relaxation and restriction elimination are employed. *Figure included from [87]*

Due to this significant acceleration, we are now able to apply the method in the approximately same amount of time to networks that are 4 times larger in terms of leaf nodes: routing assignments for networks with 4096 leaf nodes can be determined in under one day. The evolution of the average completion time is shown in Figure 3.8.

Given a larger time budget, larger networks can be optimized as well. However, due to the faster than linear scaling, the feasible size will not increase significantly. Finally, without changing in the least the method, but by using a more complex commercial linear programming solver

that has in particular the ability to perform the optimization of a single workload in parallel, further reductions in the optimization time can be envisioned.

3.7 Summary

We presented a method of determining an optimal assignment of routes for arbitrary communication workloads performed by concurrent tasks interconnected via arbitrary extended generalized fat tree networks. The method is based on formulating the routing problem as a set of constraints composing an integer linear programming problem and solving that problem by means of an open-source ILP solver.

As evidenced by the literature and by our own experiments, solving the integer linear programming problem for the entire network in a single step has proven to be infeasible from a computational point of view. Instead, our approach decomposes the global linear programming problem into multiple, smaller, LP formulations, each corresponding to a single layer of the fat tree network and solvable within a reasonable time frame.

Our approach ensures that the local optima can be combined into a globally optimal solution (and not just an approximation thereof), via a combination of additional global optimality-preserving constraints and a dynamic-programming-based construction of the global solution. Furthermore, to facilitate the formulation of the former, we introduced a novel generic mathematical XGFT model, referred to as *dual representation*, that reflects the routing-related properties of XGFTs in a much clearer way than existing models.

Finally, we evaluated the scalability of our approach by applying it to a very challenging workload from the routing point of view: a bandwidth-optimal all-to-all personalized exchange. Our method was able to derive the optimal routes for this workload on relatively large XGFT networks comprising up to 8192 nodes, using only commodity hardware and a freely available non-parallel solver.

4 Generalized Hierarchical All-to-all Exchange Patterns

The personalized all-to-all collective exchange is one of the most challenging communication patterns in HPC applications in terms of performance and scalability. We present a framework for the design of optimized collective patterns for generic hierarchical topologies. Our proposal can be applied, among others, to two types of topologies of great importance today: *(i)* the family of extended generalized fat tree networks (including k-ary n-trees and their variations) which are extensively used today in both HPC and commercial data centers, and *(ii)* direct low-diameter scalable hierarchical architectures such as the recently proposed dragonfly networks.

We argue that exchange patterns that are congruent with the underlying structure of the network have inherent advantages compared to patterns that are oblivious to this structure. However, the current commonly used hierarchical pattern, the XOR exchange, has limited applicability, because it requires that the number of communicating nodes equals an integral power of two, making it suitable only for few tree designs and unsuitable for any dragonfly network. We propose several new, generic, universally applicable approaches to perform such exchanges in a hierarchical fashion that outperform current state of the art approaches. We support our claims by means of both mathematical proofs and simulation results that show that we can achieve an improvement of almost two-fold in dragonflies, and a two- to three-fold improvement in fat tree networks in cases where the XOR exchange cannot be applied.

4.1 Motivation

Most parallel algorithms alternate between phases of computation and communication. The way in which data is exchanged among a group of parallel tasks can vary widely, but in practice,

This chapter is based on the article [85] PRISACARI, B., RODRÍGUEZ, G., AND MINKENBERG, C. Generalized hierarchical all-to-all exchange patterns. In *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013* (2013), IEEE Computer Society, pp. 537–547. https://doi.org/10.1109/IPDPS.2013.87

most data exchanges involving more than a pair of tasks can be mapped to a small set of typical exchanges, for which most parallel programming models (e.g., distributed shared memory, message passing, partitioned global address space) provide specialized collective communication operations. This serves two main purposes. It frees the programmer from the burden of correctly implementing typical (but not necessarily trivial) well-defined exchanges, and supplies the programmer with efficient exchanges, optimized for specific systems. Communication libraries generally offer several implementations for the same collective data exchange optimized for different message sizes or different numbers of communicating tasks. In this chapter, we focus on the most communication-intensive of these collective communication operations, namely the *personalized all-to-all exchange*, in which each task must share a distinct message with every other task.

Naturally, the performance of such communication-intensive operations depends strongly on the communication subsystem and its main characteristics, such as bandwidth, latency, topology, and routing. The focus of this chapter is on the topological aspect. Hierarchical topologies dominate HPC systems and commercial data centers with variations of fat trees as the preferred hierarchical network topology in use today. Non-hierarchical direct networks such as tori still have a place in HPC, but the current trend is towards hierarchical, low-diameter, scalable direct networks such as dragonflies and their variations [54], which are widely considered as the cornerstone of the Exascale effort [14].

Hierarchical topologies consist of multiple layers, each layer comprising a plurality of identical instances of the preceding layer in a recursive manner. Our objective is to demonstrate that adopting a topology-aware exchange pattern that is congruent with the topological hierarchy of the network can significantly reduce time-to-completion compared to topology-oblivious exchange patterns. We introduce a set of generic, hierarchy-aware exchange patterns and prove that they maximize locality of communication at each layer of the hierarchy. Moreover, we evaluate the performance gains they entail on fat tree and dragonfly topologies in comparison with conventional linear and XOR exchange patterns.

The chapter is organized as follows. We start by defining the concept of a hierarchical topology and show how this concept maps onto practical interconnection network implementations such as extended generalized fat trees and dragonflies in Section 4.2. We proceed to review existing all-to-all exchange patterns and existing approaches towards improving their performance in Section 4.3. Our proposals for hierarchy-aware exchanges and the mathematical proofs of their properties are the subject of Section 4.4. Finally, section 7.4 presents experimental results, obtained by means of simulation, that confirm the advantages of performing the all-to-all exchange in a hierarchy-aware way.

4.2 Hierarchical Topologies

A significant fraction of the designs used today in architecting high-performance computing systems, datacenter networks and on-chip interconnects can be viewed as hierarchical topolo-

gies. This concept includes, among others, all variations of tree interconnects (trees, fat trees [60], extended generalized fat trees (XGFTs) [76]) as well as hierarchical fully-meshed designs such as the dragonfly [53, 54]. All hierarchical topologies share a few key features. Every such topology can be viewed as a set of progressively more complex layers, where each subnetwork at layer l is made up of several layer-$(l-1)$ subnetworks interconnected in a specific way. In this chapter, we will only concern ourselves with topologies that exhibit the regularity property that all subnetworks of a specific layer are identical, which immediately leads to every layer-l subnetwork having the same number H_l of sub-subnetworks at layer $l-1$. Also, we consider the top layer to be the unique layer encompassing the entire network.

In the case of a tree-shaped topology, with $L+1$ levels numbered from 0 (leaves) to L (root), the first layer is that of the individual leaf node ($H_0 = 1$). Each layer-1 subnetwork is then composed of a maximal set of leaf nodes that share the same level-1 ancestor. Each layer-2 subnetwork is composed of a maximal set of leaf nodes that share the same level-2 ancestor, and so on. In the case of an XGFT, where a single node may have several ancestors, the definition changes slightly. A layer-l subnetwork in this case is the maximal set of leaf nodes that share the same *set* of level-l ancestors. The top layer L has a single subnetwork encompassing all leaf nodes. Thus, the hierarchical model of the tree has $L+1$ levels as well, with $H_0 = 1$ and H_l equal to the number of descendants per node at level l, where l is an arbitrary non-leaf level of the tree, $1 \leq l \leq L$.

In the case of an explicitly hierarchical topology such as the dragonfly, the notion of layers is explicitly present in the topology's definition. Layer-0 subnetworks are once more the individual host nodes, a layer-1 subnetwork is the maximal set of host nodes connected to the same switch node, a layer-2 subnetwork is the maximal set of host nodes belonging to the same dragonfly level-1 group of switch nodes and so on. If we use the (p,a,h) notation introduced by [54], the hierarchical model has thus 4 levels, with $H_0 = 1$, $H_1 = p$, $H_2 = a$ and $H_3 = a \cdot h + 1$,

An immediately emerging property of this layered structure is a notion of locality. A specific leaf node n belongs to a specific layer-1 subnetwork, which in turn belongs to a specific layer-2 subnetwork, and so on, up to the final unique layer-L subnetwork. Owing to the hierarchical structure, node n is closest to the nodes within its local layer-1 subnetwork. The next closest nodes are those in the other layer-1 subnetworks within the same layer-2 subnetwork as node n, and so on. As such, we can qualify the distance between two nodes by determining their lowest common ancestor layer, i.e., the lowest layer at which both nodes are in the same subnetwork. Figures 4.1 and 4.2 illustrate the "vicinities" of a given node in a tree-shaped topology and in a dragonfly topology, respectively.

In such a design, we propose to identify or address a specific node in a way that carries hierarchical information. First, for every l between 1 and L, we assign every layer-$(l-1)$ subnetwork belonging to a same layer-l subnetwork an index h_l between 0 and $H_l - 1$. The

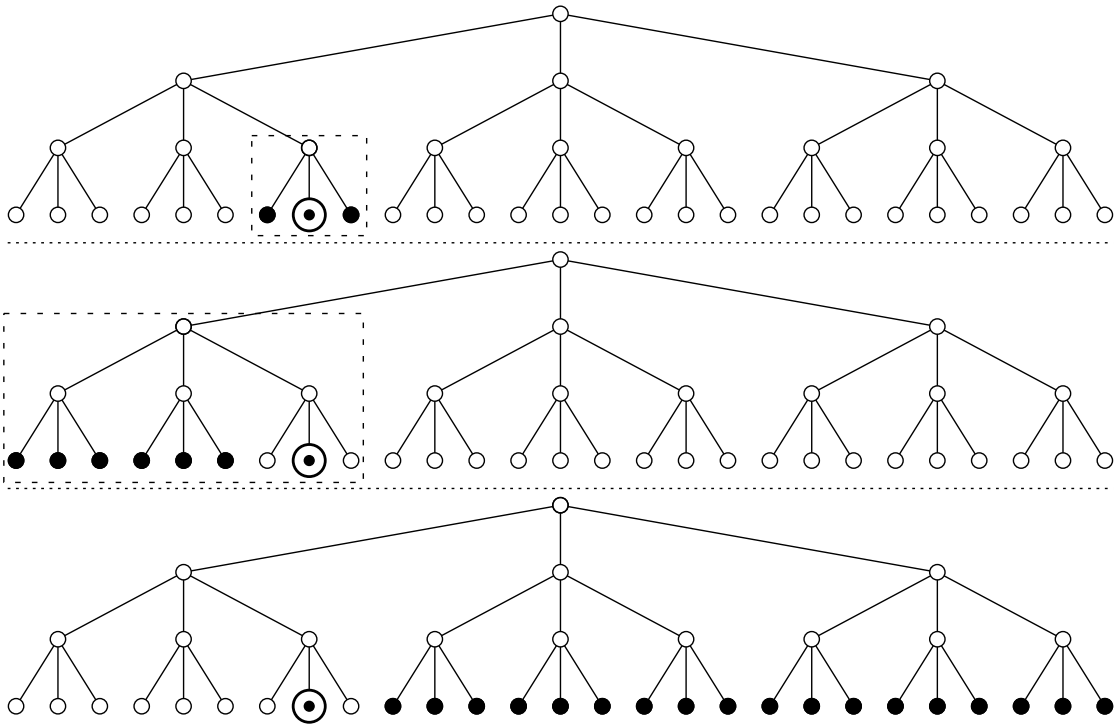


Figure 4.1 – The three different vicinities of a given node (the dotted node) in the context of a 3-level tree interconnect. The vicinities are composed of the black filled nodes and have a decreasing level of locality from top to bottom. A vicinity is defined by the layer at which the nodes in that vicinity and the original node share the lowest common subnetwork (shown with a dashed outline, except for the last vicinity where the lowest common subnetwork is the entire network).
Figure included from [85] ©2013 IEEE

address of a node n will then be the tuple of indices $(h_0^n, h_1^n, ..., h_L^n)$ where h_l^n is the index assigned to the layer-l subnetwork that n belongs to and $h_0^n = 1$.

By construction, this addressing scheme uniquely identifies every communicating node. That is to say:

$$n_1 = n_2 \Leftrightarrow h_l^{n_1} = h_l^{n_2}, \forall\, l \text{ s.t. } 0 \leq l \leq L. \tag{4.1}$$

In section 4.4 we will leverage this addressing model to define generic hierarchical exchange patterns.

4.3 All-to-all Exchange Patterns

The all-to-all exchange, also known in literature as the personalized all-to-all exchange, is a collective communication pattern where every task in a group of N tasks needs to send a

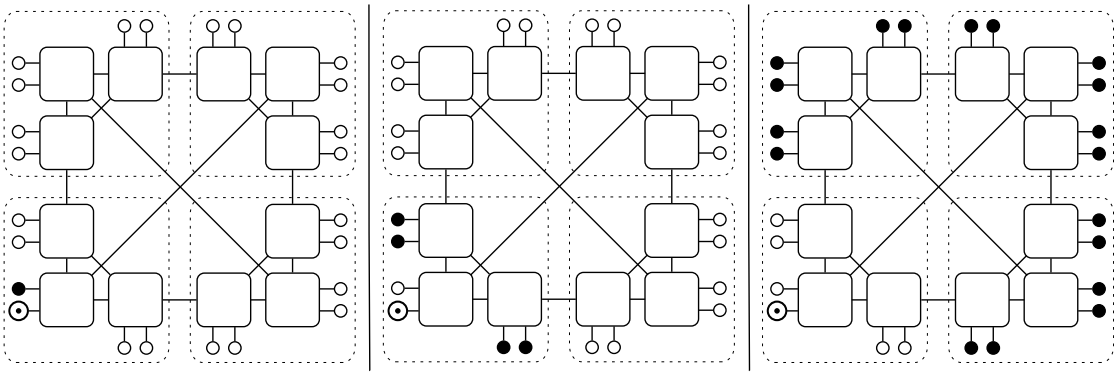


Figure 4.2 – The three vicinities of a given host node (the dotted node) in the context of a dragonfly where every switch is connected to 2 hosts, there are 3 switches in every local group and every switch has 1 remote port ($p = 1$, $a = 3$, $h = 1$). The vicinities are composed of the black filled nodes and have a decreasing degree of locality from left to right.
Figure included from [85] ©2013 IEEE

distinct message to every other task. We will consider that a task sends a message to itself as well, and as such N^2 message must be exchanged in total.

As such a large number of messages are injected into the network, it is very important to arrange the messages in such a way as to reduce contention. Conflicts can appear at three main points:

1. At the source (the interface from task to network): multiple messages from the same task may contend simultaneously for entry into the network.

2. At the destination (the interface from network to task): multiple messages may contend simultaneously for delivery to the destination task.

3. Within the interconnect: multiple messages may contend simultaneously for a given network resource (link or switch buffer).

A straightforward way [111] to eliminate the first two sources of conflicts is to separate the exchange in so called *phases*. In every such phase, every task injects a single message, i.e., no new message is sent before the previous one has completed. This can be achieved, for example, by using blocking message transfers where the destination must signal to the source the reception of the message via an acknowledgement. Furthermore, in every phase, no destination receives more than one message. In other words, the all-to-all exchange becomes a series of phases where each phase is a permutation of the communicating tasks. Such an exchange is completely determined by the function $f_p(s) = d$ that establishes the destination d of the message from task s in phase p.

The third source of contention is more difficult to counteract, as the occurrence of conflicts

depends not only on the structure of the phases, but also on the underlying topology and the policies in place (e.g. the routing algorithm).

4.3.1 Typical exchange patterns

Two widely used exchange patterns today are the binary XOR exchange and the linear shift exchange [113].

In the case of the XOR exchange, the mapping of phase and source to destination is given by the function $f_p^{XOR}(s) = s \oplus p$ where $\oplus$ is the logical exclusive or operation performed bit by bit on the binary representations of s and p [97]. A major drawback of this method is that for the properties of the all-to-all exchange to be maintained, the total number of communicating tasks must be a power of two, otherwise the XOR operation can yield destination identifiers that are out of the valid range. When this is the case, the exchange has the additional property of being performed in an increasingly remote and contained manner. The first exchanges performed are all possible exchanges between nodes whose indices yield the same quotient when divided by 2, then all remaining possible exchanges between nodes whose indices yield the same quotient when divided by 4, and so on, in increasing powers of two. This is illustrated in Figure 4.3.

In the case of the linear shift exchange, the mapping of phase and source to destination is given by the function $f_p^{LIN}(s) = (s + p + \text{shift}) \bmod N$, where shift is a parameter of the exchange that takes a constant integer value between 0 and $N-1$. Unlike XOR, this exchange does not have the property of being composed of contained increasingly remote exchanges. However, it is also not limited to power-of-two numbers of communicating tasks, making it currently the preferred alternative in such cases. Figure 4.4 illustrates this exchange pattern for 16 communicating tasks in the case where shift = 0.

4.3.2 Related work on improving all-to-all exchanges

Among other research directions in the optimization of collective communication algorithms, two types of endeavors are related to our contribution: *(i)* approaches that change the exchange pattern itself to better adapt to the underlying topology, and *(ii)* approaches focused on mapping the communicating tasks to an approximate, virtual, typically hierarchical layout onto which to execute the exchange pattern.

Topology-aware optimizations of the exchange pattern *(i)* have either considered an abstract model of the network and proposed general exchanges that lead to an optimal number of rounds or an optimal amount of bytes sent for that abstract model [21] (but that approach is too abstract and too general to lead to an efficient exchange for a concrete topology) or they have considered topologies different from fat trees and dragonflies [47] (2D and 3D meshes and hypercubes). A notable attempt in this direction, but not related to our focus on hierarchical networks is that of Bruck [20]. He proposes a way to construct an optimal set of exchanges,

for a fully-connected mesh of multi-port nodes, that, by leveraging message aggregation and store-and-forward capabilities (proxy processes), can tune the number of rounds (phases) to complete the exchange (at the expense of increasing the sizes of the messages exchanged). Finally, the main exchange pattern targeting hierarchical topologies is the previously presented binary XOR exchange. This pattern has its origin in an algorithm for organizing the exchanges in a hypercube network topology [19, 97]. The XOR exchange was an elegant solution to the all-to-all exchange, guaranteeing no contention at the end points (sender or receiver). It could also be extended to other topologies by overlaying them with a virtual hypercube layout [111], and furthermore, it mapped particularly well to fat tree networks and full-bisection XGFTs (or k-ary n-trees). Along with the linear shift pattern, it is one of the most widely used patterns today, implemented in most MPI libraries as the preferred pattern for exchanges involving powers of two numbers of processors [113].

Many works have attempted to take the approach *(ii)* of mapping the communicating tasks to a virtual topology that approximates the real interconnect. Among the different possible mappings, hierarchical virtual topologies, typically tree-shaped, have been the preferred choice. Mapping the tasks onto a tree provides an intuitively efficient topology to plan and execute collective operations. Moreover, if the network exhibits some hierarchical structure, the virtual tree would capture it. Such a hierarchy exists for instance within the MPI tasks running in a cluster of SMPs. Husbands [43] took a first step towards solving this problem for SMPs, where the intra-node communication is more efficient than inter-node communication, by hierarchically splitting the communication phases in a broadcast (between SMPs first, then within the SMPs). Karonis [50] improved on it for the Broadcast collective operation. Sistare [99] adopted the basic approach from Husbands and improved it by selecting from each SMP a single MPI process to be responsible for the communications at the edge of the SMP and then constructing the spanning tree among those selected MPI processes for the inter-node communication.

A work that does not attempt to optimize the all-to-all exchange, but that provides an interesting analysis for stencil communication patterns (multi-dimensional lattice exchanges, which generally consist of exchanges among tasks separated by constant strides) is [16]. In the context of dragonflies, this work shows that random task placement significantly improves the performance of these stencil patterns against other more deterministic task-to-node placements. Stencil communication patterns and all-to-all exchange patterns are highly related. We will confirm in this chapter that a random placement does indeed lead to improved performance against a contiguous placement for strided (linear shift) exchange patterns, but we will also show that the approach that we propose achieves even better performance with a deterministic task placement and organization of the exchanges.

Other efforts in network performance optimization (such as for example [42, 113, 115]) deal with aspects of traffic management, such as the routing strategy, that are orthogonal to our work. These efforts take the traffic pattern as an input and optimize the way that the network handles it, but they do not attempt to shape the pattern itself.

To our knowledge, the XOR exchange is the only attempt at a generic hierarchical exchange (albeit with limited applicability). The aforementioned efforts that go in the same direction as our work either provide simple solutions to very precise scenarios or try to map the real topology onto a virtual topology where XOR is applicable. In contrast, here we propose a set of generic patterns, applicable to any hierarchical topology, from arbitrary XGFTs (widely deployed in HPC systems and datacenters today) to dragonfly networks (conceived and identified as one of the most promising topologies for the Exascale effort).

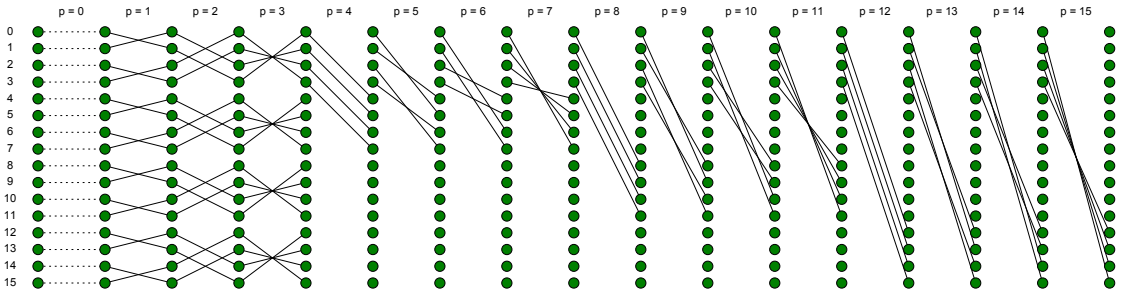


Figure 4.3 – Graphical representation of the binary XOR exchange pattern for a set of 16 communicating tasks. The first 2 phases are composed of messages between tasks whose indices yield the same quotient when divided by 2. The next 2 phases are composed of the remaining messages between tasks whose indices yield the same quotient when divided by 4. Phases 4 to 7 are composed of the remaining messages between tasks whose indices yield the same quotient when divided by 8 and the rest of the phases are composed of the remaining messages. For the first three categories, all exchanges within those phases are shown. For the rest, for clarity, only the exchanges originating from the first 4 tasks are shown.
Figure included from [85] ©2013 IEEE

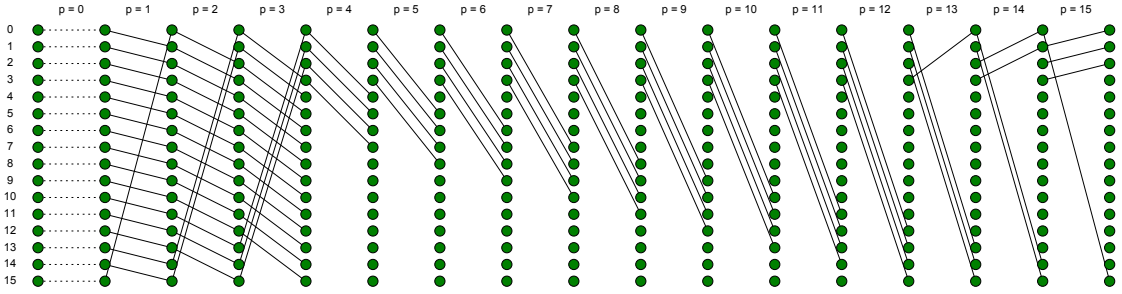


Figure 4.4 – Graphical representation of the linear shift exchange pattern for a set of 16 communicating tasks and a shift parameter equal to 0. For clarity and in order to have a similar illustration of the exchange as that for the binary XOR, starting with phase 4 we only show messages originating from the first 4 tasks.
Figure included from [85] ©2013 IEEE

4.4 Generalized Hierarchical Exchange Patterns

As discussed in Section 4.2, all hierarchical topologies have an intrinsic notion of locality. When needing to perform an all-to-all exchange, there are benefits to performing it in a locality-

aware way that ensures that as many phases as possible are contained within every layer. We say that a phase p is contained within layer l of the hierarchy if for all message exchanges happening in phase p the source and the destination of any given message both belong to the same layer l subnetwork. We call an exchange that has this property of locality awareness a *hierarchy-aware exchange* or simply a *hierarchical exchange*. Hierarchical exchanges have the advantage of ensuring that traffic is contained at lower hierarchy levels in as many phases as possible, leaving the rest of the network free for as much time as possible. This containment of traffic in certain regions of the network has the benefit of allowing additional independent application traffic to use the network with less interference, and thus increased performance.

We further define a phase p to be a *layer l phase* if it is contained within layer l but does not contain any message exchanges between a source and a destination belonging to a common layer $l-1$ subnetwork. Due to the property presented above of maximizing the number of phases contained within every layer, it follows that the set of phases contained in layer l is made up of the set of phases contained in layer $l-1$, phases that exhaust all messages within layer $l-1$ subnetworks, and a set of layer l phases, phases that exhaust messages between layer $l-1$ subnetworks. Recursively, this leads to every phase p being a layer-l_p phase, for a certain value of l_p. Layer-l phases have the property of ensuring that all exchanges belonging to such a phase have identical or similar communication latency (in the absence of contention). Therefore, an exchange pattern that takes into account the hierarchical structure of the topology inherently provides much better synchronization within each of the phases. This is a critical factor in optimizing the overall performance of the exchange, as desynchronization implies either that consecutive phases will overlap, thus causing additional contention, or that gaps in between consecutive phases will emerge, unnecessarily increasing the completion time of the exchange.

As a final emphasis, these benefits are an exclusive consequence of the internal structure of the phases while their order is in itself not relevant, i.e., once the structure of each phase is established in a way that respects the constraints presented above, the phases themselves can be performed in any order. Nonetheless, choosing an ordering that groups together layer l phases, for every l, or even more, further ordering these groups in increasing order of l, has the additional advantage of giving temporal breadth to the benefits presented above, e.g. keeping higher levels of the hierarchy unused for longer contiguous amounts of time.

To date, a single algorithm exists that performs exchanges in this manner, the binary XOR pattern, and due to its benefits it is widely employed. However, this algorithm is only usable when the total number of communicating tasks is a power of two, which is an important restriction. In this section, we will propose a universally applicable approach to performing all-to-all exchanges in a hierarchical fashion that performance-wise is on par with XOR where XOR is applicable and outperforms the approaches used as alternatives to XOR when XOR is or is not applicable.

4.4.1 Generalized Hierarchical Exchange

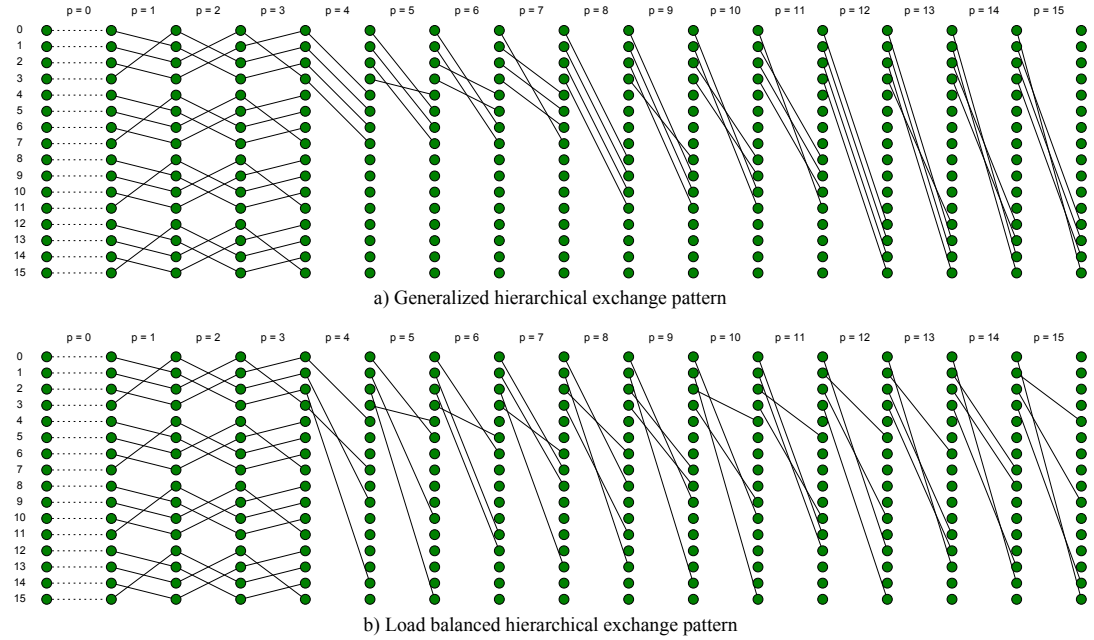


Figure 4.5 – Graphical representation of a) the generalized hierarchical exchange pattern and b) the load balanced hierarchical exchange pattern for a hierarchy with 3 layers and parameters $H_0 = 1, H_1 = H_2 = 4$. The first phase takes place at layer 0 (dotted lines), phases 1 to 3 take place at layer 1 while the rest of the phases take place at layer 2. For the layer 2 phases, only the messages originating in the first layer 1 subnetwork are shown for clarity, while for the rest of the phases, all messages are represented.
Figure included from [85] ©2013 IEEE

The approach we propose is based on the addressing scheme presented in Section 4.2: every task t is assigned an address that takes the form of a $(h_0^t, h_1^t, ..., h_L^t)$ tuple, with $0 \leq h_l^t \leq H_l - 1$. The transition from node address to task address is done in the following way. Since several tasks can share a single node, H_0 is no longer always 1 but rather is equal to the number of tasks assigned to each node. As before, each task belonging to a common node is given a unique index h_0 between 0 and $H_0 - 1$. The remaining components of the task address take the same values as the corresponding components of the address of the node on which the task is placed.

As an all-to-all exchange among N tasks is made up of N phases, and as within a hierarchical topology with parameters H_0, H_1, ..., H_L the number of tasks equals $N = H_0 \cdot H_1 \cdot ... \cdot H_L$, it immediately follows that we can assign to each phase a unique index p between 0 and $N - 1$. As such, every phase p can also be assigned a representation $(h_0^p, h_1^p, ..., h_L^p)$ that is similar in

structure to the task addresses, in that every component h_l^p takes a value between 0 and the hierarchy parameter H_l minus 1. The components of the phase address are determined by

$$h_l^p = (p \div (H_0 \cdot H_1 \cdot ... \cdot H_{l-1})) \bmod H_l, \tag{4.2}$$

$\div$ being the integer division operation. The representation thus defined satisfies the uniqueness property:

$$p_1 = p_2 \Leftrightarrow h_l^{p_1} = h_l^{p_2}, \forall l \text{ s.t. } 0 \leq l \leq L. \tag{4.3}$$

As explained in Section 4.3, a phased all-to-all exchange pattern is completely determined by the relationship that defines for every given source s and phase p what the destination d of the message sent by s during that phase is. The relationship that we propose is the following. If $(h_0^s,h_1^s,...,h_L^s)$ is the address of the source, $(h_0^d,h_1^d,...,h_L^d)$ is the address of the destination and $(h_0^p,h_1^p,...,h_L^p)$ is the representation of the current phase, then the destination is determined as follows:

$$h_l^d = (h_l^s + h_l^p) \bmod H_l, \forall l \text{ s.t. } 0 \leq l \leq L. \tag{4.4}$$

As, according to Equation (4.1), a given address uniquely identifies a specific task, Equation (4.4) uniquely identifies a specific destination $d_{s,p}$ for any given source and phase.

The exchange thus defined is a valid phased all-to-all exchange and, moreover, it has the hierarchical properties described above, as proven by Theorem 4 below. A graphical representation of the exchange is provided in Figure 4.5 a).

Lemma 1.

$\forall m \in \mathbb{N}^*, a,b,c \in \mathbb{N}$ s.t. $0 \leq b,c < m$

$((a+b) \bmod m = (a+c) \bmod m) \Leftrightarrow b = c.$

Proof. The right to left implication $(((a+b) \bmod m = (a+c) \bmod m) \Leftarrow b = c)$ is obviously true. To prove the left to right implication $(((a+b) \bmod m = (a+c) \bmod m) \Rightarrow b = c)$, let us denote $r = (a+b) \bmod m = (a+c) \bmod m$, where $0 \leq r < m$. Thus, $a+b = q_1 \cdot m + r$ and $a+c = q_2 \cdot m + r$, where q_1 and q_2 are integers. It follows that $b-c = (q_1 - q_2) \cdot m$ and since $-m < b-c < m$ we can conclude that $b-c = 0$, equivalent to $b = c$.

□

Theorem 4. *The generalized hierarchical exchange defined by Equation* (4.4) *has the following properties and is thus a valid and hierarchical exchange:*

- *For any given source, no two messages are sent to the same destination over the entire set of phases.*
- *In any given phase, no two messages are sent to the same destination.*
- *For any given hierarchy layer, a maximum number of phases are contained within that layer.*

Proof. Proving the first property is equivalent to showing that:

$$d_{s,p_1} = d_{s,p_2} \Rightarrow p_1 = p_2, \forall s. \tag{4.5}$$

For an arbitrary s, $d_{s,p_1} = d_{s,p_2}$ implies that $h_l^{d_{s,p_1}} = h_l^{d_{s,p_2}}$, $\forall l$ s.t. $0 \leq l \leq L$. For an arbitrary l and s, this further implies that $(h_l^s + h_l^{p_1}) \bmod H_l = (h_l^s + h_l^{p_2}) \bmod H_l$. As all of h_l^s, $h_l^{p_1}$ and $h_l^{p_2}$ have values between 0 and $H_l - 1$, we can conclude that this leads to $h_l^{p_1} = h_l^{p_2}$ (using Lemma 1). Since this was proven for an arbitrary l, it holds for all the possible values of l which means that we have proven that the representations of p_1 and p_2 are identical, which in turn leads according to Equation (4.3) to $p_1 = p_2$.

The second property of the exchange is equivalent to:

$$d_{s_1,p} = d_{s_2,p} \Rightarrow s_1 = s_2, \forall p \tag{4.6}$$

and the demonstration is analogous to that of the first property.

To prove the third property of the hierarchy, let us first start by determining what the maximum number of phases contained within layer l is. Every subnetwork at layer l contains $N_l = H_0 \cdot H_1 \cdot ... \cdot H_l$ tasks. Among the messages that every one of these tasks sends, exactly N_l are destined to tasks within the same subnetwork. As such, at most N_l phases can be contained within the layer. We will show that the exchange we propose leads to exactly N_l phases being contained within layer l of the hierarchy. In order for a given phase p to be contained within layer l, for every source s, the destination d must belong to the same layer l subnetwork as s. With the addressing scheme we proposed, this is equivalent to $h_k^s = h_k^d, \forall k$ s.t. $l < k \leq L$. It follows that $(h_k^s + 0) \bmod H_k = (h_k^s + h_k^p) \bmod H_k, \forall k$ s.t. $l < k \leq L$ which, using Lemma 1, leads to $h_k^p = 0, \forall k$ s.t. $l < k \leq L$. Or the number of phases that have this property is $H_0 \cdot H_1 \cdot ... \cdot H_l$, that is, N_l. □

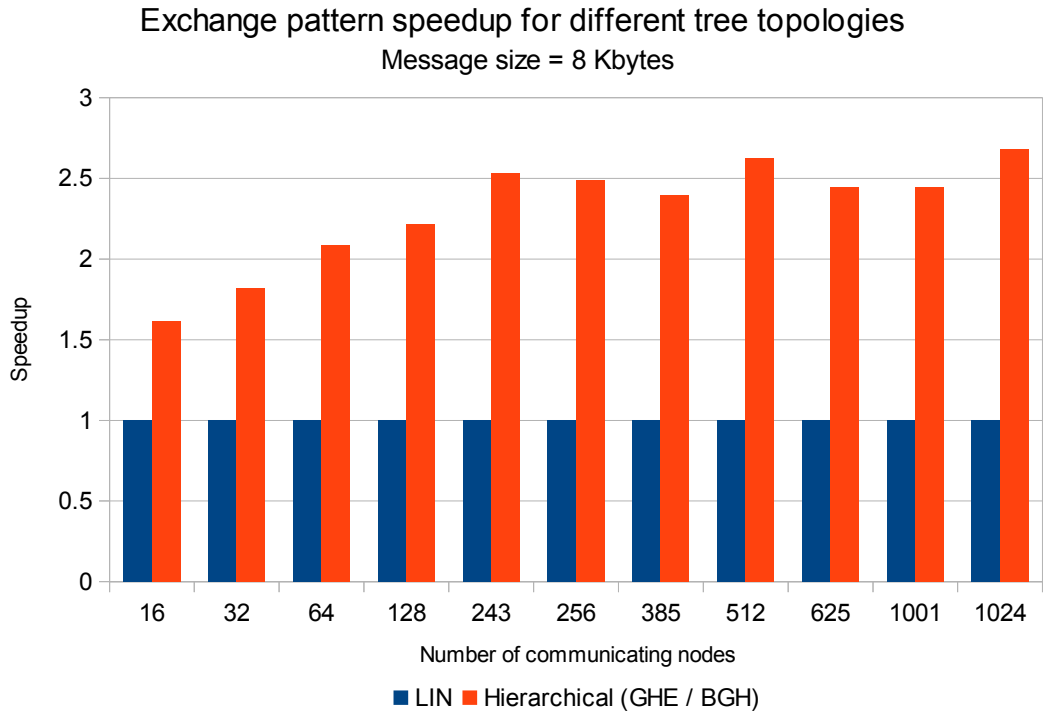


Figure 4.6 – Speedup of exchange completion (where the base completion time is that of LIN) for several XGFT topologies, for a size of the exchanged messages equal to 8 Kbytes. Higher bars correspond to lower completion time - thus, higher is better. In all but the smallest configurations, both hierarchical exchanges exhibit a completion speed that is at least twice as large than that of the linear shift.
Figure included from [85] ©2013 IEEE

4.4.2 Load Balanced Generalized Hierarchical Exchange

The exchange introduced in Section 4.4.1, although hierarchical, has a slight drawback. Every phase at layer l has the property that if a source in a certain subnetwork sn_l^1 at layer l sends to a destination in a certain other subnetwork sn_l^2 also at layer l, then, in that phase, all sources in sn_l^1 will send to destinations that are all in sn_l^2. When there is enough bandwidth between subnetworks, such as is the case in fat trees, this poses no problem. However, in dragonfly networks there typically is not enough bandwidth to accommodate all this traffic in a contention-free manner. Therefore, it is useful to distribute the destinations of the messages issued by sources in sn_l^1 over as many subnetworks (distinct from sn_l^1) at layer l as possible, in every phase. This maintains the hierarchical property of the exchange, while also load balancing traffic over the inter-subnetwork links at layer l. The exchange we propose is the

following. As before, let $(h_0^s,h_1^s,...,h_L^s)$ be the address of the source, $(h_0^d,h_1^d,...,h_L^d)$ be the address of the destination and $(h_0^p,h_1^p,...,h_L^p)$ the representation of the current phase. In addition, let

$$\omega_l^s = \sum_{k=0}^{l-1} h_k^s \cdot N_{k-1} \tag{4.7}$$

and

$$\alpha_l^{s,p} = \begin{cases} 0, & \text{if } h_l^p = 0 \\ (h_l^p - 1 + \omega_l^s) \bmod (H_l - 1) + 1, & \text{otherwise} \end{cases}. \tag{4.8}$$

Then the destination is determined as follows:

$$h_l^d = (h_l^s + \alpha_l^{s,p}) \bmod H_l,\ \forall l \text{ s.t. } 0 \le l \le L. \tag{4.9}$$

Within each subnetwork at layer l, ω_l^s assigns each source a unique offset (between 0 and $N_l - 1$). This offset is then used (via $\alpha_l^{s,p}$) to determine what other subnetwork at layer l should contain the destination of the corresponding source. Whereas before, during an arbitrary phase at layer l, all sources belonging to a same layer l subnetwork were sending to destinations within a single, different, layer l subnetwork, now their destinations are evenly spread out across all layer l subnetworks distinct from the source subnetwork. Also, let us remark that for all s, p and l, $\alpha_l^{s,p} = 0 \Leftrightarrow h_l^p = 0$ and $1 \le \alpha_l^{s,p} < H_l \Leftrightarrow h_l^p \neq 0$.

This exchange is valid and hierarchical - this is proven in Theorem 5 below. A graphical representation of this exchange is provided in Figure 4.5 b).

Theorem 5. *The balanced generalized hierarchical exchange defined by Equation* (4.9) *has the following properties and is thus a valid and hierarchical exchange:*

- *For any given source, no two messages are sent to the same destination over the entire set of phases.*
- *In any given phase, no two messages are sent to the same destination.*
- *For any given hierarchy layer, a maximum number of phases are contained within that layer.*

Proof. As in Theorem 4, we will start by showing that an arbitrary source sends to distinct destinations in all phases. Indeed, if we assume that in two phases p_1 and p_2 the destinations

of a fixed source s are identical, we have that $(h_l^s+\alpha_l^{s,p_1}) \bmod H_l = (h_l^s+\alpha_l^{s,p_2}) \bmod H_l \forall l \text{s.t.} 0 \leq l \leq L$. For an arbitrary l, by virtue of Lemma 1 and of the fact that $0 \leq h_l^s, \alpha_l^{s,p_1}, \alpha_l^{s,p_2} < H_l$ it follows that $\alpha_l^{s,p_1} = \alpha_l^{s,p_2}$. This leads to either both αs being 0, case in which $h_l^{p_1} = h_l^{p_2} = 0$, or to both αs being strictly larger than 0 and $(h_l^{p_1} - 1 + \omega_l^s) \bmod (H_l - 1) = (h_l^{p_2} - 1 + \omega_l^s) \bmod (H_l - 1)$. Using Lemma 1 this leads to $h_l^{p_1} = h_l^{p_2}$. We have proven that for an arbitrary l, $h_l^{p_1} = h_l^{p_2}$ thus by virtue of Equation (4.3) this implies $p_1 = p_2$. Thus in no two distinct phases does an arbitrary source s send to two identical destinations.

Now we will show that no two different sources send to the same destination within the same phase. Let us assume that two sources s_1 and s_2 send to the same destination in phase p, i.e., $(h_l^{s_1} + \alpha_l^{s_1,p}) \bmod H_l = (h_l^{s_2} + \alpha_l^{s_2,p}) \bmod H_l \; \forall l \, \text{s.t.} \, 0 \leq l \leq L$. We will show by mathematical induction that for every l, $h_l^{s_1} = h_l^{s_2}$. For $l = 0$, $\omega_0^{s_1} = \omega_0^{s_2} = 0$ and thus $\alpha_l^{s_1,p} = \alpha_l^{s_2,p} = h_l^p$. From $h_0^{d_1} = h_0^{d_2}$ we thus obtain $(h_0^{s_1} + h_0^p) \bmod H_0 = (h_0^{s_2} + h_0^p) \bmod H_0$ which leads by virtue of Lemma 1 to $h_0^{s_1} = h_0^{s_2}$. Let us now assume that for every level k below l, $h_k^{s_1} = h_k^{s_2}$. It immediately follows that $\omega_l^{s_1} = \omega_l^{s_2}$ which leads to $\alpha_l^{s_1,p} = \alpha_l^{s_2,p}$. Or since $(h_l^{s_1} + \alpha_l^{s_1,p}) \bmod H_l = (h_l^{s_2} + \alpha_l^{s_2,p}) \bmod H_l$ by virtue of Lemma 1 this leads to $h_l^{s_1} = h_l^{s_2}$. In conclusion, by mathematical induction, for all l, $h_l^{s_1} = h_l^{s_2}$ which by virtue of Equation (4.1) entails that $s_1 = s_2$.

Regarding the hierarchical property of this balanced exchange, the same argument that we used in the proof of Theorem 4 can be applied here. □

4.4.3 Latency Balanced Generalized Hierarchical Exchange

Although we have thus far analyzed hierarchical topologies in general, there is an essential difference between tree-based and dragonfly topologies that stems from their indirect, respectively direct, nature. In the former case, communication between subnetworks is done via layers of routing nodes without compute capability (and therefore without any tasks), whereas in the latter case, each routing node is also a compute node. Therefore, in the case of tree-based hierarchies, regardless of how a hierarchical exchange is structured, its phases are made up exclusively of exchanges of identical duration (because the paths of the messages of each exchange are identical in terms of number and type of links). For the dragonfly, however, this is no longer the case. As such, we introduce a final hierarchical exchange pattern in which all messages in a given phase traverse paths of equal length and structure in a dragonfly.

For a given source task t, we define the following exchange:

- The first H_0 messages are sent to the tasks within the same node as t (t included), node n
- The next $H_0 \cdot (H_1 - 1)$ messages are sent to tasks on nodes connected to the same switch as n (n excluded), switch s

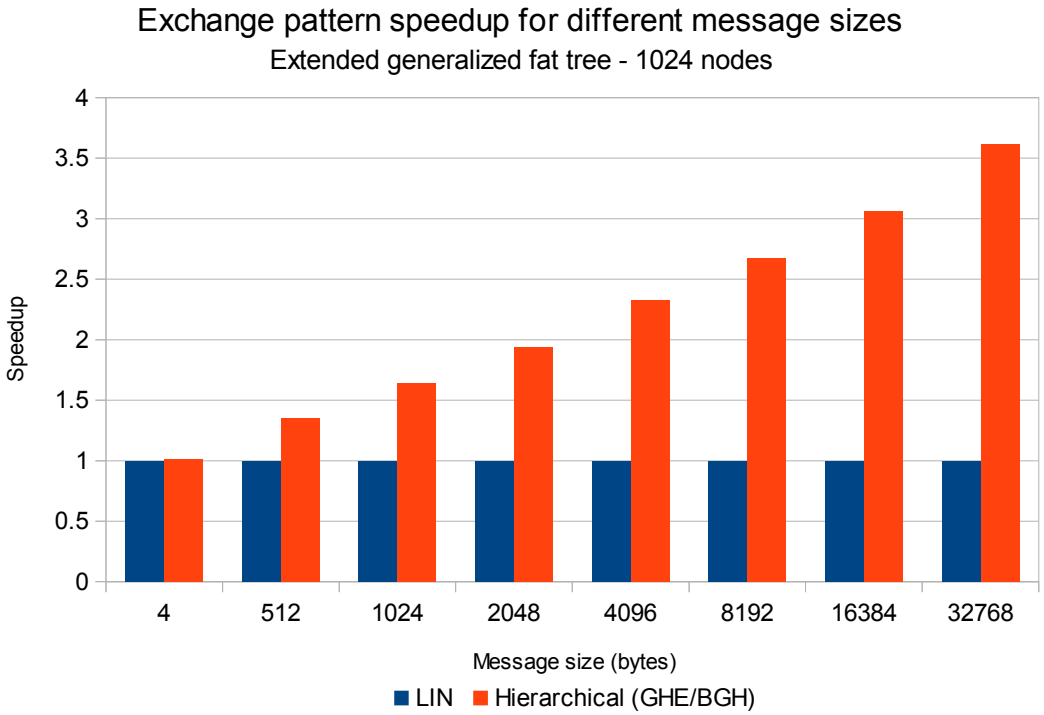


Figure 4.7 – Speedup of exchange completion (where the base completion time is that of LIN) for a fixed XGFT topology interconnecting 1024 nodes, for different sizes of the exchanged messages. Higher bars correspond to lower completion time - thus, higher is better. While for the smallest of messages, there is practically no difference between the linear shift and hierarchical exchanges, as the message size increases, so does the completion speed of the hierarchical exchanges relative to that of LIN. For the largest analyzed message size, the hierarchical exchanges are more than three times faster.
Figure included from [85] ©2013 IEEE

- The next $H_0 \cdot H_1 \cdot (H_2 - 1)$ messages are sent to tasks on nodes connected to switches directly connected to s through local links
- The next $H_0 \cdot H_1 \cdot h$ messages are sent to tasks on nodes connected to switches directly connected to s through distant links (h is the number of distant link ports on every switch, $h = (H_3 - 1)/H_2$)
- The next $H_0 \cdot H_1 \cdot h \cdot (H_2 - 1)$ messages are sent to tasks on nodes connected to switches connected to s through a distant link followed by a local link
- The next $H_0 \cdot H_1 \cdot h \cdot (H_2 - 1)$ messages are sent to tasks on nodes connected to switches connected to s through a local link followed by a distant link
- The next $H_0 \cdot H_1 \cdot h \cdot (H_2 - 1)^2$ messages are sent to tasks on nodes connected to switches connected to s through a local link followed by a distant link followed by a local link

To avoid excessive contention, we again apply a load balancing technique analogous to that of the load balanced generalized hierarchical exchange. Whenever messages issued by several tasks need to use the same type of links, they are distributed over as many links of that type as possible, with the same (ω,α)-based approach as before, thus preserving the exchange's hierarchical structure.

In the following section, we will demonstrate by means of system performance simulations that these exchanges significantly outperform current state-of-the-art approaches.

4.5 Experimental Results

4.5.1 Framework

The experiments were performed using a simulation framework [69] that accurately models generic and custom networks, including hierarchical networks such as XGFTs and dragonflies, at a flit level. The simulator provides a high level of customization and modularity, allowing the configuration of the desired model in minute detail. Several tests have been performed, on topologies interconnecting as few as 16 and as many as 1332 communicating nodes, both from the dragonfly and the XGFT family. The exact configurations used are detailed in Table 4.1 and Table 4.2.

Powers of two topologies		**Non-powers of two topologies**	
N	**Topology**	**N**	**Topology**
16	3 : 4,2,2 : 1,4,2		
32	3 : 4,4,2 : 1,4,4		
64	3 : 8,4,2 : 1,8,4		
128	3 : 8,8,2 : 1,8,8	243	3 : 9,9,3 : 1,9,9
256	4 : 8,4,4,2 : 1,8,4,4	385	3 : 11,7,5 : 1,11,7
512	4 : 8,8,4,2 : 1,8,8,4	625	4 : 5,5,5,5 : 1,5,5,5
1024	4 : 16,8,4,2 : 1,16,8,4	1001	3 : 13,11,7 : 1,13,11

Table 4.1 – Benchmarked XGFT topological configurations. Each topology is specified using the the $L : M_1,M_2,...,M_L : W_1,W_1,W_L$ notation introduced by [76]. L is the number of switching layers, the list of M parameters defines the hierarchy values H_1 to H_L while the list of W parameters characterizes the bandwidth available at each level in the tree.
Table included from [85] ©2013 IEEE

4.5.2 Parameters and metrics

The switch architecture that was chosen was that of an input-output-buffered switch with 4 KB of buffer space per port per direction. The links had a bandwidth of 10 Gbit/second and a latency of 50 nanoseconds. Credit based flow control was used as well as wormhole switch traversal. The messages had a constant size which, depending on the scenario, took

N	Topology	N	Topology	N	Topology
72	2,4,2	342	3,6,3	1056	4,8,4
42	2,3,2	240	3,5,3	812	4,7,4
110	2,5,2	462	3,7,3	1332	4,9,4

Table 4.2 – Benchmarked dragonfly topological configurations. Each topology is specified using the p,a,h notation introduced by [54]. p represents the number of host nodes connected to each switch, a is the number of switches in the fully connected mesh at level 1 of the dragonfly while h is the number of ports connecting each switch to other groups. The hierarchy parameters are thus $H_1 = p$, $H_2 = a$ and $H_3 = a \cdot h + 1$. Each column corresponds to a target topology and within it the first row corresponds to the balanced version, the second to the under-provisioned configuration and the third to the over-provisioned configuration.
Table included from [85] ©2013 IEEE

values between 4 bytes and 32 KB, while the flit size was fixed to 64 B. Each message was acknowledged by a single-flit acknowledgment message to trigger the injection of the next message from that task. The routing algorithm used for the XGFTs was destination-modulo-radix (d-mod-k) [61], whereas for the dragonfly we employed shortest-path routing with virtual-channel-based deadlock avoidance [54].

The traffic pattern is that of a single all-to-all collective exchange performed by a set of tasks, one on every communicating node. The exchange is split up in consecutive phases in each of which every node sends a single message and in which every node receives a single message and acknowledges it. No explicit synchronization or separation of the phases is enforced. The exact structure of each phase (the set of (source, destination) pairs) is defined according to the following exchange models:

- the classical linear shift model (LIN),
- the generalized hierarchical exchange described in Section 4.4.1 (GHE),
- the load balanced generalized hierarchical exchange described in Section 4.4.2 (BGH),
- the latency balanced generalized hierarchical exchange described in Section 4.4.3 (LAT).

Note that we also evaluated the performance of the binary XOR exchange, in the cases where it was applicable (topologies with a number of communicating nodes equal to a power of two), and we have always obtained the same completion time as that obtained when using the generalized hierarchical exchange.

The metric used to quantify the performance of a certain design is that of the time necessary for the traffic pattern to complete.

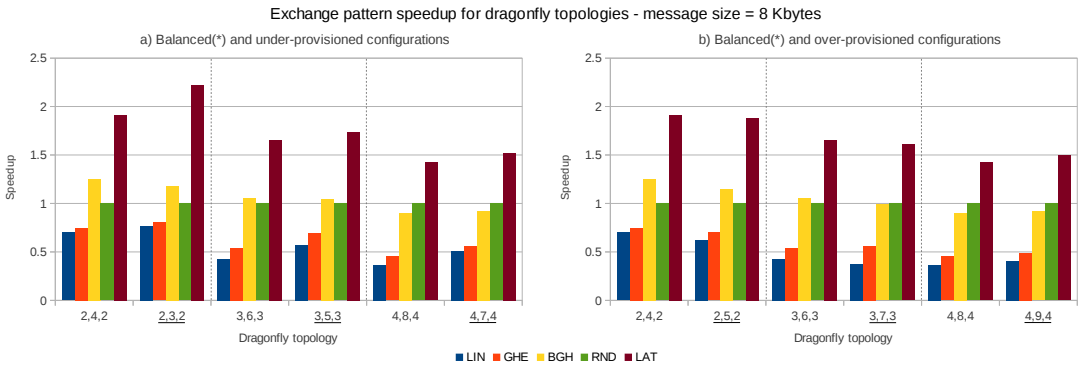


Figure 4.8 – Speedup of exchange completion (where the base completion time is that of RND) for several dragonfly topologies, for 8 KB messages. Higher bars correspond to lower completion time - thus, higher is better. The simple hierarchical exchange has a performance similar to that of LIN, the load balanced hierarchical exchange is significantly better, but on par with RND, while the latency balanced hierarchical exchange is more than 1.5 times faster than either of the latter two and more than 3 times faster than LIN. (*) Note: *Balanced* [54] in this context refers to dragonfly configurations satisfying $a = 2 \cdot p$, and is not related to the *load balanced* or *latency balanced* hierarchical exchanges. Under- and over-provisioned configurations are underlined.
Figure included from [85] ©2013 IEEE

4.5.3 Results

First, we will analyze the performance of the exchange patterns on XGFT interconnects. All the topologies that we tested provide sufficient bandwidth capacity to accommodate any phase where each source sends to a distinct destination, regardless of the exchange pattern, in a contention free manner. As such, all exchange patterns should achieve optimal completion times in the idealized case of 0 link latency; we verified this through simulation.

However, when dealing with realistic conditions, the advantages of a hierarchical exchange become obvious. As the speedups illustrated in Figure 4.6 show, for messages of 8 KB, the hierarchical exchanges exhibit completion times between 1.5 and 2.5 times smaller than those of the linear shift pattern. As there is enough capacity in the network to accommodate all the GHE traffic, there is no differentiation between GHE and BGH.

As the message size increases, the advantages of the hierarchical exchange also become more significant. Figure 4.7 clearly shows how for small messages there is practically no difference between LIN and the two hierarchical exchanges but as the message size grows, both GHE and BGH achieve progressively smaller completion times, up to more than 3 times better (for 32 KB messages) than LIN.

Next, we benchmark exchange performance in dragonfly topologies. We start once more by comparing the completion time of the different approaches for a fixed message size (8 KB) but for different network sizes. We chose three *balanced* dragonflies [54] (dragonflies that

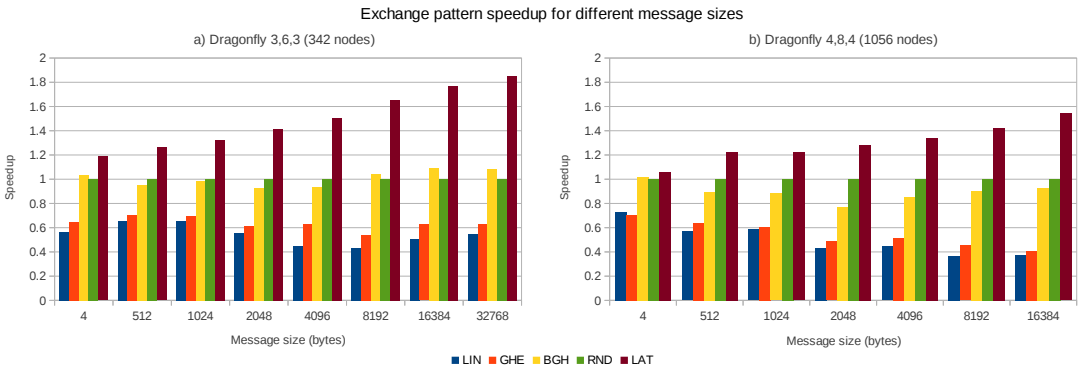


Figure 4.9 – Speedup of exchange completion (where the base completion time is that of RND) for a fixed dragonfly topology interconnecting a) 342 nodes or b) 1056 nodes, for different sizes of the exchanged messages. Higher bars correspond to lower completion time - thus, higher is better. The performance benefits that we were able to associate to the use of the hierarchical exchanges are amplified by increasing the size of the messages.
Figure included from [85] ©2013 IEEE

have the properties $a = 2 \cdot p$ and $h = p$) along with their unbalanced variants, over-provisioned ones ($a > 2 \cdot p$) and under-provisioned ones ($a < 2 \cdot p$). In addition to the linear shift exchange, we added a new benchmark, a linear shift exchange executed on a random placement of tasks (RND), to compare the performance of our exchange pattern against a task-placement proposal leading to the best known performance for an all-to-all exchange up to now [16].

The results for balanced, under-provisioned and over-provisioned dragonfly configurations, are shown in Figure 4.8. As we explained in the previous section, the fact that in the basic generalized hierarchical exchange all messages sent by sources in a common layer $l-1$ subnetwork in a layer l phase are sent to destinations that share a common layer $l-1$ subnetwork as well causes important congestion on the dragonfly links at layer l. This is confirmed by the simulation, as GHE now performs similarly to LIN (albeit slightly better). The load balanced exchange outperforms them both by a significant margin, but is only similar in performance to RND (slightly better for smaller dragonflies and for under-provisioned bandwidth, slightly worse for the other cases). Finally, the latency balanced exchange outperforms both RND and BGH by a significant margin (almost twice as efficient), but the advantages of this approach become slightly less important as the network grows. The margin is also more important when the bandwidth is under-provisioned and less important when the bandwidth is over-provisioned.

We performed a final set of tests to determine the influence of the message size. We analyzed this influence in the context of two balanced dragonfly topologies, the 342-node and the 1056-node dragonfly. The results are presented in Figure 4.9. We can see that once more, the larger the messages used in the exchange, the more performance can be gained relative to current state of the art methods by using the proposed generalized hierarchical exchanges, in particular, the latency balanced generalized hierarchical exchange (LAT).

We have shown that in hierarchical topologies, be they tree-shaped or dragonflies, using our proposed hierarchical exchanges yields significant performance increase: typically, more than a twofold and up to a threefold reduction in completion time for XGFT networks and a completion time for dragonfly networks that is between 1.2 and 1.8 times smaller. In addition to the gain in efficiency, the traffic patterns we propose also exhibit a contained utilization of the network, making them much more suitable for cohabitation with other application or IO traffic.

4.6 Summary

The goal of this chapter was to show that, in the broad context of hierarchical interconnection network architectures, there are important advantages to performing personalized all-to-all collective message exchanges in a hierarchy-aware way. To this end, we first showed that many popular interconnect designs can be viewed as hierarchical topologies and we introduced a unified hierarchical addressing scheme for nodes and tasks that applies equally to networks as different as fat trees and dragonflies. We then introduced the concept of hierarchical exchanges and we qualitatively argued that such an exchange would exhibit several performance enhancing qualities. Leveraging the unified addressing scheme, we proceeded to define three novel exchange patterns that we mathematically proved to satisfy the prerequisites of a hierarchical exchange. Finally we benchmarked our own approaches against state-of-the-art algorithms and confirmed that in practical scenarios the exchange patterns that we proposed perform consistently better than currently employed methods, often by a large margin, with completion times of the exchange as much as 3 times faster in extended generalized fat trees and up to 1.8 times faster in dragonflies.

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

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.

This chapter is based on the article [83] PRISACARI, B., RODRÍGUEZ, G., HEIDELBERGER, P., CHEN, D., MINKENBERG, C., AND HOEFLER, T. Efficient task placement and routing of nearest neighbor exchanges in dragonfly networks. In *The 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC'14, Vancouver, BC, Canada - June 23 - 27, 2014* (2014), ACM, pp. 129–140. https://doi.org/10.1145/2600212.2600225

5.1 Motivation

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 [10]) and Kim et al. [54]. 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 [48]. 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 [40].

Any deterministic traffic pattern may exhibit poor performance if the computation is mapped unfavorably to a Dragonfly [8, 16, 32, 54]. For example, we show in this chapter 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 [32, 54], topology [32] and task mapping [16] 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 chapter, 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.

5.2 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 [10] and in the Cray Cascade [27] 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 [54]. 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 5.1 illustrates the topological layout of a Dragonfly network and the meaning of the (p,a,h) parameters.

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

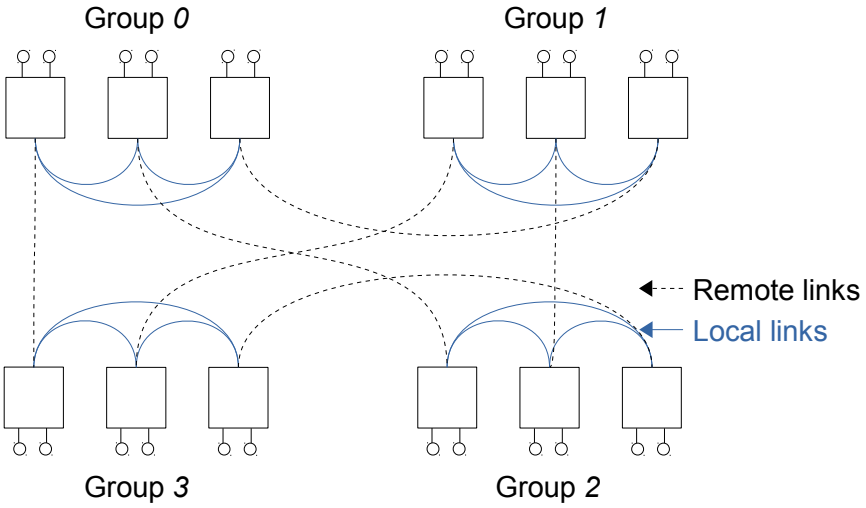


Figure 5.1 – Example of a ($p = 2, a = 3, h = 1$) Dragonfly topology. Every switch is connected to $p = 2$ nodes, there are $a = 3$ switches in every group and every switch has $h = 1$ links connecting it to switches outside its group.
Figure included from [83]

However, this lack of shortest path diversity can lead to an extreme degradation in performance for certain adversarial traffic patterns [16,32]. One option to alleviate this degradation is to use Valiant's algorithm [106]. 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 [22]).

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 [54], 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 [10]. *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 minimal 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 chapter, 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 [48]. Relevant examples are Cactus [35], which uses a grid decomposition to solve Einstein's equations to study astrophysical phenomena, GTC [62] (Gyrokinetic Toroidal Code), which solves the gyrophase-averaged Vlasov-Poisson equations on a 3D toroidal domain, LBMHD [109] (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 [16] 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 [46] 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.

5.3 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 5.3.2 where we introduce the main notations and then skip directly to Section 5.3.5.

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.

$$\hat{t}_{sd} = t_{sd} / \sum_{d} t_{sd} \tag{5.1}$$

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.

$$\Delta_{ij}^{\text{R,direct}} = \sum_{s \in G_i} \sum_{d \in G_j} \hat{t}_{sd} \cdot B_{\text{P,eff}}^{s}. \tag{5.2}$$

The demand $\Delta_{ij}^{\text{R,indirect}}$ 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.

$$\begin{aligned} \Delta_{ij}^{\text{R,indirect}} = & \sum_{p \neq i,j} \sum_{s \in G_i} \sum_{d \in G_p} \hat{t}_{sd} \cdot P(i \xrightarrow{j} p) \cdot B_{\text{P,eff}}^{s} \\ & + \sum_{p \neq i,j} \sum_{s \in G_p} \sum_{d \in G_j} \hat{t}_{sd} \cdot P(p \xrightarrow{i} j) \cdot B_{\text{P,eff}}^{s} \end{aligned} \tag{5.3}$$

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 5.4.3.

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, $B_{\mathrm{P,eff}}$ 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 $B_{\mathrm{P,eff}}$.

$$\begin{aligned}\delta_{ij}^{\mathrm{R,direct}} &= \Delta_{ij}^{\mathrm{R,direct}} / B_{\mathrm{P,eff}} \\ \delta_{ij}^{\mathrm{R,indirect}} &= \Delta_{ij}^{\mathrm{R,indirect}} / B_{\mathrm{P,eff}}\end{aligned} \tag{5.4}$$

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. (5.5) for direct routing and Eq. (5.6) for indirect routing.

$$B_{\mathrm{P,eff}}^{\mathrm{direct}} \leq \min\left(B_P, \frac{B_L}{\max\limits_{\mathrm{Llinks}}(\delta^{\mathrm{L,direct}})}, \frac{B_R}{\max\limits_{\mathrm{Rlinks}}(\delta^{\mathrm{R,direct}})}\right) \tag{5.5}$$

$$B_{\mathrm{P,eff}}^{\mathrm{indirect}} \leq \min\left(B_P, \frac{B_L}{\max\limits_{\mathrm{Llinks}}(\delta^{\mathrm{L,indirect}})}, \frac{B_R}{\max\limits_{\mathrm{Rlinks}}(\delta^{\mathrm{R,indirect}})}\right) \tag{5.6}$$

Eq. (5.5) and (5.6) coupled to the load formulations (Eq. (5.2) and (5.3)) and to a particular traffic demand matrix allow us to predict network performance for arbitrary workloads in arbitrary dragonflies.

5.3.2 Formal description of targeted workloads

The workloads we focus on in this chapter 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. 5.2 for a two-dimensional application domain.

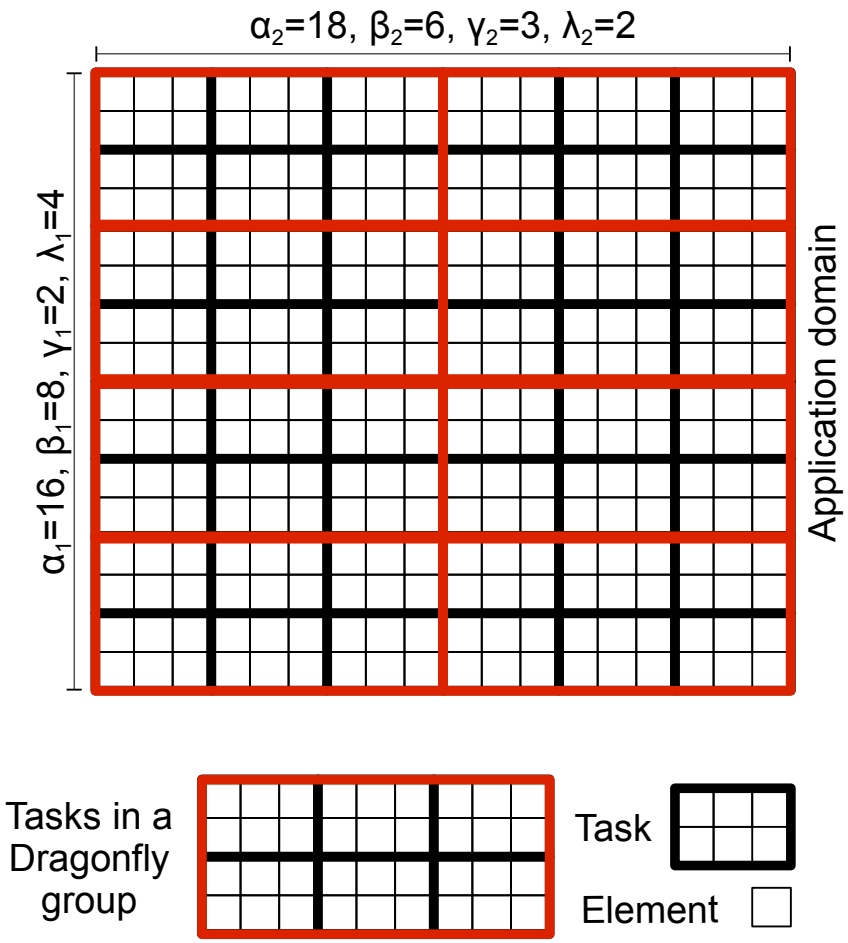


Figure 5.2 – Application domain for a two-dimensional nearest neighbor exchange. The domain admits an intrinsic decomposition into identical elements (the small squares in the figure) along the axes of the Cartesian domain. This decomposition is characterized by the two-dimensional vector α which in 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, $\beta = (8,6)$).
Figure included from [83]

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 $v^k_{x,y}$ which

takes a value of 1 if tasks corresponding to x and y are neighbors along the k dimension and 0 otherwise, as follows:

$$v_{x,y}^{k} = \begin{cases} 1 & \text{if } (\beta_k + x_k - y_k) \bmod \beta_k = 1 \\ & \text{and } \forall i \neq k, x_i = y_i \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

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 $|\gamma| = \Pi_{k=1}^{d} \gamma_k$ 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 $|\lambda| = \Pi_{k=1}^{d} \lambda_k$ 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 [27]. 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 scope.

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 [82].

5.3.3 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

$$\mu_k = \mu \cdot \Pi_{i=1, i\neq k}^{d} \frac{\alpha_i}{\beta_i} = \mu \cdot \frac{|\alpha|}{|\beta|} \cdot \frac{\beta_k}{\alpha_k}. \tag{5.8}$$

5.3.4 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 5.3.1 the value of the latter is

$$P(i \xrightarrow{p} j) = \frac{1}{ah-1}. \tag{5.9}$$

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 (5.7), 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

$$\begin{aligned} m_k &= \frac{\mu_k \cdot \mathbb{1}_{\beta_k>1}}{2\sum_{i=1}^{d} \mu_i \cdot \mathbb{1}_{\beta_i>1}} \\ &= \frac{\mu \cdot \frac{|\alpha|}{|\beta|} \cdot \frac{\beta_k}{\alpha_k} \cdot \mathbb{1}_{\beta_k>1}}{2\sum_{i=1}^{d} \mu \cdot \frac{|\alpha|}{|\beta|} \cdot \frac{\beta_i}{\alpha_i} \cdot \mathbb{1}_{\beta_i>1}} \\ &= \frac{\beta_k \cdot \mathbb{1}_{\beta_k>1}/\alpha_k}{2\sum_{i=1}^{d} \beta_i \cdot \mathbb{1}_{\beta_i>1}/\alpha_i}. \end{aligned} \tag{5.10}$$

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:

$$\hat{t}_{sd} = \sum_{i=1}^{d} m_i \cdot v_{s,d}^{i} \cdot (1 + \mathbb{1}_{\beta_i=2}) \tag{5.11}$$

5.3.4.1 Cartesian placement

The case where the tasks assigned to any fixed group form a Cartesian sub-domain is of interest because it allows keeping 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. (5.2) and (5.3).

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_k^g satisfies $0 \leq x_k^g < \lambda_k, \forall 1 \leq k \leq d$. Similarly to the neighborhood relationship for tasks defined by (5.7) we can define a neighborhood relationship for groups as

$$\tilde{v}_{i,j}^{k} = \begin{cases} 1 & \text{if } (\lambda_k + x_k^i - x_k^j) \bmod \lambda_k = 1 \\ & \text{and } \forall l \neq k, x_l^i = x_l^j \\ 0 & \text{otherwise.} \end{cases} \tag{5.12}$$

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. (5.13) shows the resulting direct routing demand.

$$\begin{aligned} \delta_{ij}^{\text{R,direct}} &= \Delta_{ij}^{\text{R,direct}} / B_{\text{P,eff}} \\ &= \sum_{s \in G_i} \sum_{d \in G_j} \hat{t}_{sd} \\ &= \sum_{k=1}^{d} (1 + \mathbb{1}_{\lambda_k=2}) \cdot |\gamma|/\gamma_k \cdot m_k \cdot \tilde{v}_{i,j}^{k} \end{aligned} \tag{5.13}$$

In the case of indirect routing, we start by expressing $\delta^{\text{R,indirect}}$ from Eq. (5.3) and (5.4).

$$\begin{aligned}\delta_{ij}^{\text{R,indirect}} &= \sum_{p\neq i,j}\sum_{s\in G_i}\sum_{d\in G_p}\hat{t}_{sd}\cdot P(i\xrightarrow{j}p)\\ &\quad+\sum_{p\neq i,j}\sum_{s\in G_p}\sum_{d\in G_j}\hat{t}_{sd}\cdot P(p\xrightarrow{i}j)\\ &=\frac{1}{ah-1}\cdot\left(\sum_{p\neq i,j}\sum_{s\in G_i}\sum_{d\in G_p}\hat{t}_{sd}\right.\\ &\quad\left.+\sum_{p\neq i,j}\sum_{s\in G_p}\sum_{d\in G_j}\hat{t}_{sd}\right)\end{aligned} \tag{5.14}$$

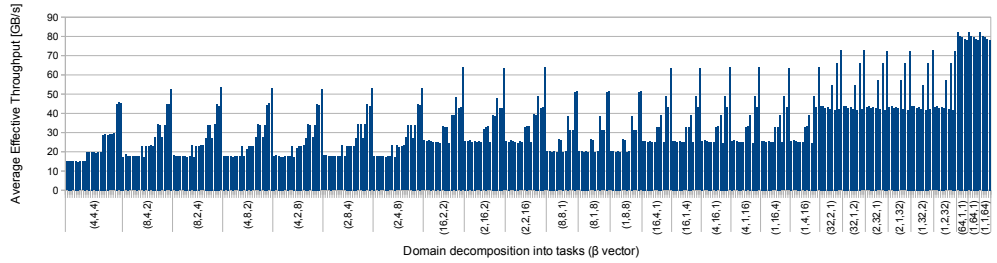
The two triple sums each evaluate to the same value for reasons that are linked to the symmetry of the communication pattern. Indeed, the amount of traffic sent by tasks in a group i to tasks in all groups $p\neq i,j$ (first triple sum) is equal to the aggregate external nearest neighbor traffic of i 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. (5.15) shows the resulting indirect routing demand.

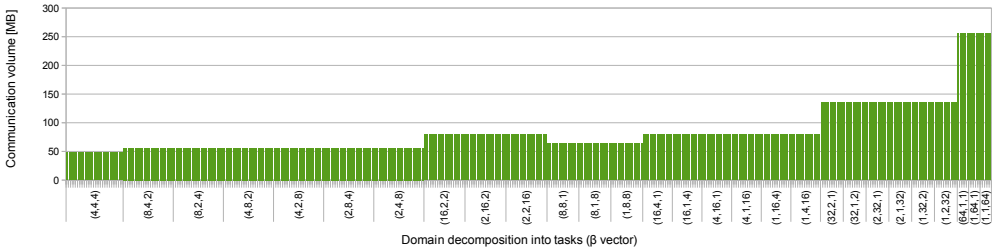
$$\begin{aligned}\delta_{i,j}^{\text{R,indirect}} &= (4\sum_{k=1}^{d}(|\gamma|/\gamma_k\cdot m_k\cdot\mathbb{1}_{\lambda_k>1})\\ &\quad-2\delta_{i,j}^{\text{R,direct}})/(ah-1)\end{aligned} \tag{5.15}$$

To be able to now derive performance estimations (via Eq. (5.6)) we maximize the demand. For direct routing, using the fact that Eq. (5.12) implies that task domains mapped to two groups can only be neighbors in at most one direction and Eq. (5.13), we obtain the bound in Eq. (5.16)

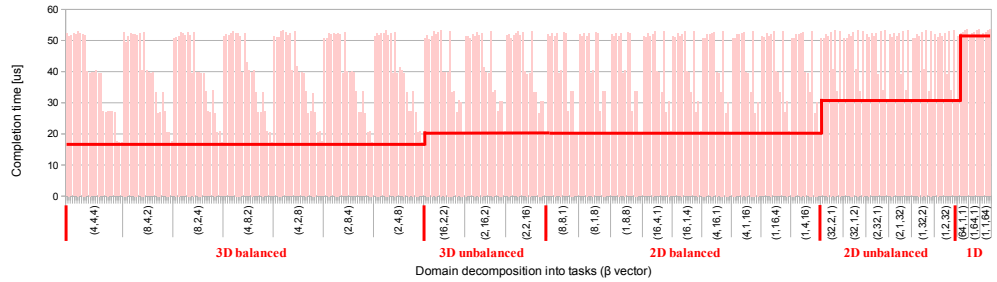
$$\max_{i,j}(\delta_{i,j}^{\text{R,direct}})=\max_{k}((\mathbb{1}_{\lambda_k>1}+\mathbb{1}_{\lambda_k=2})\cdot|\gamma|/\gamma_k\cdot m_k) \tag{5.16}$$



(a) Throughput (higher is better)



(b) Communication Volume



(c) Completion time (lower is better)

Figure 5.3 – 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. *Figure included from [83]*

Similarly, Eq. (5.17) shows the bound obtained when maximizing indirect routing induced demand

$$\max_{i,j}(\delta^{\mathrm{R,indirect}}_{i,j}) = 4\sum_{k=1}^{d}(|\gamma|/\gamma_k \cdot m_k \cdot \mathbb{1}_{\lambda_k>1})/(ah-1). \tag{5.17}$$

As expressed by Eq. (5.6), maximizing performance is equivalent to minimizing the maximum

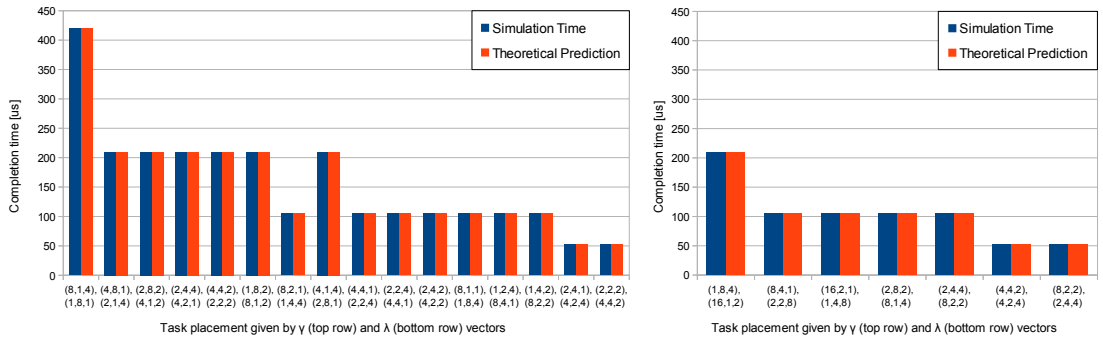


Figure 5.4 – Comparison of the completion time (estimated and measured) for a balanced dragonfly topology under direct routing for a collection of all possible task-placements γ for a 3-dimensional β = (8,8,4) nearest neighbor exchange of 256 tasks (Figure a)) and a β = (16,8,8) nearest neighbor exchange 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. *Figure included from [83]*

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).

5.3.4.2 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. (5.5), 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 [45], 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 has the most balls will have. As proven in [34, 88], for large balls-to-bins ratios, this is given by Eq. (5.18).

$$\frac{n_{\text{balls}}}{n_{\text{bins}}} \cdot \frac{\log n_{\text{bins}}}{\log\log n_{\text{bins}}} \tag{5.18}$$

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

$$\frac{|\gamma|}{|\lambda|} \cdot \frac{\log|\lambda|}{\log\log|\lambda|} \cdot m_k. \tag{5.19}$$

Summing over all dimensions and both directions we obtain

$$\begin{aligned} \max_{\text{Rlinks}}(\delta^{\text{R,direct}}) &= 2\sum_{k=1}^{d} \frac{|\gamma|}{|\lambda|} \cdot \frac{\log|\lambda|}{\log\log|\lambda|} \cdot m_k \\ &= \frac{|\gamma|}{|\lambda|} \cdot \frac{\log|\lambda|}{\log\log|\lambda|} \cdot 2\sum_{k=1}^{d} m_k \\ &= \frac{|\gamma|}{|\lambda|} \cdot \frac{\log|\lambda|}{\log\log|\lambda|}. \end{aligned} \tag{5.20}$$

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 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.

5.3.5 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. (5.16) and (5.5)) that effective

injection bandwidth is limited by

$$B_{\mathrm{P,eff}} \leq \frac{B_R}{\max_{k,\gamma,\beta}((\mathbb{1}_{\lambda_k>1} + \mathbb{1}_{\lambda_k=2}) \cdot |\gamma|/\gamma_k \cdot m_k)}. \tag{5.21}$$

By substituting m_k using Eq. (5.10) we obtain the optimization criterion for this configuration: optimal performance is obtained by minimizing

$$\max_{k,\gamma,\beta} \left[(\mathbb{1}_{\lambda_k>1} + \mathbb{1}_{\lambda_k=2}) \cdot \frac{|\gamma|}{\gamma_k} \cdot \frac{\beta_k \cdot \mathbb{1}_{\beta_k>1}}{\alpha_k} \right]. \tag{5.22}$$

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. (5.17) and (5.6)) that effective injection bandwidth is limited by

$$B_{\mathrm{P,eff}} \leq \frac{B_R}{\max_{\gamma,\beta}(4\sum_{k=1}^{d}(|\gamma|/\gamma_k \cdot m_k \cdot \mathbb{1}_{\lambda_k>1})/(ah-1))}. \tag{5.23}$$

By substituting m_k using Eq. (5.10) we obtain the optimization criterion for this configuration: optimal performance is obtained by minimizing

$$\max_{\gamma,\beta} \left[\sum_{k=1}^{d} \left(\mathbb{1}_{\lambda_k>1} \cdot \frac{|\gamma|}{\gamma_k} \cdot \frac{\beta_k \cdot \mathbb{1}_{\beta_k>1}}{\alpha_k} \right) \right]. \tag{5.24}$$

For random placement and direct routing, we have shown (Eq. (5.20) and (5.5)) that effective injection bandwidth is limited by

$$B_{\mathrm{P,eff}} \leq \frac{B_R}{\max_{\gamma,\beta} \left[\frac{|\gamma|}{|\lambda|} \cdot \frac{\log|\lambda|}{\log\log|\lambda|} \right]}. \tag{5.25}$$

Optimal performance is thus obtained by minimizing

$$\max_{\gamma,\beta} \left[\frac{|\gamma|}{|\lambda|} \cdot \frac{\log|\lambda|}{\log\log|\lambda|}. \right] \tag{5.26}$$

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.

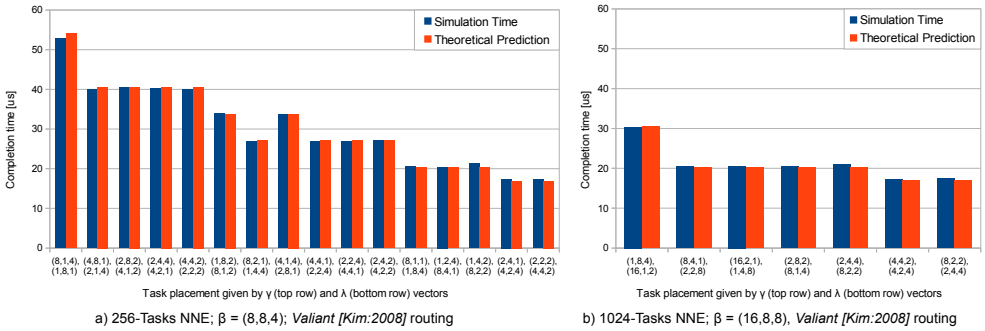


Figure 5.5 – Comparison of the completion time (estimated and measured) for a balanced dragonfly topology under *Valiant [Kim:2008]* indirect routing for a collection of all possible task-placements γ for a 3-dimensional β = (8,8,4) nearest neighbor exchange of 256 tasks (Figure a)) and a β = (16,8,8) 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.
Figure included from [83]

5.4 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 [69]. 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 5.3.2. 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.

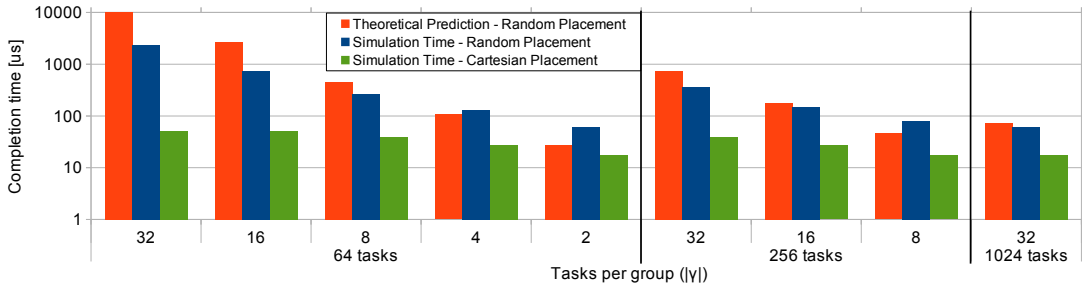


Figure 5.6 – Comparison of the completion time (estimated and measured) for a balanced dragonfly topology under direct routing and random task placement for three 3-dimensional nearest neighbor exchanges ($\beta \in \{(4,4,4),(8,8,4),(16,8,8)\}$). The x axis lists the 3 exchanges from left to right (identifying 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.
Figure included from [83]

5.4.1 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. (5.8) 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. 5.3; the measured effective throughput is shown in Figure 5.3a the total volume of exchanged data is shown in Figure 5.3b and the effective completion time is shown in Figure 5.3c.

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 5.3c). 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 on this measure, our conclusions would have been the exact opposite. Indeed, as shown in Figure 5.3a, 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 5.3b) 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.

5.4.2 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 β = (8,8,4) while for the latter we will consider β = (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 5.3. 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 5.4 for direct routing and in Figure 5.5 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 5.4) and indirect (Figure 5.5) 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 5.6), 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

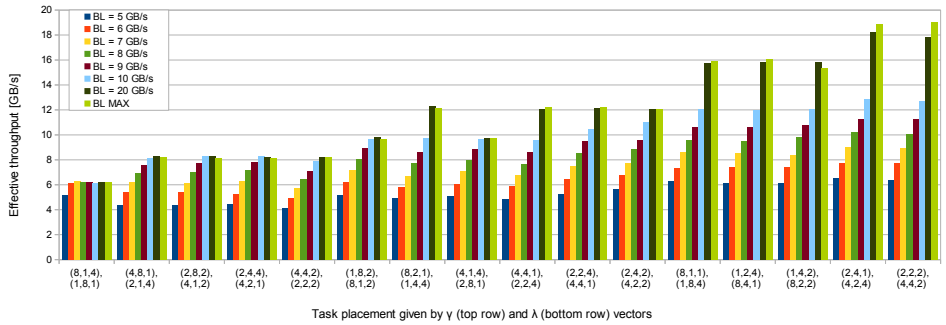


Figure 5.7 – Effective throughput upper bound for nearest neighbor exchange among $|\beta| = 256$ tasks mapped on a 3-dimensional domain of size $\beta = (8,8,4)$ for increasingly higher local link bandwidths. The routing used is indirect routing and the placement is Cartesian: the first line of the 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.
Figure included from [83]

captured faithfully and the decision of which configuration suits a particular workload best can still be taken based solely on the model.

Finally, Figure 5.6 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.

5.4.3 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 5.7 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.

5.4.4 Guidelines for application-network joint configuration and design

In Section 5.3.5 we have summarized the analytical performance model introduced in this chapter, 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. (5.22), Eq. (5.24) and Eq. (5.26) 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. (5.21), Eq. (5.23) and Eq. (5.25) 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 [54] (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.

5.5 Summary

In this chapter we analyzed 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 Randomizing task placement and route selection do not randomize traffic (enough).

Dragonflies are one of the most promising topologies for the Exascale effort for their scalability and cost. Dragonflies achieve very high throughput under uniform traffic, but have a pathological behavior under other regular traffic patterns, some of them very common in HPC applications, such as the multi-dimensional stencil communication pattern or certain permutation patterns. A recent study showed that randomization of task placement greatly improves the performance of these pathological traffic patterns by increasing the similarity of the load they induce to a uniformly distributed load.

In this chapter we provide a theoretical model that is able to predict the expected performance of a generic dragonfly network under uniform traffic and characterize performance-optimal, minimal cost dragonflies. We then match the predictions of this model with the performance obtained through the detailed simulation of a wide range of dragonfly configurations. In these same scenarios, we explore the performance of other non-uniform traffic patterns and investigate the impact of randomization techniques based on both task placement and indirect routing. For these previously unexplored traffic patterns, we obtain similar results to those obtained in previous works for the multi-dimensional stencil communication pattern: randomizing task placement and/or path choice is effective in improving the performance of pathological workloads. However, we also show that neither uniformization technique is able to close the gap between the performance of these traffic patterns and the ideal performance of uniform random traffic, leaving significant room for improvement (best achieved performance is only roughly 50% of uniform performance).

This chapter is based on the article [84] PRISACARI, B., RODRÍGUEZ, G., JOKANOVIC, A., AND MINKENBERG, C. Randomizing task placement and route selection do not randomize traffic (enough). *Design Automation for Embedded Systems 18*, 3-4 (2014), 171–182. https://doi.org/10.1007/s10617-014-9133-x ©2014 Springer

6.1 Motivation

Commercial and high performance computing (HPC) systems with a number of nodes in the tens to hundreds of thousands require high-bandwidth low-latency interconnection networks to achieve high performance. The dragonfly network topology [54] is a promising choice to achieve this level of scalability as it exhibits a low diameter, an attractive cost factor thanks to its hierarchical structure (allowing the efficient use of a mix of electrical/optical communication channels), and an excellent performance under uniform traffic patterns.

However, several works [16,32,54] have pointed out that, in contrast to the high uniform traffic performance, adversarial traffic patterns exist that lead to extremely poor performance. These include several popular exchange patterns used in HPC, such as the near-neighbor multi-dimensional exchange. Efforts have been made to adapt the routing [32,54], the topology [32] and the task placement [16] to improve the performance of these adversarial traffic patterns. These improvements rely on the idea of randomizing the (low performance) *regular* adversarial traffic pattern and in doing so increase its similarity to a (high performance) *uniform random* pattern. Randomization can be achieved in two main ways, either by using a randomized task placement or by using random indirect routing algorithms [106], the latter at the expense of longer paths.

In this chapter, we first provide a theoretical characterization of dragonfly performance under uniformly distributed load and validate the predictions of this model against benchmarks on a variety of dragonfly topologies. Using this model, we provide a theoretical justification of the manner in which dragonfly networks should be topologically configured in practice for maximum performance at minimum cost. We then introduce two non-uniform workloads, one of which is adversarial to the dragonfly, and evaluate experimentally whether randomization techniques are effective in enhancing their performance to a level close to that of uniform random traffic. By extending the evaluation in [16] to these previously not evaluated workloads we effectively show the impact of each randomization technique on system performance and expose the lingering gap to the ideal performance achievable under true uniform load.

6.2 Background

Dragonfly topologies [54] are highly scalable direct networks with a good cost-performance ratio, used for example in the PERCS [10] and the Cascade [27] interconnects and one of the main options for future Exaflop/s machines.

A dragonfly is a two-level hierarchical network, where the low-radix switches at the first level form a virtual high-radix switch. These first-level groups are generally interconnected in a fully-connected mesh, as is the case in the PERCS interconnect [10], but could also be interconnected in other ways (e.g., as flattened butterflies, as is the case in the Cascade interconnect [27]). These virtual high-radix switches are connected to form a fully-connected graph of first-level groups [54] at the second level. The ports that the virtual high-radix switches

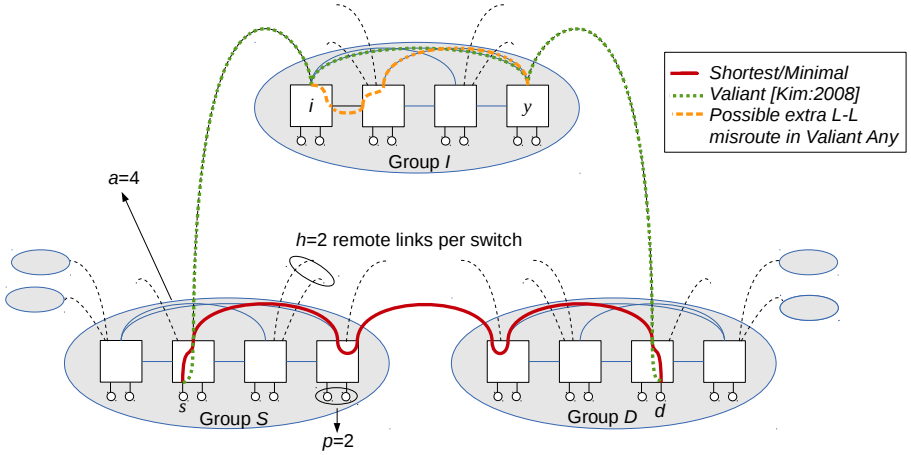


Figure 6.1 – Three of the groups of a balanced dragonfly system with parameters $p = 2$, $a = 4$ and $h = 2$. The small circles represent the nodes of the system, the squares represent the real low-radix switches while the large blue ellipses represent the groups of the dragonfly. Every switch is connected to p nodes and has $a - 1$ links towards every other switch in its group and h links towards switches in other groups. The 3 routing strategies considered here are also illustrated via a possible path between nodes s and d for each routing approach. The contiguous thick red line represents the minimal path. The green dashed line represents one of the possible paths under *Valiant [Kim:2008]* routing, namely the one using group I as intermediate. Finally, the orange dashed line represents one of the possible extra L-L misroutes in the intermediate group under *Valiant Any*.
Figure included from [84] ©2014 Springer

use to connect to the other virtual switches are distributed across the low-radix real switches that make up the virtual switch.

The dragonflies we consider 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 for these parameters, ideal throughput can be achieved for uniform traffic.

In a dragonfly, minimal paths between pairs of nodes are unique. The longest possible minimal path (Figure 6.1) is composed of a traversal of a *local* intra-group (L) link in the group of the source node to get to the switch that has the *global* or *remote* (R) link towards the destination group, a traversal of that remote link and a second *local* link traversal in the destination group to get to the switch directly connected to the destination node.

This scarcity of minimal paths can lead to an extreme degradation in performance for certain adversarial traffic patterns. For example, should a large proportion of traffic originating in the nodes of a group be destined to nodes that all belong to a unique other group, the unique remote link connecting the two groups will need to accommodate all this aggregate

traffic under minimal routing, effectively becoming a very strong bottleneck and leading to severe performance degradation. One option to alleviate this degradation is to use Valiant's algorithm [106]. 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 traffic distribution, at the expense of longer paths, doubling the load under random traffic itself [22]. In the previous example, such an approach would lead to traffic being sprayed across all remote links exiting the source group and the previous bottleneck would no longer appear. The longer paths in Valiant routing require the use of additional virtual channels to guarantee deadlock freedom. In particular, the Valiant routing variant for dragonflies as described in [54], which we will call *Valiant [Kim:2008]* (Figure 6.1), requires 3 virtual channels (one more than the 2 required for shortest path routing) for the L links and 2 virtual channels (instead of the 1 for shortest path routing) for the R links to guarantee deadlock freedom.

Valiant [Kim:2008] can be described as follows: when a source s in group S sends a message to destination d in 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 the first reachable switch, i, in group I. Once the packet arrives to this intermediate group I, the packet follows the unique minimal route from i to the destination d (requiring at most two L hops and one R hop). The longest path using this Valiant variant would visit the following link types: *LR-LRL*.

Another variant of Valiant routing could choose any switch (not just the first reachable one) as the intermediate misroute destination, thus potentially incurring an extra L hop within the intermediate group (see Figure 6.1), and requiring one extra VC for the L channels. The longest paths possible for this variant would visit the following link types: *LRL-LRL*. This routing variant, allowing for the extra intermediate group L-L misroute might negatively affect some traffic patterns due to an increased load on the L links, while helping others, by providing a route around a congested L link. This option has not been thoroughly studied in the literature mainly because of the cost of the extra VC. We will call this routing *Valiant Any*.

Another approach to achieve shifting the original nature of traffic towards a uniform random distribution and in doing so avoid low performance for adversarial traffic patterns does not rely on improving the amount of path diversity or the routing function, but rather on randomizing the task placement [16]. Bhatele et al. [16] analyze the performance achieved in dragonfly networks executing *stencil pattern exchanges* (a kind of adversarial traffic) under several task-placements and routing schemes (shortest path and Valiant). This work shows that a randomization of the task placement using shortest path routing achieves similar performance to a contiguous task placement using Valiant routing.

6.3 Theoretical Performance Bounds

In this section we will derive an expression for the theoretical throughput of a dragonfly network described by a set of parameters (p, a, h) [54], while also providing a detailed explanation and a proof for the relationships in [54] that p, a and h need to satisfy for the achievement of optimal throughput under uniform traffic to be possible.

To compute the throughput, let us analyze the load on arbitrary L and R channels. Assuming shortest path routing, a given L link can be traversed in a given direction by three types of messages. If the link in the given direction connects switch S to switch S', then the three types are:

1. messages from the p sources connected to S to the p destinations connected to S'
2. messages from the p sources connected to S to the $a \cdot p \cdot h$ destinations in groups reachable through R links on S'
3. messages from the $a \cdot p \cdot h$ sources in groups reachable through R links on S to the p destinations connected to S'

Given a set of messages traversing a link, originating on s sources and having a destination among a set of d destinations, in the context of uniform traffic, the probability that the link is occupied is $s \cdot d/N$, where $N = p \cdot a \cdot (a \cdot h + 1)$ is the total number of nodes. As such, the probability that an L link is occupied (similar to a probabilistic load) is, according to the enumeration of possible messages above:

$$P_L = (p^2 + p \cdot a \cdot p \cdot h + a \cdot p \cdot h \cdot p)/N = p^2 \cdot (2ah + 1)/N. \tag{6.1}$$

If we denote by B_p the bandwidth of each p link (and consequently the rate at which these messages enter the network) and by B_L the bandwidth of each L link, then the maximum throughput that an L link will be able to sustain is then the inverse of this probabilistic load multiplied by the ratio of the rates at which messages are transported and produced:

$$T_L = min(1, \frac{N}{p^2(2ah+1)} \cdot \frac{B_L}{B_p}). \tag{6.2}$$

In order to be able to achieve optimal throughput on a given L-link, T_L must be equal to 1. From this relationship we can derive a lower bound for the number a of switches in a group such that optimal throughput is achievable:

$$a \cdot B_L \geq 2 \cdot p \cdot B_p. \tag{6.3}$$

In the case where $B_L = B_p$ this becomes:

$$a \geq 2 \cdot p. \tag{6.4}$$

Under the same assumption of shortest path routing, a given R link that connects group G to group H can be traversed in that direction by a single type of message: a message sent by one of the $a \cdot p$ sources in G to one of the $a \cdot p$ destinations in H. As such the probability that an R link is occupied is $P_R = (a \cdot p) \cdot (a \cdot p)/N$ and the maximum throughput sustainable on that link will be:

$$T_R = min(1, \frac{N}{(a \cdot p)^2} \cdot \frac{B_R}{B_p}), \tag{6.5}$$

where B_R is the bandwidth of each R link. Optimal throughput on the R links implies $T_R = 1$ which further implies:

$$h \cdot B_R \geq (p - 1/a) \cdot B_p \tag{6.6}$$

In the case where $B_R = B_p$ and since h, p and a are integers this leads to:

$$h \geq p. \tag{6.7}$$

In summary, we have shown that in a dragonfly network the maximal throughput under uniform balanced traffic and shortest path routing is given by:

$$T = min(1, \frac{N}{p^2(2ah+1)} \cdot \frac{B_L}{B_p}, \frac{N}{(pa)^2} \cdot \frac{B_R}{B_p}), \tag{6.8}$$

and furthermore we have shown that the sufficient conditions for this throughput to be optimal are:

$$a \cdot B_L \geq 2 \cdot p \cdot B_p. \tag{6.9}$$

and

$$h \cdot B_R \geq (p - 1/a) \cdot B_p. \tag{6.10}$$

In the remainder of this chapter, we will call topologies which satisfy the equality conditions in these inequalities *balanced*, as introduced in [54]. We refer to any dragonfly network that satisfies at least one of the greater-than conditions (either $a \cdot B_L > 2 \cdot p \cdot B_p$. or $h \cdot B_R > (p - 1/a) \cdot B_p$), as an *over-provisioned* dragonfly.

The derivation presented here applies for shortest path routing. For Valiant routing variants, we can see that in general each packet will take paths of roughly twice the size of the shortest path,

thus increasing the load of every link by a factor of two, and reducing expected throughput to a half the expected throughput under shortest path routing for that particular topology. The exact performance of a Valiant routing depends on the actual strategy chosen to select the intermediate destinations (for instance, *Valiant [Kim:2008]* uses a choosing strategy that leads to slightly shorter paths than *Valiant Any*) so slight variations around this value are to be expected.

6.4 Experimental Results

6.4.1 Framework, parameters and metrics

The results presented in this section were obtained using a simulation framework [69, 70] that is able to accurately model and measure generic and custom networks, including dragonflies, at a flit level. The simulator provides a high level of customization and modularity, allowing the configuration of the desired model in minute detail. The switch architecture chosen was that of an input-output-buffered switch with 4 Kbytes of buffer space per port per direction per VC. The links had a bandwidth of 10 Gbit/second. Credit based flow control was used and the exchanged messages had a constant one-flit size of 64 bytes.

From the point of view of the destination distribution, three traffic patterns were benchmarked:

- uniform random traffic, where each task selects for each message a destination from the set of all the other tasks with equal probability;
- bit complement traffic, where each task t (where $0 \leq t < T$, T being the total number of tasks) selects as the destination of every message task $T - t - 1$;
- bit reversal traffic, where each task selects as the destination of every message the task that has an index whose binary representation is the reverse of the binary representation of the source task[1].

Each dragonfly node was assigned a single task and every such task sent messages at maximum rate (equal to the bandwidth of the link connecting the node to its corresponding switch). The assignment of tasks to nodes, or task placement, was performed in one of two ways: *contiguous* and *random*. The former signifies that every task was assigned to the node that has a node index equal to the task's index, while the latter signifies that the task index to node index bijection was selected in a uniform random fashion from all the possible such bijections. In both cases, the nodes themselves are indexed topologically: the nodes are numbered consecutively (starting with 0) one group at a time, and within a given group, one switch at a time.

[1] For topologies with a number of nodes that is not a power of two, we exercise the bit reversal pattern in the remaining nodes by recursively partitioning them in decreasing powers of two.

Three routing algorithms were used, all of them described in detail in Section 6.2:

- shortest-path routing;
- *Valiant [Kim:2008]* routing;
- *Valiant Any* routing;

For each approach, the corresponding virtual channel allocation scheme (described in [54], adapted for *Valiant Any* by simply adding an extra VC to the L links and the corresponding incremental transition) was enforced guarantee deadlock freedom.

For every experiment, the system was simulated for a fixed amount of time (100 milliseconds) that was chosen such that, in all cases, the throughput was estimated within a confidence interval of 1%. In cases where the system setup was dependent on a set of random numbers (as is the case for the uniform traffic pattern as well as for all traffic patterns under randomized task placement and/or *Valiant [Kim:2008]* and *Valiant Any* routing) each experiment was run 20 times, each time with a different seed for the random number generator. In these cases, results are presented together with error bars that illustrate the 3-sigma confidence interval. The metric was in all cases the average throughput, expressed as a percentage of the maximum cumulative injection throughput.

6.4.2 Results

First, we tested the performance of dragonfly networks under uniform traffic and compared it to the theoretical bounds we have derived in Section 6.3. As Figure 6.2 shows, the measured performance was almost exactly the same as the performance predicted theoretically. The small gap between the two stems from the slight imperfections in the simulated uniform distribution inherent to taking into account a finite number of messages. Note how the performance for the *balanced* configurations (highlighted in Figure 6.2 by a red border), i.e., networks that satisfy the minimum bandwidth requirements ($a = 2 \cdot p, h = p$), guaranteeing maximum throughput under uniform traffic when using shortest path routing (Section 6.3), achieved close to 95% of the injection throughput. *Over-provisioned* dragonflies, satisfying at least one greater-than inequality of the conditions: $a \geq 2 \cdot p, h \geq p$, and thus providing more bandwidth than necessary to sustain maximum throughput under uniform random traffic, did not show any advantage over the *balanced* configurations. As expected for uniform random traffic, task placement (*contiguous* and *random*) played no role in the performance achieved for a specific configuration, as the workload itself already led to a uniform distribution of load in the network.

We thus confirmed an important and well-known property of the dragonfly network, which is the high performance it exhibits when accommodating traffic that is uniformly distributed across the network. In the case of an uniform random workload, this distribution appears as

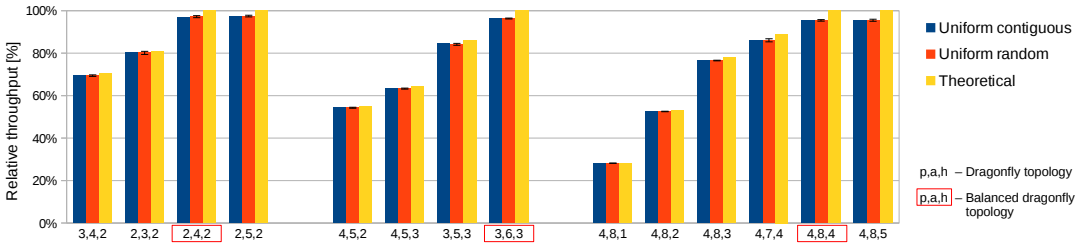


Figure 6.2 – Comparison between theoretically estimated and measured throughput for different dragonfly topologies under uniform traffic. The predictions generated by the theoretical model are confirmed by the measurements.
Figure included from [84] ©2014 Springer

an inherent consequence of the spatial (source-destination pairing) and temporal distribution of messages generated by the application. Other workloads do not have this property of inducing uniform load on the network. The question we aim to answer is whether the same high performance can be achieved for such traffic patterns by means of a randomization not inherent to the application itself but rather that comes from a random placement of the tasks on the network nodes and/or a random choice of the paths the messages will take through the network. For this purpose, we chose permutation traffic patterns, i.e., where the destination for a given source is always fixed and thus traffic is no longer spatially distributed in a uniform fashion. The two patterns we chose are bit complement and bit reversal traffic, which are almost at the two ends of the performance spectrum (in absence of external randomization via routing or placement). Specifically, the former not only concentrates traffic coming from a specific source to a specific destination, but also concentrates traffic across sets of sources to unique hot spots in the network, effectively becoming a worst case workload for the dragonfly. The latter on the other hand has the opposite property that a set of sources will always spread traffic as much as possible, approaching the uniform traffic spatial distribution to the maximal extent permitted by the permutation traffic restriction.

We first analyze the impact of workload load uniformization via task placement changes. Figures 6.3 and 6.4 show the performance of the two patterns for contiguous and randomized task placement. The theoretical estimation of the uniform traffic throughput is also presented, for reference.

What is immediately obvious is that task placement randomization does *not* lead to the same distribution of load in the network as that induced by uniform random traffic, as evidenced by the very large differences in measured throughput. Indeed, performance is between 40% and 60% worse than that of uniform traffic (typically around 50% worse for balanced dragonfly topologies). Nonetheless, an important fact that can be observed is that the uniform randomization of the task placement leads to a significant increase in performance for the pathological bit complement traffic (Figure 6.3) versus the contiguous placement case. For the bit reversal traffic (Figure 6.4), task placement randomization decreases performance slightly. The overall effect is that, through randomized task placement, any fixed permutation

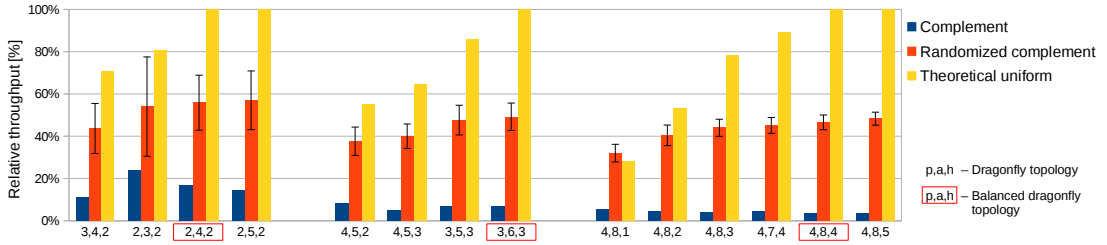


Figure 6.3 – Evaluation of the effect of the randomization of the task placement on the bit complement traffic pattern, which in absence of randomization is a pathological low performance case for the dragonfly. Randomized task placement is able to mitigate quite well the negative effects of the bit complement traffic, inducing a significant performance increase compared to contiguous task placement but nonetheless reaching only approximately 50% of the uniform traffic performance.
Figure included from [84] ©2014 Springer

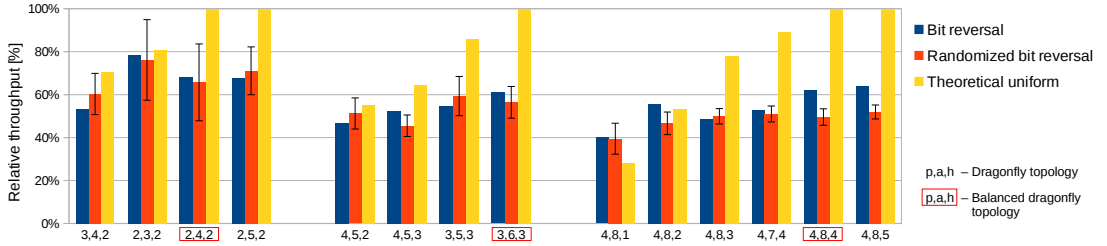


Figure 6.4 – Evaluation of the effect of the randomization of the task placement on the bit reverse traffic pattern, which in absence of randomization is a high performing permutation traffic on dragonfly networks. Randomized task placement induces a small decrease in performance compared to contiguous task placement, bringing throughput to approximately 50% of the uniform traffic performance.
Figure included from [84] ©2014 Springer

traffic (i.e., where a given source sends messages to a single destination and every destination receives messages from a single source) is reduced to an arbitrary permutation among the permutations with the same permutation cycle structure. Thus, although randomized task placement might lead to some loss of performance in the case of some efficient permutations, such as the bit reversal one, it will also lead with a high probability to the elimination of pathological cases.

Looking at what the minimal requirements are in terms of network structure (in the sense of whether the same "balancing" conditions apply as before), we notice that it is not always necessary to have a balanced dragonfly in order to reach maximum throughput. Compared to the uniform traffic case, where even small deviations from the balanced parameter values led to decreases in performance of $10-20\%$, here we can see that the performance impact of using under-provisioned dragonflies is much smaller, even inexistent for certain network scales and not too large a degree of under-provisioning. Indeed, for example for the (4,8,4) case, the performance of bit complement and bit reversal traffic using random task placement is practically indistinguishable from the same performance obtained on the (4,8,3) and (4,7,4) under-provisioned versions of the network. As for the actual value of this maximal performance, we experimentally measure it to be only roughly 50% of the corresponding balanced uniform traffic performance for the same scale, with little variation according to the pre-randomization structure of the permutation traffic.

Random task placement is thus unable to completely balance network load. The next question to answer is whether the remaining gap can be covered by route randomization. Unlike task placement changes, resorting to indirect routing has the added disadvantage of significantly increasing path lengths (as much as doubling them in some cases) and of requiring additional switch resources (in the form of extra virtual channels for deadlock avoidance). Nonetheless, it also offers a significant load balancing potential that is orthogonal to that achieve by altering the task placement. In the remainder of this section we will analyze the impact on the performance on permutation traffic patterns of two indirect routing algorithms, *Valiant [Kim:2008]* and *Valiant Any*, both on their own and coupled to randomized task placement.

Once more, our goal is to approach uniform traffic like load. As indirect routing has the drawbacks we just enumerated, even uniform traffic will suffer performance degradation under this routing approach. To establish the baseline impact of these drawbacks, we first analyze uniform traffic performance under indirect routing. Figure 6.5 presents the performance obtained for uniform random traffic under *Valiant [Kim:2008]* and *Valiant Any* under contiguous task placement. We included the shortest path routing results and the theoretical bound for reference. From literature [106], Valiant indirect routing, due to using twice a remote link in every path, instead of only once as is the case for minimal routing, is expected to achieve for uniform random traffic only half of the minimal routing throughput. As Figure 6.5 shows, this is indeed the case also in our experimental results, with a slight but consistent advantage for the *Valiant [Kim:2008]* approach versus the *Valiant Any* one. This is due to the fact that, although both use a remote link twice, it is only in the case of *Valiant Any* that the indirect

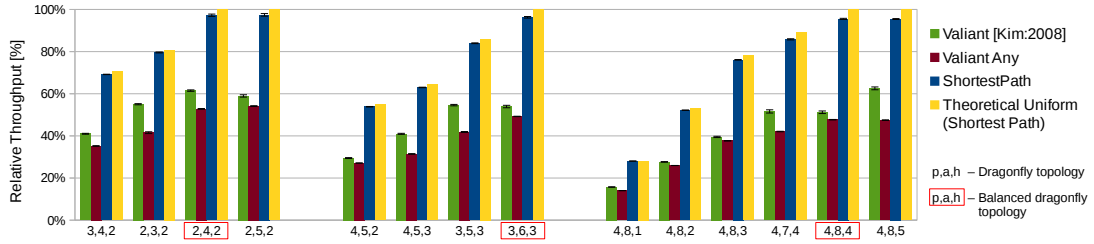


Figure 6.5 – Measured throughput for different dragonfly topologies under *Valiant [Kim:2008]* and *Valiant Any* for the uniform traffic pattern. Shortest path routing and the theoretical maximum included for reference.
Figure included from [84] ©2014 Springer

path is effectively a series of two complete direct paths. *Valiant [Kim:2008]* on the other hand uses an incomplete direct path as the first part of the indirect path, effectively avoiding an unnecessary increase of the load in the local links of the intermediate group (*Valiant Any* can and typically will use a sequence of two local hops in the intermediate group whereas *Valiant [Kim:2008]* will always use at most one local link in the same intermediate group). Thus, unlike the task placement approach, where the extra randomization had no effect on traffic that was already uniform (Figure 6.2), in the case of routing induced uniformization, the pre-existing or potentially resulting uniform traffic will be penalized by 40 – 50%.

Figure 6.6 shows the throughput achieved by the bit complement traffic under Valiant routings assuming a *contiguous* placement (including the previously discussed results with shortest path, for comparison). The first thing the results show is that both indirect routing approaches increase performance significantly compared to the minimal routing case. However, among the two, only *Valiant Any* is able to increase performance up to the level of uniform traffic. *Valiant [Kim:2008]* on the other hand hits a performance limit that is increasingly lower as the scale of the network grows (as low as 30% of uniform traffic performance for the larger scales benchmarked). This difference in performance between *Valiant [Kim:2008]* and *Valiant Any* stems from local bottlenecks that appear in the intermediate groups and that are inherent to the bit complement traffic pattern. Unlike *Valiant [Kim:2008]*, *Valiant Any* has the ability, due to the extra local hop it takes in the intermediate groups, to better distribute the load across local links in the intermediate groups, and thus to route around local potential hot-spots.

Next, we benchmark how the two randomization techniques behave when used in conjunction. Figure 6.7 shows a comparison for the bit complement traffic between the throughput obtained when using the Valiant variants both for contiguous and for random placements against the throughput obtained for shortest path routing with randomized placement. Several conclusions can be drawn from this chart. First, as *Valiant Any* was already achieving uniform-like performance before task placement randomization, the latter has little impact on performance, and typically a negative one. *Valiant [Kim:2008]* on the other hand benefits significantly from placement randomization, as the latter eliminates the local bottlenecks mentioned previously, and due to the shorter paths it uses, manages to achieve better perfor-

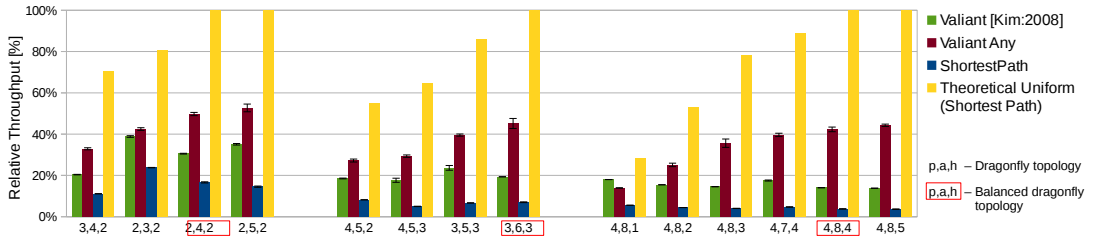


Figure 6.6 – Measured throughput for different dragonfly topologies under *Valiant [Kim:2008]* and *Valiant Any* for the bit complement traffic pattern. Shortest path routing and the theoretical maximum included for reference.
Figure included from [84] ©2014 Springer

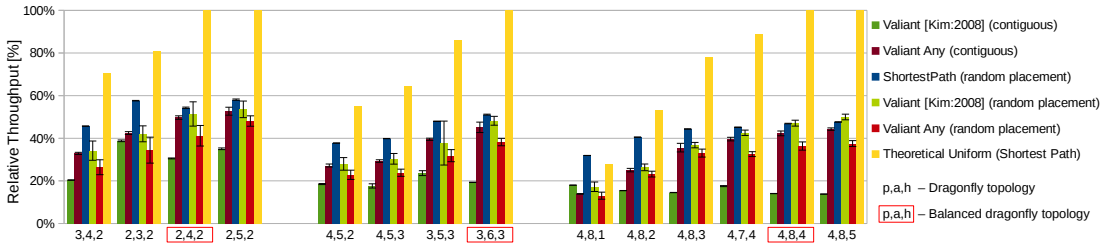


Figure 6.7 – Comparison of the measured throughput for different dragonfly topologies under *Valiant [Kim:2008]* and *Valiant Any* for both contiguous and random placement for the bit complement traffic. Theoretical maximum under uniform traffic added for reference.
Figure included from [84] ©2014 Springer

mance than *Valiant Any*. Both are outperformed however by shortest path routing with the same randomized task placement. We therefore arrive, for the bit complement traffic, at the same conclusion that was obtained for the multi-point stencil communication pattern [16], namely that the best performance is achieved by just randomizing the task placement and avoiding indirect routing altogether.

Finally, Figure 6.8 shows the same set of results for the second permutation workload, the bit reversal traffic pattern. As before, valiant routing is surpassed in performance by shortest path routing, regardless of the task placement strategy chosen. Furthermore, the two uniformization techniques, either alone or together, are generally not able to induce a better performance than the performance obtained initially for the contiguous shortest path routing. As such, we can conclude that, in general, these techniques seem to be able to normalize workload performance to approximately half the performance of uniform traffic. That is, workloads that exhibit a performance inferior to this level will benefit while workloads that originally surpass this level will likely not benefit from it and possibly be negatively impacted by it, especially in the case of the indirect routing uniformization approach.

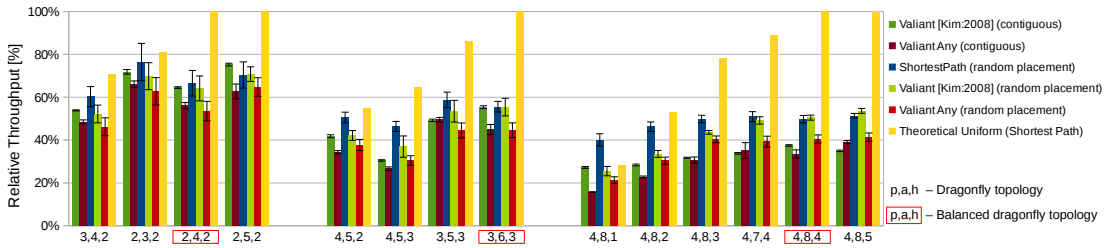


Figure 6.8 – Comparison of the measured throughput for different dragonfly topologies under *Valiant [Kim:2008]* and *Valiant Any* for both contiguous and random placement against shortest path routing with random placement for the bit reversal traffic pattern. Theoretical maximum under uniform traffic load added for reference.
Figure included from [84] ©2014 Springer

6.5 Summary

In this chapter, we derived a theoretical model of the throughput that can be obtained in a generic dragonfly network under a uniform traffic pattern. We then used this theoretical characterization to derive the topological conditions that ensure optimal performance at minimal cost under this traffic pattern. By conducting benchmarks on a wide array of dragonfly configurations, we were able to show that indeed the performance of the network under uniform traffic is correctly estimated by the model.

Furthermore, we evaluated whether techniques of network load uniformization (based on randomizing the placement of tasks on the compute nodes and/or randomizing path choice under indirect routing) are effective when applied to non-uniform, permutation workloads. By conducting once more an extensive series of benchmarks, we were able to show that, depending on the degree of non-uniformity of the workload, such techniques (task placement randomization in particular) could indeed be helpful as they achieve a normalization of performance across non-uniformity degrees. Specifically, the performance of highly inefficient permutation patterns is improved significantly, while the performance of efficient permutation patterns is left almost unchanged, resulting in roughly the same level of performance in both cases. However, we also showed that this resulting level of performance is still significantly lower (roughly 50%) than performance under perfectly uniform traffic and thus that there is still an important opportunity for optimization.

7 Cost-Effective Diameter-Two Topologies: Analysis and Evaluation

HPC network topology design is currently shifting from high-performance, higher-cost Fat-Trees to more cost-effective architectures. Three diameter-two designs, the Slim Fly, Multi-Layer Full-Mesh, and Two-Level Orthogonal Fat-Tree excel in this, exhibiting a cost per endpoint of only 2 links and 3 router ports with lower end-to-end latency and higher scalability than traditional networks of the same total cost. However, other than for the Slim Fly, there is currently no clear understanding of the performance and routing of these emerging topologies. For each network, we discuss minimal, indirect random, and adaptive routing algorithms along with deadlock-avoidance mechanisms. Using these, we evaluate the performance of a series of representative workloads, from global uniform and worst-case traffic to the all-to-all and near-neighbor exchange patterns prevalent in HPC applications. We show that while all three topologies have similar performance, OFTs scale to twice as many endpoints at the same cost as the others.

7.1 Related Work

One of the most popular interconnection network designs, used in both the High Performance Computing and datacenter space, is the Fat-Tree [60, 76, 80]. Full bisection Fat-Trees ensure that any permutation traffic can traverse the network at maximum bandwidth and can attain close to ideal behavior for any communication pattern (with a properly chosen routing strategy) in practice. At small scale, Fat-Trees require only two levels of switches and as such are very cost effective. When moving to larger scales, they require increasingly more resources as the number of levels increases, both in terms of routers and network cables. Typical cost reduction measures (such as slimming for example) generally have the drawback of impacting performance across many traffic patterns, particularly the more global ones. In

This chapter is based on the article [52] KATHAREIOS, G., MINKENBERG, C., PRISACARI, B., RODRÍGUEZ, G., AND HOEFLER, T. Cost-effective diameter-two topologies: analysis and evaluation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015* (2015), ACM, pp. 36:1–36:11. https://doi.org/10.1145/2807591.2807652

practice, the network does not need to be optimal for *every* pattern, but global communication (such as uniform random global traffic, or all-to-all exchanges) is central to many applications. Therefore, several alternatives have been proposed that are more cost effective than a three level Fat-Tree while maintaining close to ideal performance for a subset of communication patterns, particularly global uniform traffic. The most widely deployed of these has been the Dragonfly topology [54], employed for example in the IBM PERCS systems [10] and in Cray Cascade systems [27] (e.g., Piz Daint and Shaheen II [2]).

However, most of these alternative topologies are still not a match for the two level Fat-Tree, neither in terms of cost nor in terms of latency/diameter. Two classes of topologies, one direct and one indirect, are exceptions to this rule, sharing the same cost and diameter metrics with the two-level Fat-Tree but having better, approaching maximal, scalability, making them ideal candidates for current HPC and datacenter interconnects. The direct topology is the recently proposed Slim Fly (SF) [15]. The indirect topology class is one that we introduce in this chapter and that we call Stacked Single-Path Trees (SSPT). The Multi-Layer Full-Mesh (MLFM) [31] as well as the two-level Orthogonal Fat-Trees (OFT) [104, 105] are examples of members of this class.

While the individual topologies have already been described in the literature, the SSPT class has not. Also, other than for the Slim Fly there have been no concrete proposals as to how to perform routing and deadlock avoidance, nor have there been comparative performance studies between the three options. In this chapter:

- We introduce and analyze the SSPT topology class.
- We compare the diameter-two topologies in terms of cost, scalability, and bandwidth limitations.
- We propose load oblivious and adaptive deadlock-free routing strategies for the MLFM and OFT topologies and discuss these in comparison with respective strategies for the SF.
- We evaluate the performance of the three topologies under the proposed routing and deadlock avoidance strategies through simulations, for several representative communication patterns: on the one hand global uniform and adversarial worst-case synthetic traffic and on the other hand two of the most representative traffic patterns in the HPC space: all-to-all and nearest-neighbor exchanges.

7.2 Diameter-Two Topologies

In this section we present the description of the topologies and perform an analysis of their main characteristics.

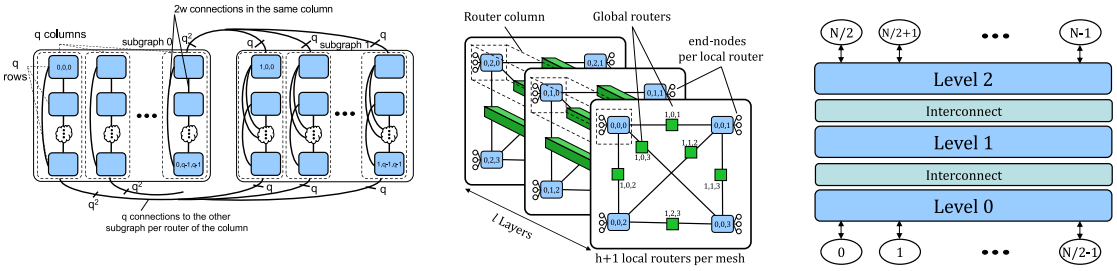
		Symbol	Explanation
General		N	Number of end-nodes
		R	Number of routers
		r	Router radix
		p	Number of end-nodes attached to routers (Applies only to routers with end-nodes for the OFT and MLFM)
		N_p	Number of total router ports in the topology
		N_l	Number of total links in the topology
Direct	SF	q	Prime power that defines N, R, and r
		r'	Network radix
		(i,j,k)	Router in the j-th row and k-th column of the i-th subgraph
Indirect	MLFM	l	Number of layers
		h	Network radix of local routers
		R_g, R_l	Number of global, local routers
	OFT	k	Network radix of routers with end-nodes
		(i,j)	j-th router of the i-th level
		R_L	Number of routers per level

Table 7.1 – Symbols used in this chapter.
Table included from [52]

7.2.1 Direct Topologies

7.2.1.1 Two-Dimensional HyperX

The n-dimensional HyperX (also known as Generalized Hypercube) [3,17] is the direct topology resulting from the Cartesian product of n fully-connected graphs. The longest path between a pair of routers in such a topology has length n, corresponding to one movement in each of the n dimensions. The two-dimensional HyperX is thus a direct diameter-two topology. It is defined by three main parameters: the sizes of the two fully-connected graphs in the Cartesian product and the number p of end-nodes connected to each router. Typical configurations use the same size for the two fully-connected graphs while p is chosen such that the network is balanced, that is, it is able to sustain full injection bandwidth for uniform traffic. Given a router with radix r, this translates into an equal number of ports $r/3$ being used to i) connect to routers in the same fully-connected graph in the first dimension; ii) connect to routers in the same fully-connected graph in the second dimension; and iii) connect to end-nodes. The number of routers in each fully-connected graph is thus $\frac{r}{3}+1$ leading to a total number of routers of $R=(\frac{r}{3}+1)^2$ and a total number of end-nodes of $N=pR=\frac{r}{3}\cdot(\frac{r}{3}+1)^2$. The network has a per end-point cost of 3 router ports and 2 links.

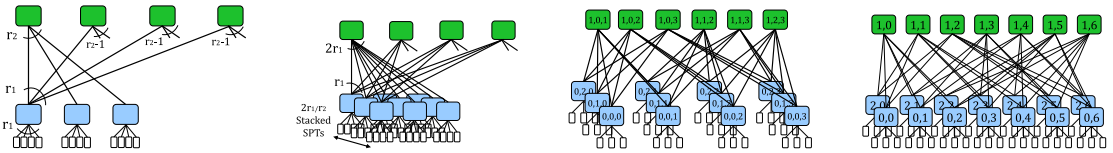


(a) Diameter-Two Slim Fly (End-nodes are omitted for clarity).

(b) 3,3,3 MLFM.

(c) Two-level OFT.

Figure 7.1 – System view of diameter-two cost-effective topologies.
Figure included from [52]



(a) SPT.

(b) SSPT.

(c) 3,3,3-MLFM.

(d) 3-OFT.

Figure 7.2 – Tree view of diameter-two cost-effective topologies.
Figure included from [52]

7.2.1.2 Diameter-Two Slim Fly

The SF [15] is a direct topology created by arranging the routers in a McKay, Miller, and Širán (MMS) graph [66]. MMS graphs approximate the Moore Bound [68], the maximum number of nodes a graph can have for a given node degree and diameter. This effectively means that the SF is among the largest direct diameter 2 networks possible, reaching approximately 88% of the Moore Bound [15].

The creation of the SF topology begins with the selection of a prime power q of the form $q = 4w + \delta$, $w \in \mathbb{N}$, $\delta \in \{-1,0,1\}$. From that, we calculate ξ, a primitive element of the Galois Field F_q and create two *generator sets* X and X', as follows (all calculations are performed over the field):

- $X = \{1, \xi^2, \cdots, \xi^{q-3}\}$, $X' = \{\xi, \xi^3, \cdots, \xi^{q-2}\}$, if $\delta = 1$
- $X = \{1, \xi^2, \xi^4, \cdots, \xi^{2w-2}, \xi^{2w-1}, \xi^{2w+1}, \cdots, \xi^{4w-3}\}$,
  $X' = \{\xi, \xi^3, \xi^5, \cdots, \xi^{2w-1}, \xi^{2w}, \cdots, \xi^{4w-4}, \xi^{4w-2}\}$,
  if $\delta = -1$
- $X = \{1, \xi^2, \cdots, \xi^{q-2}\}$, $X' = \{\xi, \xi^3, \cdots, \xi^{q-1}\}$, if $\delta = 0$

The topology consists of $R = 2q^2$ routers, arranged in two subgraphs, each with q^2 routers in q rows and columns (Fig. 7.1a). Each router has two kinds of network connections: $2w$

connections to routers of the same column in its subgraph, and q connections directed to the other subgraph, one in each of its columns, leading to a network radix $r' = \frac{3q-\delta}{2}$. Specifically:

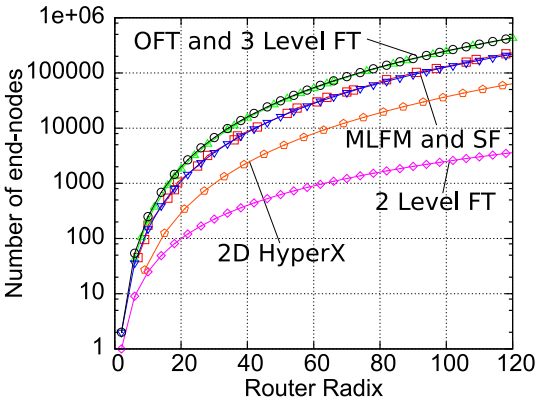
$(0,x,y)$ connects to $(0,x,y')$ if $y - y' \in X$

$(1,m,c)$ connects to $(1,c,c')$ if $c - c' \in X'$

$(0,x,y)$ connects to $(1,m,c)$ if $y = mx + c$

According to Besta and Hoefler [15], the number p of end-nodes connected to each router is selected to be equal to half the network radix: $p = \lceil \frac{r'}{2} \rceil$. This value ensures full global bandwidth for the topology (i.e., given a uniform communication pattern, end-nodes are theoretically able to sustain full bandwidth injection). Here, we argue that using the ceiling function, while enabling higher scalability and better cost per endpoint, slightly overestimates the number of nodes that can be attached to a router, leading to lower performance. As such, we also consider configurations with $p = \lfloor \frac{r'}{2} \rfloor$ and show in Section 7.4 the performance implications of this choice.

The Slim Fly comprises $R = 2q^2$ routers. These routers have a network radix of $r' = q + 2w = \frac{3q-\delta}{2}$ and $p = \frac{r'}{2} \approx \frac{3q-\delta}{4}$ attached end-nodes, amounting to a router radix of $r = \frac{3}{4}(3q - \delta)$. The number of end-nodes is $N = pR \approx \frac{q^2(3q-\delta)}{2}$. The total number of router ports is $N_p = rR \approx \frac{3}{2}q^2(3q - \delta)$, and the total number of links is $N_l = N + \frac{r'R}{2} \approx q^2(3q - \delta)$, resulting in a cost of approximately 3 ports and 2 links per end-node. As practical values of q are relatively small, the choice of rounding up or down when determining p has a non-negligible impact on the cost metrics. As an example, for $q = 13$, selecting $p = 10$ results in a cost of 2.9 ports and 1.95 links per endpoint, while selecting $p = 9$ results in 3.11 ports and 2.05 links per endpoint.



	Topology	Diam.	Scale	$\frac{N_l}{N}$	$\frac{N_p}{N}$
Direct	2D HyperX	2	$\approx r^3/27$	2	3
	Slim Fly (SF)	2	$\approx r^3/8$	2*	3*
Indirect	2-lvl Fat-Tree	2	$r^2/2$	2	3
	3-lvl Fat-Tree	4	$r^3/4$	3	5
	MLFM	2	$\approx r^3/8$	2	3
	OFT	2	$\approx r^3/4$	2	3

* See Section 7.2.1.2

Figure 7.3 – Comparison of the scale and cost (links and ports per end-node) of various low diameter topologies.
Figure included from [52]

7.2.2 Indirect Topologies

7.2.2.1 Two Level Fat-Trees

A two level Fat-Tree is an indirect topology built from two levels of routers. The endpoints are connected to the routers on the first level while the second level is used to provide an increase in bandwidth/path diversity. A full bisection two level Fat-Tree must additionally obey the constraint that every level-one router have as many links connecting it to level-two routers as endpoints. This effectively translates into the network theoretically providing sufficient bandwidth to accommodate any permutation traffic pattern at full injection. Given a configuration where every router has a fixed even router radix r, the number of nodes connected to every level-one router is then $p = r/2$, the total number of routers is $3r/2$ and the total number of endpoints is $r^2/2$. The cost of the network is 3 router ports and 2 links per endpoint.

7.2.2.2 Stacked Single-Path Trees

While the full bisection two-level Fat-Tree simultaneously offers a low cost, a low diameter, and high throughput for any permutation pattern (due to high path diversity), its use in practice is limited by its low scalability under a fixed router radix. In this section we will show that we can sacrifice one of the defining characteristics of this design, the high path diversity, and still preserve the other two advantages, while also providing high throughput for random uniform traffic. To define the structure of *Single-Path Trees* (SPT) (Fig. 7.2a), we start as in the Fat-Tree case with two levels of routers, where nodes will be connected only to routers in the first layer and every router-to-router link has an endpoint in each of the levels. Unlike the Fat-Tree though, we propose to interconnect the two layers in such a way that i) exactly a single minimal path exists between any pair of level-one routers, and ii) a minimal number of level-two routers are used. Given a router-to-router radix of r_1 for the first level routers and

i	j, s.t. (1,j) and (0,i) are connected			
0	9	10	11	12
1	9	0	1	2
2	9	3	4	5
3	9	6	7	8
4	10	0	3	6
5	10	1	4	7
6	10	2	5	8
7	11	0	4	8
8	11	1	5	6
9	11	2	3	7
10	12	0	5	7
11	12	1	3	8
12	12	2	4	6

Table 7.2 – 4-ML3B.
Table included from [52]

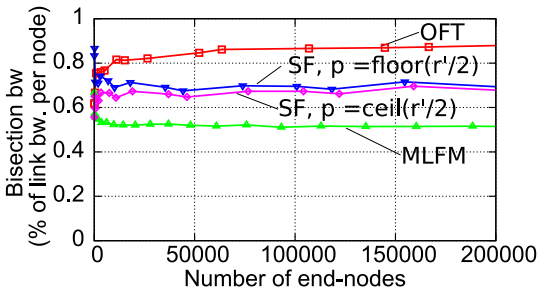


Figure 7.4 – Approximating bisection bandwidth.
Figure included from [52]

of r_2 for the second level routers, such a design scales to $R_1 = 1 + r_1 \cdot (r_2 - 1)$ first level routers (requiring $R_2 = R_1 \cdot r_1 / r_2$ second level routers). To achieve maximum performance for random uniform traffic, the number of nodes connected to each level-one router is limited to $p = r_1$. The scalability of an SPT is thus given by $N = r_1^2 \cdot (r_2 - 1) + r_1$. In terms of cost, an SPT then requires R_1 routers with $2r_1$ ports each and R_2 routers with r_2 ports each, for a total of $3R_1 r_1$ ports, or 3 ports per end-point. Similarly, the number of links per endpoint is equal to 2.

Determining level-one to level-two interconnection patterns with the constraints characteristic of SPTs is not straightforward and as such we might be limited in practice to certain combinations of (r_1, r_2) values. In particular, precise procedures are known to build such interconnection patterns for the $r_2 = r_1$ case when $r_1 - 1$ is prime as well as for any value of r_1 when $r_2 = 2$. However, when building interconnection networks in practice, it is often desirable that individual routers be identical, i.e., have the same radix. An interconnection pattern might not be readily available for the $(r_1, r_2 = 2r_1)$ case corresponding to this goal for an SPT. One might however be available for the (r_1, r_2) case, r_2 being some divisor of $2r_1$. In that case, one way of achieving the goal of building the network from individual routers is a procedure that we will call stacking SPTs and leading to *Stacked Single-Path Tree* (SSPT) topologies (Fig. 7.2b). The procedure consists of logically instantiating $2r_1 / r_2$ identical $SPT(r_1, r_2)$ topologies and then "merging" each $2r_1 / r_2$-tuple of corresponding radix-r_2 level-two routers together to form a single $2r_1$ radix physical router. For every intra-SPT pair of endpoints, the SPT properties are trivially preserved. For every inter-SPT pair of endpoints, the diameter property is preserved (a shortest path always consists of an upward SPT traversal in the source SPT and a downward SPT traversal in the destination SPT), while the single path property is preserved for all pairs excepting those containing corresponding nodes in two different SPTs. Indeed, for the latter, the path diversity is equal to r_1. The scale of the resulting network is: $N = \left(r_1^2(r_2 - 1) + r_1\right) \cdot \frac{2r_1}{r_2} = \frac{r^3}{4}\left(\frac{r_2 - 1}{r_2}\right) + \frac{r^2}{2r_2}$, where $r = 2r_1$ is the router radix of the resulting topology. As each SPT that we stack has a network cost of 3 ports and 2 links per end-node, the SSPT has the same cost per end-node.

In the following, we will present two particular instances of SSPTs, the Multi-Layer Full-Mesh,

obtained for $r_2 = 2$ and the two level Orthogonal Fat-Tree, obtained for $r_2 = r_1$ when $r_1 - 1$ is prime.

7.2.2.3 Multi-Layer Full-Mesh

The *MLFM* [31], as the name suggests, is initially conceived as stacked layers of full-mesh networks. The stacking procedure is performed by considering each layer as a normal full-mesh, and replacing the direct link between any pair of local routers (LRs) with two links to one global router (GR) (Fig. 7.1b). By connecting the respective pairs of all layers to the same GRs, each LR can reach any other through its GR neighbors.

The LRs of each full mesh are the routers of the lower level of the SPT, with end-nodes attached to each of them. The GRs, with $r_2 = 2$ connections on each SPT, are the routers of the upper level of the SPT, and stacking them leads to the creation of the SSPT (Fig. 7.2c).

In more detail, the *(h,l,p)-MLFM* contains l layers with $h+1$ LRs each, and p endpoints attached to each LR. The number of GRs used to perform the stacking is $R_g = \frac{h(h+1)}{2}$, and their radix is $r_g = 2l$. The radix of LRs is $r_l = h + p$. Following the description of SSPTs that can be realized with a single fixed router radix, we will only consider the case where $h = l = p$, where all routers of the MLFM have the same radix $r = 2h$ and refer from now on to the *h-MLFM*.

The h-MLFM contains h layers of $h+1$ LRs each, along with $\frac{h(h+1)}{2}$ GRs, amounting to a router count of $R = \frac{3}{2}h(h+1)$. Each LR is connected to h end-nodes, and thus the whole topology contains $N = h^3 + h^2$ end-nodes. Being an SSPT, the MLFM has a cost of 3 router ports and 2 links per end-node.

7.2.2.4 Two-Level Orthogonal Fat-Trees

Building SSPTs with $r_1 = r_2 = k$ requires stacking only 2 SPTs, leading to the *Two-Level k-OFT* [104, 105], a three-level indirect network (Fig. 7.1c). The lower layers of each of the two SPTs (which we will call $L0$ and $L2$) will consist of $R_0 = R_2 = 1 + k(k-1)$ routers each. The common upper layer of the two SPTs (which we will call $L1$) will also have the same number of switches $R_1 = R_0 = R_2 = R_L$ (due to $r_1 = r_2$ within the individual SPTs).

The interconnection pattern of the SPT that creates the k-OFT is called the *Maximal Leaves Basic Building Block of degree k* (k-ML3B) of the topology [105]. The tabular representation of the k-ML3B is a $R_L \times k$ table, whose i-th row contains all values j for which $(0,i)$ connects to $(1,j)$. An algorithm to create this representation of the k-ML3B has been described for the case where k equals a prime plus one [104]. This algorithm involves filling the table using a set of slightly modified Mutually Orthogonal Latin Squares (MOLS) [23]:

1. Fill the first row with the numbers in the range $[R_L - k, R_L - 1]$ in that order.

2. Fill the remaining empty cells of the first column with $k-1$ instances of R_L-k, $k-1$ instances of R_L-k+1, ..., $k-1$ instances of R_L-1.

3. At this point, a $k(k-1)\times(k-1)$ area of the tabular representation is unfilled. This subset is divided in k squares, each of size $(k-1)\times(k-1)$. The first is filled with the numbers 0 to $(k-1)^2-1$, ordered from left to right and from top to bottom. The second is filled with the transpose of the first. Each remaining square is filled with each of the $k-2$ MOLS of size $(k-1)\times(k-1)$ with elements in the range $[0,k-2]$. Finally, the i-th column of each of these $k-2$ squares is increased by $(i-1)\cdot(k-1), \forall i \in [1,k-1]$.

Table 7.2 shows the result of the above algorithm for the tabular representation of the 4-ML3B.

Each router of the $L0$ and $L2$ levels connects to $p=k$ end-nodes (as explained in Section 7.2.2.2). This results in a topology with $N=2kR_L=2k^3-2k^2+2k$ end-nodes, built with $2k$-radix routers. The number of routers needed is $R=3R_L=3k^2-3k+3$. As all SSPTs, the OFT also has a cost of 3 router ports and 2 links per end-node.

7.2.3 Analysis

7.2.3.1 Scalability

Fig. 7.3 shows the scalability of the considered diameter-two topologies, as well as that of the three-level Fat-Tree as a reference for comparison. The table in the same figure contrasts the scale with the cost of each of these topologies. Asymptotically, all topologies scale to a number of nodes that is proportional to the cube of the router radix with the exception of the two-level Fat-Tree which scales proportionally to only the square of the router radix. However, the exact end-node count is significantly different from topology to topology: i) among the direct topologies, a SF will be able to accommodate $\approx 27/8$ more end-nodes than a 2D HyperX build with the same-sized router; ii) among the indirect topologies, both OFT and MLFM offer significantly higher scalability than the same diameter FT, with the OFT scaling to twice as large a network than the MLFM (thus achieving a number of end-points similar to the much more costly 3-level FT).

As an example, using a radix-64 router design, the OFT can support approximately 63.5K nodes, while the MLFM and SF support around 36K and 33.7K, respectively. Thus, they are both good candidates for the interconnection network of the largest of today's datacenters or that of exascale HPC systems (e.g. a system comprising 40TFlop nodes – i.e., CORAL nodes [74], 2017 time frame – would require an interconnect that scales to 25,000 nodes to reach peak exaflop performance).

Comparing between the best direct topology option (SF) and the best indirect topology option (OFT) is more problematic, as the choice of one over the other is highly dependent on the technology envisioned for the design. Particularly, direct topologies typically benefit from having the router integrated close to the compute chips, whereas in the case of indirect

topologies routers are discrete. Thus, the choice between the two topologies is subject to a cost-scalability tradeoff.

7.2.3.2 Upper bound for bisection bandwidth

Due to the irregular nature of the OFT and the Slim Fly, an analytical calculation of their bisection bandwidth is not easy. However, we can approximate it for the topologies under discussion using a graph partitioning tool [51]. The approximate results (Fig. 7.4) suggest that the OFT benefits from the higher bisection bandwidth among the three topologies, offering $\approx 0.89b$ per end-node ($\approx 0.81b$ for small scale networks), b being the link bandwidth. Conversely, the approximate bisection bandwidth for the SF is $\approx 0.71b$ per node when $p = \lfloor \frac{r'}{2} \rfloor$ and $\approx 0.67b$ per node when $p = \lceil \frac{r'}{2} \rceil$. Finally, the MLFM seems to have the lowest bisection bandwidth of the three, being limited to $\approx 0.5b$.

7.2.3.3 Diversity of shortest paths

Compared to the two-level Fat-Tree, the considered diameter-two topologies (both direct and indirect) trade diversity of shortest paths for higher scalability. Nonetheless, in all of them there exist (source,destination) router pairs between which more than one shortest path exists. These pairs can potentially be leveraged by optimized routing and/or mapping policies to improve the network's performance under adversarial traffic patterns.

For the SF there is no path diversity between routers that are directly connected. However, some pairs of non-directly connected routers share more than one neighbor and thus there exists some path diversity between them. Such pairs are scarce, thus system-wide path diversity is relatively low. For instance, for $q = 23$, the average number of minimal paths between pairs of non-directly connected routers is approximately 1.1, with the maximum path diversity being 8.

The MLFM also exhibits path diversity that is irregularly distributed across the network. A pair of LRs that belong to the same router column, that is, having the same relative index in their respective layer (Fig. 7.1b), have h minimal routes between them. Any other pair of routers however have strictly one minimal path between them.

The OFT (and SPTs/SSPTs in general) are designed to provide high scalability through the reduction of diversity of minimal paths between routers connected to end-points. In general, only one minimal route exists between any pair of such routers, with the only exception of pairs of counterpart routers in different stacked layers. Due to the symmetry, routers $(0,i)$ and $(2,i)$ connect to the same $L1$ routers and as result there are k minimal paths between them.

7.3 Routing

We discuss routing of packets from a source node directly connected to router R_s, to a destination node directly connected to a different router R_d.

7.3.1 Oblivious Minimal Routing

The SF is the only one of the three topologies without constant length minimal paths. Being a direct topology, R_s and R_d can be either directly connected, in which case the path has a length of 1, or connected through a common neighbor, in which case there exists a 2-hop minimal path.

Minimal routing for the MLFM consists exclusively of 2-hop paths. The local source router R_s sends the routed packet to a global router, to be forwarded to R_d. If the communicating router pair belongs to the same column (Fig. 7.1b), any output port of the source leads to a global router that is connected to the destination. However, if this is not the case, there is again only one global router that is a common neighbor of both R_s and R_d.

Similarly, in minimal routing for the OFT, a packet from R_s traverses an $L1$ router to reach R_d, on a strictly 2 hop path. In the case where R_s and R_d are a symmetric pair, any $L1$ router connected to R_s can be used as the intermediate hop, since the same $L1$ routers are connected to R_d. In the opposite case, only one $L1$ router connects the source-destination pair and therefore the intermediate hop is their single common neighbor.

7.3.2 Oblivious Indirect Random Routing

Valiant's algorithm [106] can be used to load balance adversarial traffic patterns where minimal routing underperforms. A packet from R_s is first minimally routed to a uniformly randomly selected intermediate router R_i (other than the source and destination routers) and from there minimally routed to its final destination R_d. In the case of SF, any router in the topology is eligible to become an intermediate router. This means that indirect routes have a length of 2, 3 or 4 hops.

In the case of the OFT and the MLFM however, if any router in the topology is eligible to become R_i, indirect paths would have lengths of 2, 4 or 6 hops. However, this is not desirable, since short paths will result in inadequate load balancing and long paths will result in higher latency. Hence, for these two topologies, the intermediate destination is chosen among the routers that are directly connected to end-nodes ($L0$ or $L2$ routers for the OFT, local routers for the MLFM), restricting the indirect path length to 4 hops.

7.3.3 Adaptive Routing

For adaptive routing on the topologies under discussion, we explore variants of the Universal Globally-Adaptive Load-balanced (UGAL) algorithm [98], which has already been successfully used in various other topologies [15, 54]. The UGAL algorithm selects between minimal and indirect random routing on a per-packet basis, based on the channel load at the moment of the packet's injection, as conceived by the network's buffers' occupancy level. The global variant of the algorithm requires knowledge of the buffers' state for the whole topology at the point of injection, which is hard to implement in practice. Here, we only consider the local variant of UGAL, in which each router has access exclusively to information about the state of its own buffers.

In general, the generic UGAL algorithm works as follows: When a packet is injected to the network a number of possible paths is selected and each one of them is assigned a cost. The minimal path is assigned a cost C_M equal to the occupancy of the first output port of the path : $C_M = q_M$[1]. Additionally, n_I indirect routes are randomly selected, and a cost $C_I^j, j \in \{1..n_I\}$ is assigned to each one of them, calculated as follows:

$$C_I^j = c \cdot q_I^j$$

where c denotes the penalty of the selection of an indirect path over a minimal one, and q_I^j is the occupancy of the first output port of the particular path. Finally, the path with the minimum cost is selected for the routing of the packet.

The generic UGAL algorithm has the drawback that it allows packets to be routed indirectly even when the occupancy of the minimal path is low, because some indirect path starts with an empty or lower occupancy first buffer (when $q_I^j = 0$, the value of c doesn't matter). Because an indirect path is twice as long as a minimal one, we expect the packet to have an increased latency. Adding to this problem is the fact that there is a good chance that the occupancy of the first output buffer does not always accurately reflect the congestion on its links. Therefore, in our experiments, in addition to the generic UGAL, we use also a modified version of the algorithm, in which packets are routed minimally when $q_M < T$ and adaptively in the opposite case, with T being a threshold in the buffer occupancy, expressed as a percentage of the buffer size.

7.3.3.1 SF adaptive routing

For the SF we use the generic algorithm (SF-A), but we base the cost calculation to the one of the original UGAL algorithm [98], similarly to Besta and Hoefler [15]. In this calculation, the cost of an indirect path is analogous to the ratio of the length of the indirect path (L_I^j) to the

[1] In the rare cases where multiple minimal paths exist, we can either select one of them at random, or select the minimal path with the lowest cost.

length of the minimal path (L_M). Thus, we have:

$$c = \frac{L_I^j}{L_M} \cdot c_{SF}$$

where c_{SF} is a constant, selected to balance the ratio between minimal and indirect routes. The same calculation is also used for adaptive routing with a threshold (SF-ATh).

7.3.3.2 MLFM and OFT adaptive routing

For both MLFM and OFT we use the generic UGAL algorithm (MLFM-A and OFT-A, respectively) and the version of UGAL with a threshold (MLFM-ATh and OFT-ATh) with a constant value for c.

7.3.4 Deadlock Freedom and Avoidance

The deadlock avoidance scheme proposed by Besta and Hoefler [15] for the SF utilizes 2 VCs in the case of minimal routing, and 4 in the case of indirect routing to effectively avoid deadlocks without restricting turns.

The OFT and MLFM topologies require less VCs than that. Both are inherently deadlock-free when minimal routing is utilized. In both cases, all uni-directional links belong in one of two groups. In the case of the OFT these groups can be characterized as *towards* or *away* from an $L1$ router, and in the case of the MLFM, they can be characterized as *towards* or *away* from a global router. In both topologies, a minimal route comprises a *towards* link followed by an *away* link, and therefore, since an order can be imposed on the classes of links so that they are always allocated in ascending order, this kind of routing entails no risk of routing deadlock [22].

On the contrary, with indirect routing the risk is present, since the routes are now of the form: *towards*, *away*, *towards*, *away*, thus forming cycles on the channel dependency graph (CDG). These cycles can be easily avoided by using 2 Virtual Channels (VCs) at each port. The first VC is used when a packet is moving towards the intermediate destination (first *towards*, *away* pair) and the second VC is used when the packet is moving away from it (second pair). This VC allocation scheme results in two virtual networks, each of which has the same cycle-free CDG as minimal routing, thus avoiding any deadlock.

7.4 Experimental Results

In this section we present simulation-based performance results for the three topologies, using the routing and deadlock avoidance approaches we introduced.

7.4.1 Framework, parameters and metrics

The results presented in this section were obtained using a simulation framework [69] that is able to accurately model generic and custom networks at a flit level. The switch architecture chosen was that of a virtual-channel capable, input-output-buffered switch with 100 KB of buffer space per port per direction and a switch traversal latency of 100 ns. The links had a bandwidth of 100 Gbps and a latency of 50 ns. Credit based flow control was used and messages consisted of 256 byte packets.

We benchmarked several traffic patterns, both synthetic (global uniform traffic and adversarial permutation traffic) and representative of real-world applications (nearest neighbor and all-to-all communication patterns). For the synthetic traffic patterns, messages were generated continuously at link rate for the entire duration of the simulation, while for the realistic patterns, the total amount of data exchanged between any communicating pair was 512 KB for nearest neighbor and 7.5 KB (30 packets) for all-to-all. For each traffic pattern, the assignment of processes to nodes was performed contiguously with a single process per node.

For the synthetic traffic experiments, the system was simulated for 200 microseconds with a 20 microseconds warm-up. For the realistic communication patterns, the system was simulated for the entire duration of the exchange.

The topology configurations that were used are the following:

- SF with $q = 13$, $p = \lfloor \frac{r'}{2} \rfloor = 9$, $N = 3042$, $R = 338$, $r = 28$
- SF with $q = 13$, $p = \lceil \frac{r'}{2} \rceil = 10$, $N = 3380$, $R = 338$, $r = 29$
- MLFM with $h = 15$, $N = 3600$, $R = 360$, $r = 30$
- OFT with $k = 12$, $N = 3192$, $R = 399$, $r = 24$

These were selected to approximate the number of nodes in CORAL Summit [74], a 150 PetaFlops system to be deployed in the 2017 time frame in a collaboration between IBM, NVIDIA and Mellanox.

7.4.2 Worst-Case Traffic

What constitutes an adversarial or worst-case (WC) workload varies from topology to topology. We consider patterns that are not end-node limited, meaning that a node does not generate a higher load than its link to the network can accommodate, nor does it receive more. In other words, patterns for which the bottleneck is in the network itself, not in the interface from the nodes to the network.

The WC traffic pattern under minimal routing in SF is encountered when all routers communicate in pairs with a distance of 2, and pairs of the routes of this communication partially

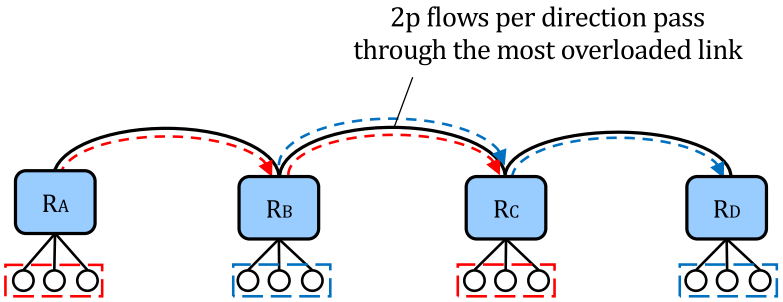


Figure 7.5 – Worst-Case traffic for the SF is encountered when all routers of the topology communicate in pairs of distance 2, with overlapping routes.
Figure included from [52]

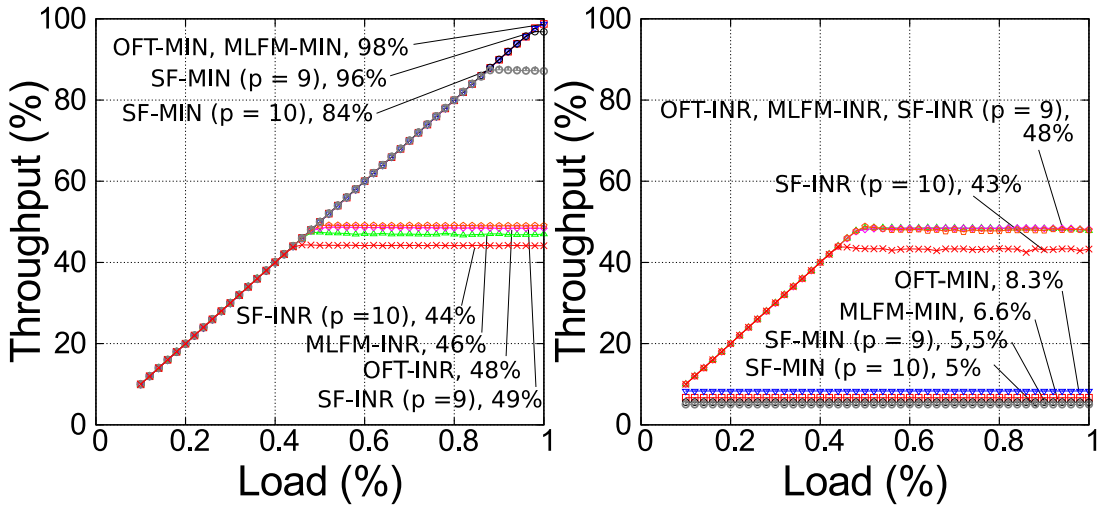
overlap. Fig. 7.5 shows an example of such a pair of overlapping routes. All end-nodes of R_A send their traffic to the end-nodes of R_C and all nodes from R_B send to all nodes of R_D. Thus, the link that connects R_B and R_C becomes overloaded, with $2p$ flows passing per direction, resulting to $\frac{1}{2p}$ of the maximum throughput. Arranging all traffic in the network in such pairs for our experiments is easily achieved with a greedy assignment.

The WC traffic pattern under minimal routing for the MLFM occurs when end-nodes belonging to pairs of routers connected by a single minimal path communicate exclusively across that path. Specifically, this occurs when the end-nodes connected to a router R_s send all their traffic to the endpoints of a router R_d which does not belong to same column as R_s, thus overloading the single minimal path between R_a and R_d with h flows. A particular case of this pattern is the shift traffic pattern with a shift value of h, which we use in our experiments.

The WC traffic pattern for the OFT under minimal routing is very similar to the one for the MLFM. Once again, it occurs when all end-nodes of some $L0$ or $L2$ router R_s communicate exclusively with all end-nodes of some other router R_d of the same levels, which is not the symmetrical equivalent of R_s. In this case, each link of the common path used is oversubscribed with k flows, resulting in a throughput equal to only $\frac{1}{k}$ of the link bandwidth. Once more, we use in our experiments a particular case of this pattern that is the shift traffic pattern with an offset of k.

7.4.3 Synthetic traffic

Our synthetic traffic experiments were conducted under global uniform and worst-case adversarial permutation traffic (as discussed in Section 7.4.2), in order to explore the limits of each topology and each routing strategy.



(a) Uniform random traffic.

(b) Worst case traffic.

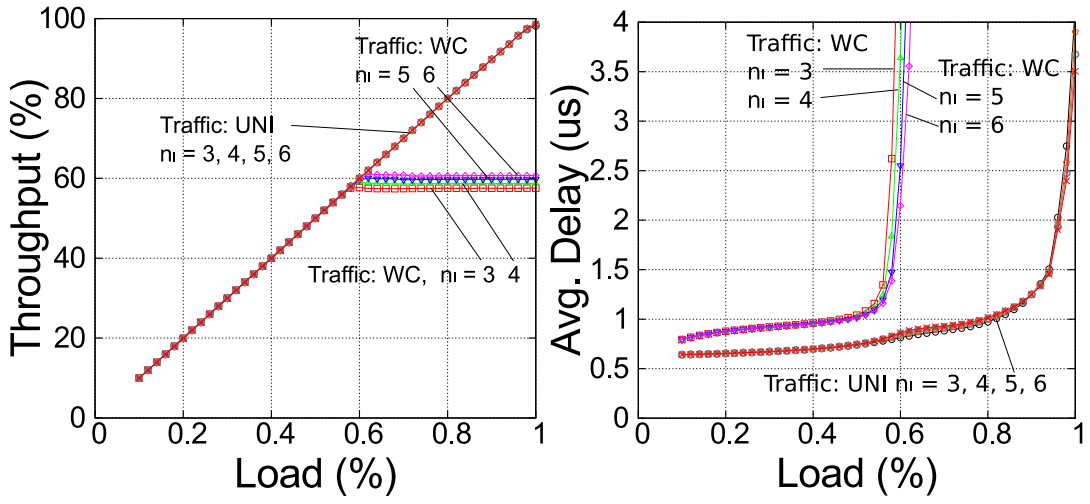
Figure 7.6 – Throughput, and throughput saturation points for oblivious (minimal, MIN and indirect random, INR) routing, under uniform random and worst-case traffic.
Figure included from [52]

7.4.3.1 Oblivious routing

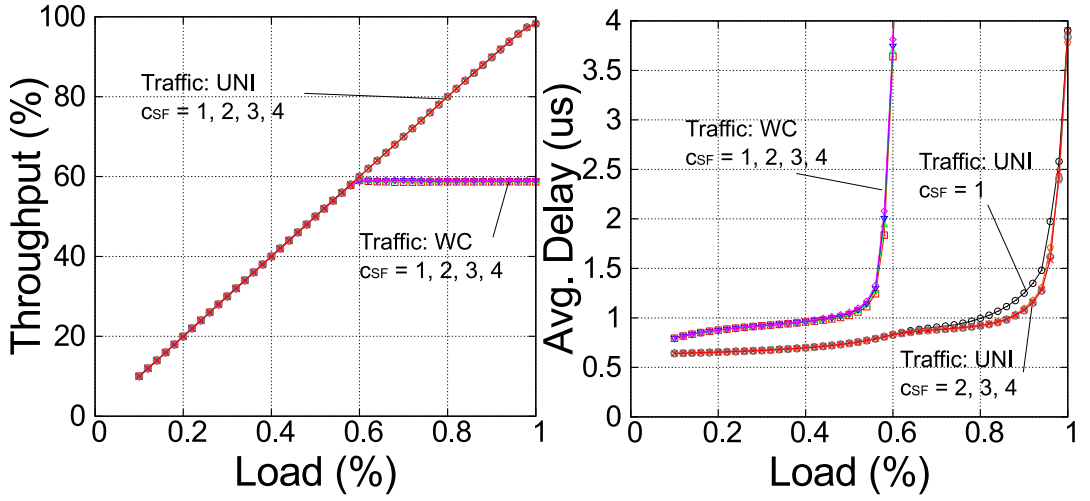
Fig. 7.6 shows the throughput achieved by the three topologies using oblivious routing, either minimal or indirect random. Under minimal routing, all three topologies are able to support almost full bandwidth in uniform traffic, up to approximately 96 to 98% of the total load. Between the two versions of SF, the one with higher p saturates faster, at approximately 87% of the injection rate (in accordance with the results presented by Besta and Hoefler [15]). Still, this routing strategy severely underperforms on WC traffic. As all three topologies have a low degree of minimal path diversity, they saturate at load levels as low as 5% (SF), 6.6% (MLFM), and 8.3% (OFT), equal to $\frac{1}{2p}$, $\frac{1}{h}$, and $\frac{1}{k}$ respectively, as calculated in Section 7.4.2. Indirect random routing helps alleviate the problem by load-balancing the network equally. However, because it effectively doubles the length of all routes, the throughput saturation point in both kinds of traffic becomes equal to half the saturation point in uniform traffic, and the average packet latency increases accordingly.

7.4.3.2 Adaptive routing

Fig. 7.7 shows throughput and packet delay results for the SF-A routing strategy when different values of c_{SF} and n_I are used. SF-A manages to match the performance of minimal routing under uniform random traffic, and exceeds indirect random routing for the worst-case, by adaptively selecting both minimal and indirect paths. Overall, varying the number of indirect routes considered at each packet injection (n_I) affects slightly the throughput saturation point under worst-case traffic, with higher numbers providing better results, as more available



(a) Varying n_I ($c_{SF} = 1$).

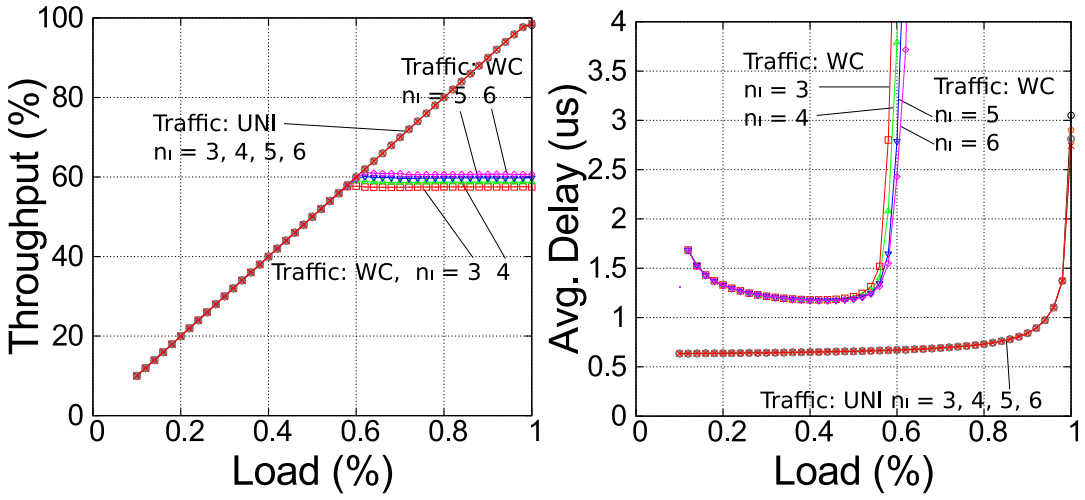


(b) Varying c_{SF} ($n_I = 4$).

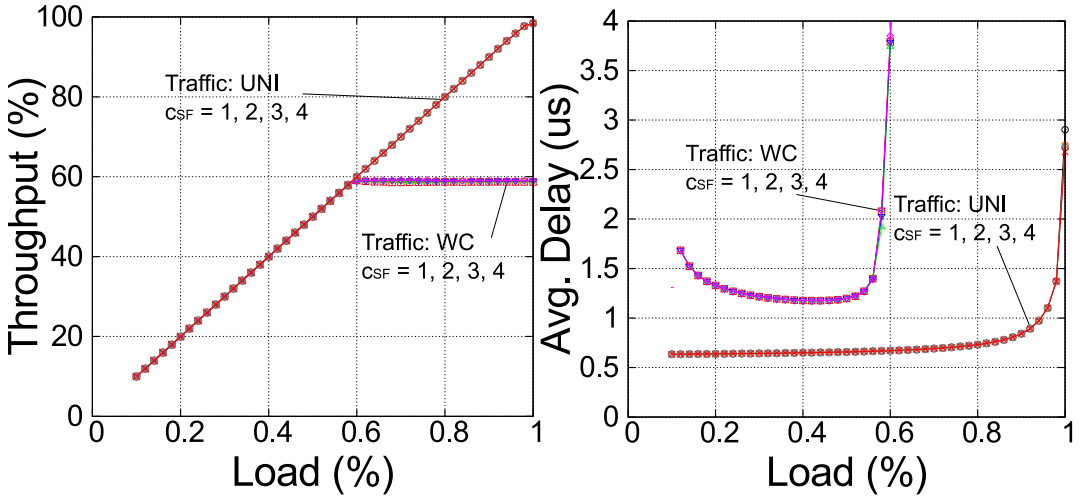
Figure 7.7 – SF-A routing on the SF with $p = \lfloor \frac{r'}{2} \rfloor$, with various values for c_{SF} and c, under uniform random (UNI) and worst case (WC) traffic.
Figure included from [52]

routes are available. On the other hand, c_{SF} affects the average delay under high loads of uniform traffic. A low value of the paremeter results in indirect paths being selected easily and their increased length negatively affects the packet delay.

The drawback of the generic UGAL algorithm is apparent under uniform random traffic in the increase in latency for higher injection loads (higher than 50%). As the load increases, even with a very low occupancy on the minimal buffers, the algorithm turns to indirect routes, effectively increasing the average packet delay. The SF-ATh routing (Fig. 7.8) manages to alleviate this problem, keeping the delay in low levels by selecting only minimal routes.



(a) Varying n_I ($c_{SF} = 1$).



(b) Varying c_{SF} ($n_I = 4$).

Figure 7.8 – SF-ATh routing on the SF with $p = \lfloor \frac{r'}{2} \rfloor$, with various values for c_{SF} and c, under uniform random (UNI) and worst case (WC) traffic ($T = 10\%$).
Figure included from [52]

However, the threshold has a negative effect in adversarial traffic, increasing the latency for low loads, as a result of $T = 10\%$ of the packets in each port being routed minimally.

Compared to the SF-A, the effects of the MLFM-A routing algorithm on its respective topology are more sensitive to the parameters c and n_I (Fig. 7.9). Although the throughput achieves the levels of minimal and indirect random routing, the delay depends heavily on both parameters: higher values for c and n_I provide lower latency under uniform random traffic, meaning that the algorithm requires a variety of choices for indirect paths to work effectively, but needs to select them with strict criteria. On the contrary, lower values for c and n_I appear best in the worst-case.

With carefully selected parameters, MLFM-A does not exhibit the symptom of the the generic UGAL algorithm (higher delay in higher loads of uniform random traffic). Thus, the MLFM-Ath algorithm (Fig. 7.11) does not provide any significant benefit under uniform random traffic, other than parameter-independence. Nevertheless, the effect of increased delay for low-load worst-case traffic is once again apparent, similarly to the SF case.

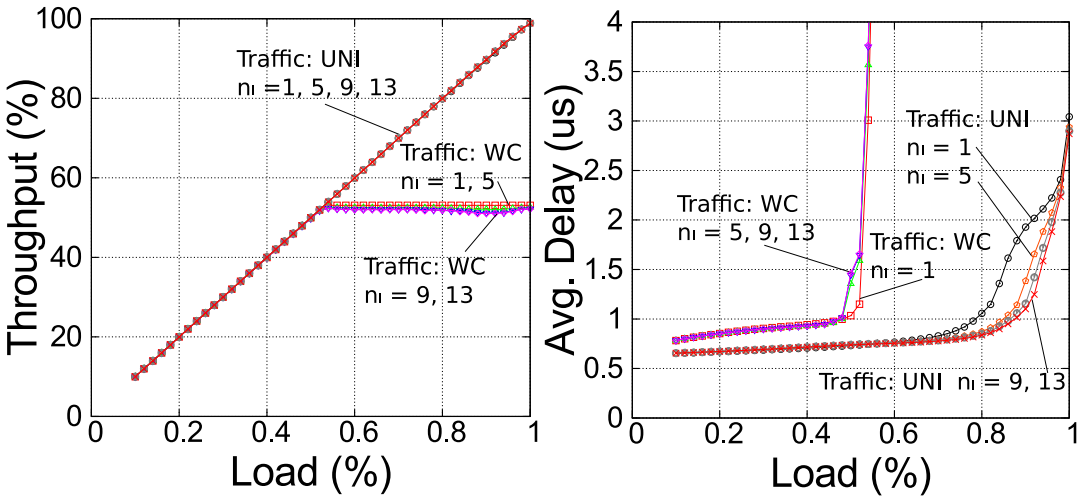
Fig. 7.10 shows throughput and packet delay results for the OFT-A routing strategy when different values of c and n_I are used. Contrary to MLFM-A, OFT-A offers the lowest delay under uniform random traffic when the selection of indirect paths is constricted (low values of n_I and high values of c). Nevertheless, the performance under worst-case traffic appears mostly independent of the routing algorithm parameters. Once again, adaptive routing with a threshold (OFT-ATh, Fig. 7.12) manages to lower the delay in uniform random traffic by trading off a higher delay in low load levels of worst-case traffic.

7.4.4 Exchange patterns

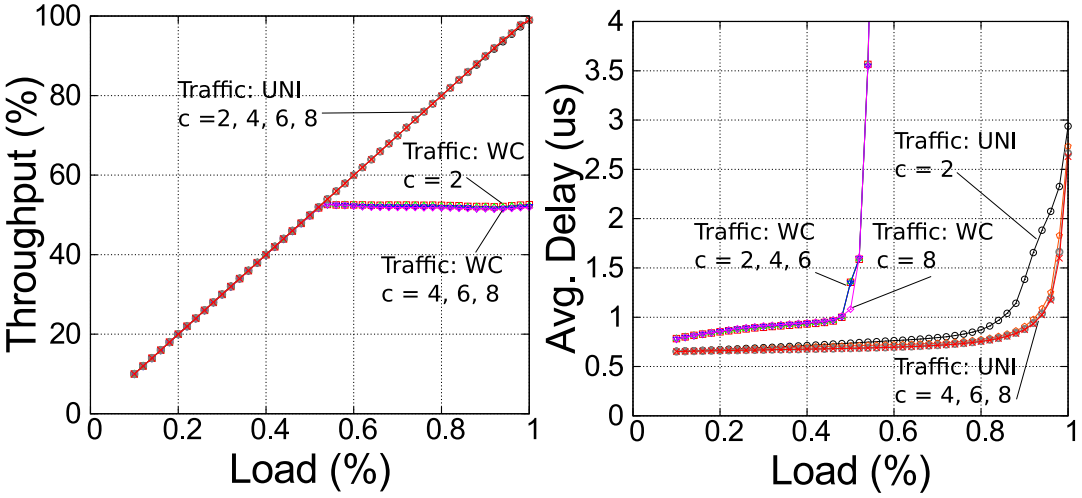
Synthetic traffic patterns, as informative as they are about a topology's limitations, are rarely encountered in real-life applications. As such, we also experimented with two prevalent data exchange patterns, the *All-to-All* (A2A) and the *Nearest-Neighbor* (NN) exchange. In the former, each process sends one message to each other process, meaning that N^2 messages are exchanged in total (a node is assigned a single process). For the latter, the processes are arranged in a 3D Torus (the largest one that fits in each topology), and each process sends one message to each of its 6 neighbors. The tori used have the following sizes: $12 \times 14 \times 19$ (OFT), $15 \times 16 \times 15$ (MLFM), $13 \times 13 \times 18$ (SF, $p = 9$) and $13 \times 13 \times 20$ (SF, $p = 10$). We chose a contiguous mapping, in which processes consecutive in dimension order in the application domain are mapped to consecutive end-nodes in the network. The order of the end-nodes in the network is derived from the morphology of each topology: the nodes are ordered consecutively, first at the intra-router level, then at the intra-column (SF, Fig. 7.1a)/intra-layer(MLFM,OFT) level and finally at the subgraph(SF)/inter-layer(MLFM,OFT) level. The routing strategies compared for each topology are the MIN, INR, and the adaptive configuration that showed the best performance under synthetic traffic.

Fig. 7.13 shows the effective throughput achieved when performing a single A2A exchange for each topology. The exchange is performed in a manner similar to that described by Kumar et al. [57]. We calculate the effective throughput of the exchange by dividing the total amount of data exchanged by the completion time (the time interval between the moment when the first message is injected in the network and the moment when the last message in the network reaches its destination). The result is normalized per end-node and expressed as a percentage of the maximum injection bandwidth.

The performance of each of the three topologies is similar, with the SF ($p = 9$), MLFM and OFT exhibiting an effective throughput close to 100% for both minimal and adaptive routing strategies and half of that for indirect random routing, similar to what we observed before



(a) Varying n_I ($c = 2$).

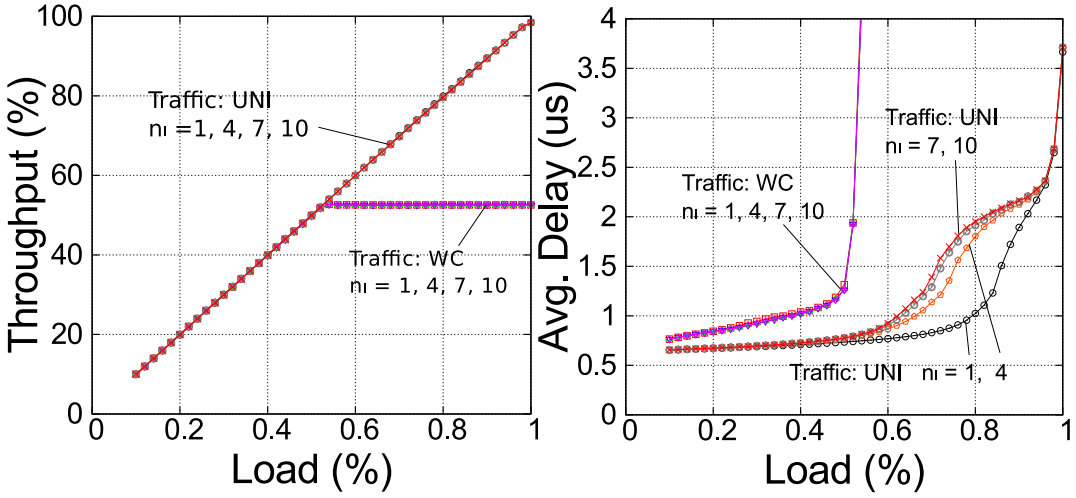


(b) Varying c ($n_I = 5$).

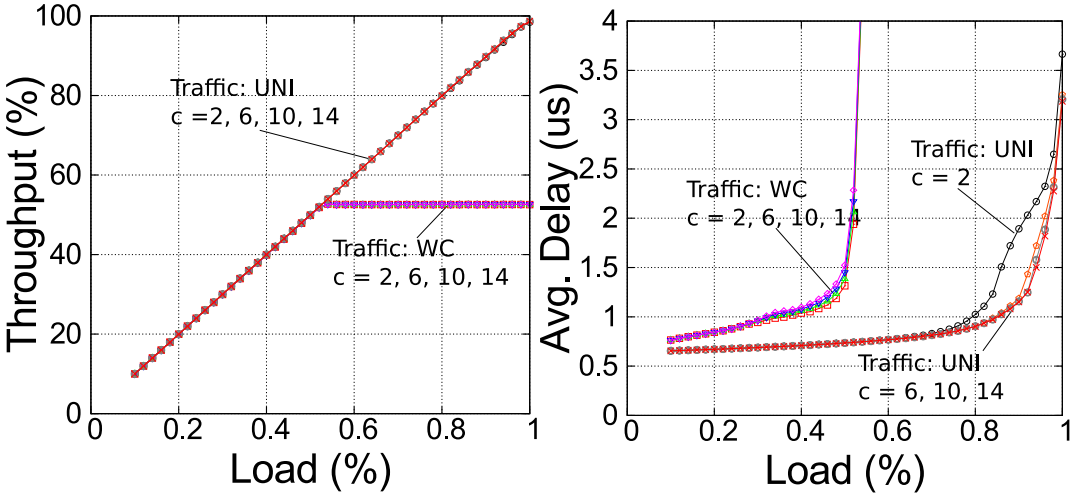
Figure 7.9 – MLFM-A routing, with various values for c and n_I, under uniform random (UNI) and worst case (WC) traffic.
Figure included from [52]

for uniform traffic. The main difference between these results and those obtained for the synthetic pattern is that here we measure the performance of a fixed load (as opposed to steady state throughput) and thus we would have also captured negative tail effects should they have occurred. The fact that the effective throughput is almost identical to the steady state throughput is a strong indicator that such tail effects are negligible.

Fig. 7.14 shows the respective results for the NN exchange. In this exchange, minimal routing has very low performance for all topologies, as only a few routes are available to accommodate the traffic from a large number of processes. Indirect random routing achieves load balancing across the network, leading to higher effective throughput. The close to 70% effective through-



(a) Varying n_I ($c = 2$).

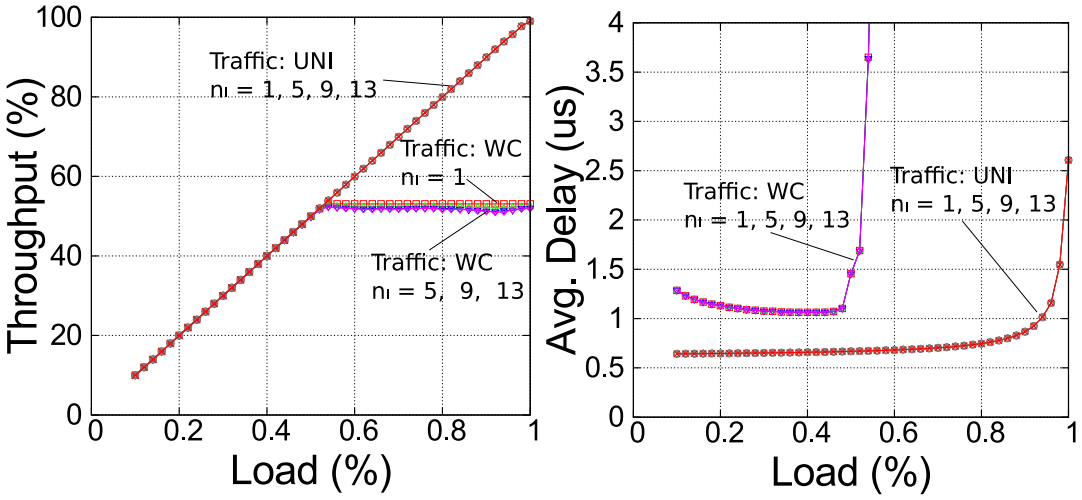


(b) Varying c ($n_I = 1$).

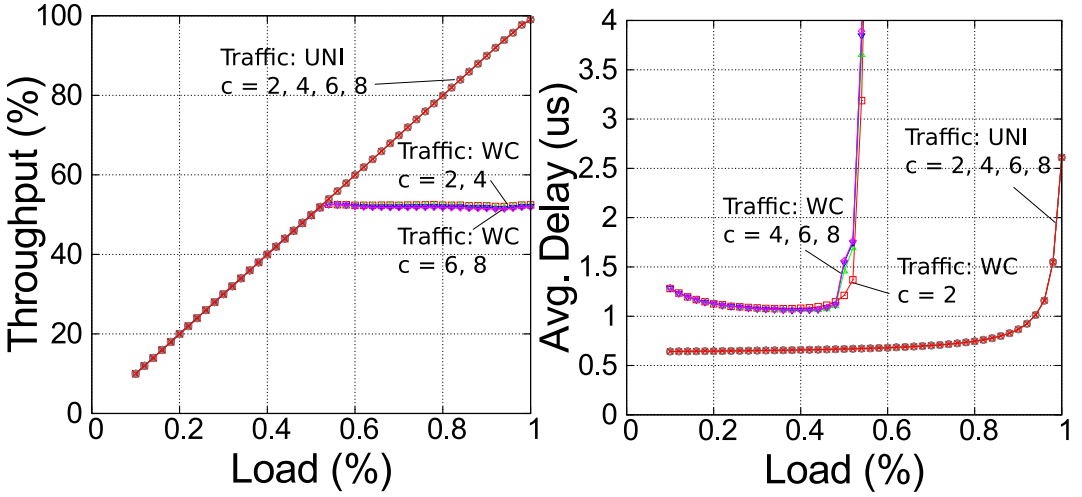
Figure 7.10 – OFT-A routing, with various values for c and n_I, under uniform random (UNI) and worst case (WC) traffic.
Figure included from [52]

put obtained is explained by the X exchanges (which stay within the first router) achieving 100% throughput while the Y and Z exchanges achieve the 50% expected of indirect random routing.

The adaptive routing schemes are generally able to improve on the performance for indirect random routing, with the exception of the OFT. For the MLFM in particular, adaptive routing achieves close to 100% effective throughput. The contiguous mapping readily maps the 3 dimensions of the 3D Torus to the three dimensional structure of the topology: the X exchanges take place inside a single router, the Y exchanges take place inside a single layer, and the Z exchanges take place across a router column. The adaptive algorithm decides to



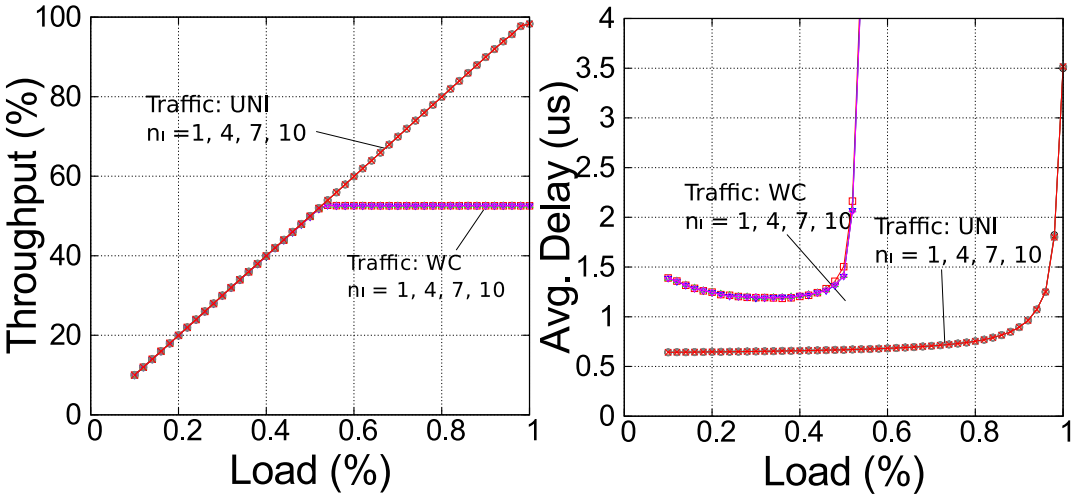
(a) Varying n_I ($c = 2$).



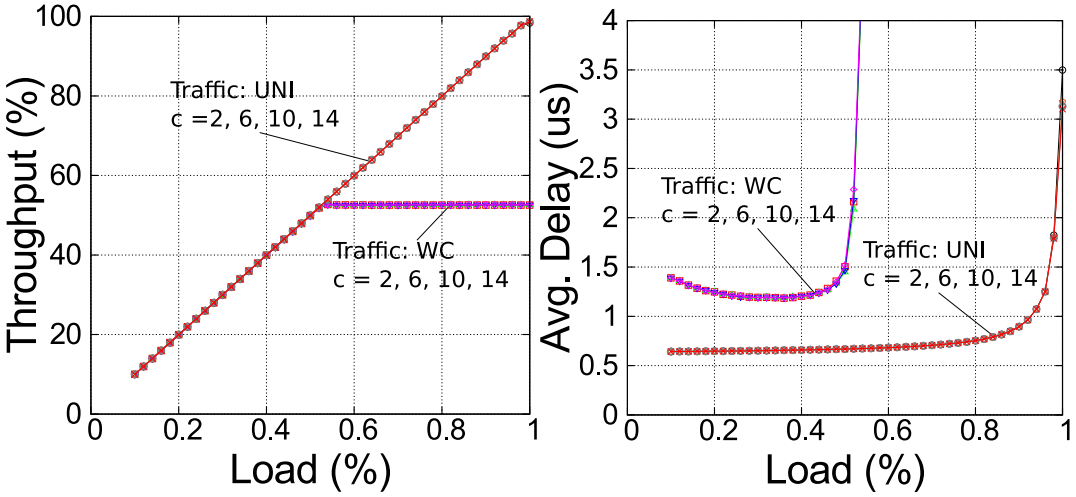
(b) Varying c ($n_I = 5$).

Figure 7.11 – MLFM-ATh routing (various c, n_I) under uniform random (UNI) and worst case (WC) traffic ($T = 10\%$).
Figure included from [52]

route minimally, indirectly, and minimally, respectively, achieving full bandwidth. The same effect is not witnessed on the OFT, as the Torus that would fit the topology in the same way would be the highly impractical: $12 \times 133 \times 2$ one. For the SF, fitting the Torus exactly on the topology would not be a trivial task given its complex structure, but nevertheless, the contiguous mapping is sufficient to allow the adaptive routing to outperform indirect random routing by $\approx 20\%$.



(a) Varying n_I ($c = 2$).



(b) Varying c ($n_I = 1$).

Figure 7.12 – OFT-ATh routing (various c, n_I) under uniform random (UNI) and worst case (WC) traffic ($T = 10\%$).
Figure included from [52]

7.5 Summary

In this chapter we survey the options for cost-effective diameter-two network topologies. We consider both direct topologies, in particular the Slim Fly, and indirect topologies, where we introduce a more scalable alternative to the traditional two-level Fat-Tree, the family of Stacked Single-Path Trees of which existing designs such as the Multi-Layer Full-Mesh and the two-level Orthogonal Fat-Tree are particular members. For the latter two, we introduce mechanisms for routing and deadlock avoidance.

Through theoretical analysis of the characteristics of these networks, we showed that all three exhibit several useful properties: a per-endpoint cost of only two links and three router ports

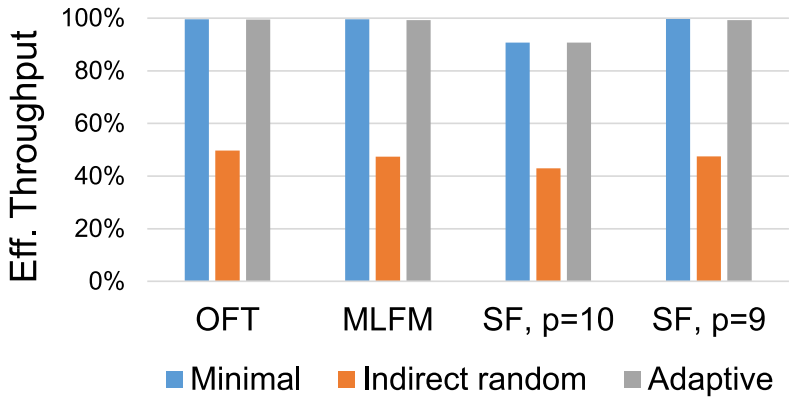


Figure 7.13 – Effective throughput for one all-to-all exchange, with different routing strategies.
Figure included from [52]

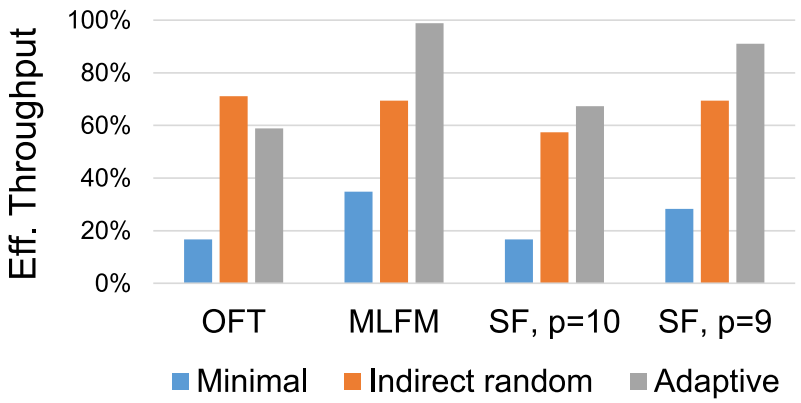


Figure 7.14 – Effective throughput for one nearest-neighbor exchange, with different routing strategies.
Figure included from [52]

(an improvement of more than 30% over the three-level Fat-Tree design), a low end-to-end network latency (at most three switch traversals for any minimal path), full global bandwidth, and reasonable scale (33K-64K end-nodes using radix-64 routers).

Furthermore, we identified for each topology adversarial traffic patterns for which performance is the lowest possible and presented mechanisms to mitigate this effect, mainly in the shape of indirect routing approaches. Finally, through detailed simulations, we showed that close to ideal levels of performance can be attained for each design for both best-case (global uniform) and worst-case (adversarial) traffic as well as good performance for traffic patterns that are representative of real world applications (nearest neighbor and all-to-all exchanges). Instrumental in achieving these performance levels was the use of adaptive routing

mechanisms. We showed that with proper tuning, such mechanisms are able to handle a wide range of communication behavior by seamlessly switching from minimal to indirect routing, as needed.

All in all, we have shown that both the diameter-two Slim Fly (as the best overall direct topology) and the two level Orthogonal Fat-Tree (as the best overall indirect topology) i) exhibit good if not ideal performance across a wide range of traffic patterns, ii) have a degree of scalability that is compatible with the requirements for future datacenter and HPC interconnects, and iii) achieve this at a very low cost, especially compared to current options such as three-level Fat-Trees or Dragonflies. The choice between the two hinges mainly on the tradeoff between, on the one side, the lower cost characteristic of direct topologies (where the routers can be integrated close to the nodes), and on the other side, the factor of two higher scalability that can be achieved with OFT (surprisingly allowing it to accommodate as many end-nodes as a three level Fat-Tree at the cost of a two-level Fat-Tree).

Conclusions

In this thesis we set out to show that, in the context of High Performance Computing systems and applications, co-optimizing the system (e.g., topological configuration of compute nodes and switches, routing algorithms) and workload (e.g., data exchange patterns between the individual processes, application deployment patterns) configuration leads to significant performance and cost benefits.

We started with considering the case of Fat Tree topologies, one of the more prevalent system layouts today, and in terms of workload class, started by focusing on applications where the dominating communication pattern is the personalized all-to-all collective message exchange.

In this context:

- we derived a theoretical lower bound on the network link occupation that is intrinsically necessary and sufficient for any all-to-all exchange;
- we formally showed that existing approaches require as much as twice the theoretical minimum;
- motivated by this insight that headroom exists, we introduced a novel way to orchestrate the data exchange that is bandwidth optimal and (requires strictly the minimum bound).

The resulting better utilization of the network resources allows for improved application performance under constrained (underprovisioned) hardware resources, or conversely, opens up new workload-tuned design options for Fat Tree systems with significantly reduced cost but minimal impact on performance. These results were both formally proven and validated by conducting detailed network simulations.

Remaining in the realm of Fat Trees, but now leaving the workload aspect arbitrary, we continued by proposing a generic way to adapt the routing strategy such that it is optimally tailored to both the specificities of the network topology and those of the application communication pattern.

Finding optimal routes is traditionally an extremely computationally intensive optimization process. Starting from the established idea of modeling this problem as an integer linear

optimization (the solving of which is prohibitively slow even for very small network sizes), we broke down the problem in a novel way, from system-wide to per-fat-tree-level wide, and came up with a way of ensuring that these local optima combine to form a globally optimal routing by introducing a novel set of additional constraints. Together, these innovations added-up to a new theoretical modeling of the routing optimization which allowed scaling the process to reasonable time frames, even for the largest system sizes found in practice.

Fixing the workload once more to the personalized all-to-all collective message exchange, we showed next that across a wide array of network graphs (what we call hierarchical interconnection network architectures), notably including both Fat Trees and Dragofly layouts, the data exchange can once more be orchestrated in a hardware-aware, hierarchical way, to achieve significant reduction in completion time compared to the state of the art (between 1.8x and 3x faster).

Fixing now the Dragonfly topology, we next quantified the extent of performance improvements that can be achieved by adapting the task placement and routing to topology and application aware. To do so, we introduced a novel theoretical framework that is able to identify the bottlenecks that appear in the network under arbitrary workloads (specified via their traffic demand matrix), as well as to determine how those bottlenecks impact the effective throughput of the nodes. By providing a very fast way to estimate performance for any given combination of task placement and routing, this framework can be used effectively to determine the *best* combination of such settings for any given workload (as characterized by its communication pattern).

As an additional contribution on this topic, to showcase the capabilities of the framework, we also showed how it can be leveraged to tune common communication patterns in High Performance computing - nearest neighbor exchanges of various shapes, as well as uniform traffic - for maximum performance on a variety of Dragonfly networks.

Finally, the last key contribution was to show that by diverging even more aggresively from established topological configurations, significant cost savings can be attained without sacrificing neither application performance or scalability. Specifically, through extensive benchmarking using detailed network simulations, we have shown that diameter-two network topologies, in particular Slim Fly (as the best overall direct topology) and the two level Orthogonal Fat-Tree (as the best overall indirect topology) i) exhibit good if not ideal performance across a wide range of traffic patterns, ii) have a degree of scalability that is compatible with the requirements for future datacenter and High Performance Computing interconnects, and iii) achieve this at a very low cost compared to three-level Fat-Trees or Dragonflies.

To summarize, through these individual contributions I believe we have been successful in showing that there is ample and within reach headroom for performance and/or cost optimization when the application and the system enabling it can be tightly co-designed or at least co-configured. As the need for ever more complex workloads grows, the computational capabilities of High Performance Computing systems continue to increase but so do power

consumption and cost concerns. As such, I believe directions such as those explored in this thesis, of more tightly integrating hardware and software will play a key role in reaching exa-scale and beyond.

Publications and Patents

Journal Peer-Reviewed Articles

Bogdan Prisacari, Germán Rodríguez, Ana Jokanovic, Cyriel Minkenberg
Randomizing task placement and route selection do not randomize traffic (enough).
Design Automation for Embedded Systems, DAES 2014

Bogdan Prisacari, Germán Rodríguez, Cyriel Minkenberg, Torsten Hoefler
Fast pattern-specific routing for fat tree networks.
Transactions on Architecture and Code Optimization, TACO 2013

Conference Peer-Reviewed Articles

Cyriel Minkenberg, Germán Rodríguez, **Bogdan Prisacari**, Laurent Schares,
Philip Heidelberger, Dong Chen, Craig Stunkel
Performance Benefits of Optical Circuit Switches for Large-Scale Dragonfly Networks.
Optical Fiber Communication Conference, OFC 2016

Bogdan Prisacari, Germán Rodríguez, Cyriel Minkenberg, Marina García, Enrique Vallejo,
Ramón Beivide
Performance optimization of load imbalanced workloads in large scale Dragonfly systems.
International Conference on High Performance Switching and Routing, HPSR 2015

Georgios Kathareios, Cyriel Minkenberg, **Bogdan Prisacari**, Germán Rodríguez,
Torsten Hoefler
Cost-effective diameter-two topologies: analysis and evaluation.
International Conference for High Performance Computing, Networking, Storage and Analysis,
SC 2015

Andreea Anghel, Germán Rodríguez, **Bogdan Prisacari**, Cyriel Minkenberg, Gero Dittmann
Quantifying Communication in Graph Analytics.
International Conference on High Performance Computing, ISC 2015

Bogdan Prisacari, Germán Rodríguez, Philip Heidelberger, Dong Chen, Cyriel Minkenberg,

Torsten Hoefler
Efficient task placement and routing of nearest neighbor exchanges in dragonfly networks.
International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2014

Andreea Anghel, Germán Rodríguez, **Bogdan Prisacari**
The importance and characteristics of communication in high performance data analytics.
International Symposium on Workload Characterization, IISWC 2014

Bogdan Prisacari, Germán Rodríguez, Cyriel Minkenberg, Torsten Hoefler
Bandwidth-optimal all-to-all exchanges in fat tree networks.
International Conference on Supercomputing, ICS 2013

Bogdan Prisacari, Germán Rodríguez, Cyriel Minkenberg
Generalized Hierarchical All-to-All Exchange Patterns.
International Symposium on Parallel and Distributed Processing, IPDPS 2013

Bogdan Prisacari, Germán Rodríguez, Cyriel Minkenberg, Ramón Beivide
Performance implications of deadlock avoidance techniques in torus networks.
International Conference on High Performance Switching and Routing, HPSR 2012

Workshop Peer-Reviewed Articles

Dong Chen, Philip Heidelberger, Craig B. Stunkel, Yutaka Sugawara, Cyriel Minkenberg, **Bogdan Prisacari**, Germán Rodríguez
An Evaluation of Network Architectures for Next Generation Supercomputers.
International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, PMBS@SC 2016

Ana Jokanovic, **Bogdan Prisacari**, Germán Rodríguez, Cyriel Minkenberg
Randomizing task placement does not randomize traffic (enough).
International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip, INA-OCMC@HiPEAC 2013

Patents

Enablement of quota-based quality of service
Inventors: Wolfgang E. Denzel, Cyriel J. Minkenberg, **Bogdan Prisacari**, German Rodriguez Herrera
Assignee: International Business Machines Corporation
Patent number: 10284682

All-to-all message exchange in parallel computing systems

Inventors: Cyriel J. Minkenberg, **Bogdan Prisacari**, German Rodriguez Herrera
Assignee: International Business Machines Corporation
Patent number: 10042683

Datacenter scheduling of applications using machine learning techniques
Inventors: Andreea S. Anghel, **Bogdan Prisacari**, German Rodriguez Herrera
Assignee: International Business Machines Corporation
Patent number: 9720738

Interconnection network topology for large scale high performance computing (HPC) systems
Inventors: Baba L. Arimilli, Wolfgang Denzel, Philip Heidelberger, German Rodriguez Herrera, Christopher J. Johnson, Lonny Lambrecht, Cyriel Minkenberg, **Bogdan Prisacari**
Assignee: International Business Machines Corporation
Patent number: 9626322

Mechanisms for deadlock avoidance support in network fabrics
Inventors: Wolfgang E. Denzel, German R. Herrera, Cyriel J. Minkenberg, **Bogdan Prisacari**
Assignee: International Business Machines Corporation
Patent number: 9509613

Bibliography

[1] Top 500 list. http://www.top500.org/list/2013/06/, June 2013. Accessed: 2013-09-10.

[2] Top 500 list. http://www.top500.org/list/2015/06/, June 2015. Accessed: 2015-07-27.

[3] AHN, J. H., BINKERT, N., DAVIS, A., MCLAREN, M., AND SCHREIBER, R. S. HyperX: Topology, routing, and packaging of efficient large-scale networks. In *Proceedings of the International Conference on High Performance Computing Networking, Storage and Analysis* (New York, NY, USA, 2009), SC '09, ACM, pp. 41:1–41:11.

[4] AL-FARES, M., LOUKISSAS, A., AND VAHDAT, A. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review* (2008), vol. 38, ACM, pp. 63–74.

[5] ALAM, S. R., AND VETTER, J. S. An analysis of system balance requirements for scientific applications. In *International Conference on Parallel Processing, 2006. ICPP 2006.* (2006), pp. 229–236.

[6] ALEXANDROV, A., IONESCU, M. F., SCHAUSER, K. E., AND SCHEIMAN, C. LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation. In *Proceedings of the* 7^{th} *annual ACM symposium on Parallel algorithms and architectures (SPAA)* (NY, USA, 1995), ACM, pp. 95–105.

[7] ALMASI, G., HARGROVE, P., GABRIEL, I., AND ZHENG, T. Y. UPC collectives library 2.0. In 5^{th} *Conference on Partitioned Global Address Space Programming Models* (2011).

[8] ALVANOS, M., TANASE, G., FARRERAS, M., TIOTTO, E., AMARAL, J. N., AND MARTORELL, X. Improving performance of all-to-all communication through loop scheduling in pgas environments. In *Proceedings of the 27th international ACM conference on International conference on supercomputing* (New York, NY, USA, 2013), ICS '13, ACM, pp. 457–458.

[9] APPLEGATE, D., AND COHEN, E. Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2003), SIGCOMM '03, ACM, pp. 313–324.

[10] ARIMILLI, B., ARIMILLI, R., CHUNG, V., CLARK, S., DENZEL, W., DRERUP, B., HOEFLER, T., JOYNER, J., LEWIS, J., LI, J., NI, N., AND RAJAMONY, R. The PERCS High-Performance Interconnect. In *Proc. of the* 18^{th} *IEEE Symposium on High Performance Interconnects* (Washington, DC, USA, 2010), HOTI '10, IEEE Computer Society, pp. 75–82.

[11] AZAR, Y., COHEN, E., FIAT, A., KAPLAN, H., AND RÄCKE, H. Optimal oblivious routing in polynomial time. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing* (New York, NY, USA, 2003), STOC '03, ACM, pp. 383–388.

[12] BAILEY, D., HARRIS, T., SAPHIR, W., VAN DER WIJNGAART, R., WOO, A., AND YARROW, M. The NAS Parallel Benchmarks 2.0. *The International Journal of Supercomputer Applications* (1995).

[13] BALAJI, P., BUNTINAS, D., GOODELL, D., GROPP, W., HOEFLER, T., KUMAR, S., LUSK, E., THAKUR, R., AND TRAEFF, J. L. MPI on Millions of Cores. *Parallel Processing Letters (PPL) 21*, 1 (Mar. 2011), 45–60.

[14] BERGMAN, K., BORKAR, S., CAMPBELL, D., CARLSON, W., DALLY, W., DENNEAU, M., FRANZON, P., HARROD, W., HILLER, J., KARP, S., KECKLER, S., KLEIN, D., LUCAS, R., RICHARDS, M., SCARPELLI, A., SCOTT, S., SNAVELY, A., STERLING, T., WILLIAMS, R. S., YELICK, K., BERGMAN, K., BORKAR, S., CAMPBELL, D., CARLSON, W., DALLY, W., DENNEAU, M., FRANZON, P., HARROD, W., HILLER, J., KECKLER, S., KLEIN, D., KOGGE, P., WILLIAMS, R. S., AND YELICK, K. Exascale computing study: Technology challenges in achieving exascale systems, 2008.

[15] BESTA, M., AND HOEFLER, T. Slim Fly: A Cost Effective Low-Diameter Network Topology. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC14* (Nov. 2014), IEEE, pp. 348–359.

[16] BHATELE, A., JAIN, N., GROPP, W. D., AND KALE, L. V. Avoiding hot-spots on two-level direct networks. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 76:1–76:11.

[17] BHUYAN, L. N., AND AGRAWAL, D. P. Generalized hypercube and hyperbus structures for a computer network. *IEEE Trans. Comput. 33*, 4 (Apr. 1984), 323–333.

[18] BOGDANSKI, B., SEM-JACOBSEN, F. O., REINEMO, S.-A., SKEIE, T., HOLEN, L., AND HUSE, L. P. Achieving predictable high performance in imbalanced fat trees. In *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on* (2010), IEEE, pp. 381–388.

[19] BOKHARI, S. H. Multiphase complete exchange on a circuit switched hypercube. In *ICPP (1)'91* (1991), pp. 525–529.

[20] BRUCK, J., HO, C.-T., UPFAL, E., KIPNIS, S., AND WEATHERSBY, D. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Trans. Parallel Distrib. Syst. 8*, 11 (Nov. 1997), 1143–1156.

[21] CHAN, E., HEIMLICH, M., PURKAYASTHA, A., AND VAN DE GEIJN, R. On optimizing collective communication. In *Cluster Computing, 2004 IEEE International Conference on* (sept. 2004), pp. 145 – 155.

[22] DALLY, W., AND TOWLES, B. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[23] DÉNES, J., AND KEEDWELL, A. *Latin squares and their applications*. Academic Press, 1974.

[24] DING, Z., HOARE, R. R., JONES, A. K., AND MELHEM, R. Level-wise scheduling algorithm for fat tree interconnection networks. In *Proc. 2006 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 2006), ACM, p. 96.

[25] ELMALLAH, E. S., AND CULBERSON, J. C. Multicommodity flows in simple multistage networks. *Networks 25* (1994), pp.

[26] EVEN, S., ITAI, A., AND SHAMIR, A. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM Journal on Computing 5*, 4 (1976), 691–703.

[27] FAANES, G., BATAINEH, A., ROWETH, D., COURT, T., FROESE, E., ALVERSON, B., JOHNSON, T., KOPNICK, J., HIGGINS, M., AND REINHARD, J. 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* (Los Alamitos, CA, USA, 2012), SC '12, IEEE Computer Society Press, pp. 103:1–103:9.

[28] FIEDLER, R. Preparing Applications for Sustained Petascale Performance, 2011.

[29] FLICH, J., MALUMBRES, M. P., LÓPEZ, P., AND DUATO, J. Improving routing performance in Myrinet networks. In *Proc. of the* 14^{th} *International Parallel and Distributed Processing Symposium* (Los Alamitos, CA, USA, 2000), IEEE Computer Society, pp. 27–32.

[30] FLICH, J., SKEIE, T., MEJÍA, A., LYSNE, O., LÓPEZ, P., ROBLES, A., DUATO, J., KOIBUCHI, M., ROKICKI, T., AND SANCHO, J. C. A survey and evaluation of topology-agnostic deterministic routing algorithms. *IEEE Transactions on Parallel and Distributed Systems. 23*, 3 (2012), 405–425.

[31] FUJITSU LABORATORIES LTD. Fujitsu Laboratories Develops Technology to Reduce Network Switches in Cluster Supercomputers by 40%. https://www.fujitsu.com/global/about/resources/news/press-releases/2014/0715-02.html, 2014. Accessed: 2015-04-16.

[32] GARCÍA, M., VALLEJO, E., BEIVIDE, R., ODRIOZOLA, M., CAMARERO, C., VALERO, M., RODRÍGUEZ, G., LABARTA, J., AND MINKENBERG, C. On-the-fly adaptive routing in

high-radix hierarchical networks. In *Proceedings of the 41st International Conference on Parallel Processing (ICPP)* (09 2012).

[33] GEOFFRAY, P., AND HOEFLER, T. Adaptive routing strategies for modern high performance networks. In *High Performance Interconnects, 2008. HOTI '08. 16th IEEE Symposium on* (Aug. 2008), pp. 165–172.

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

[35] GOODALE, T., ALLEN, G., LANFERMANN, G., MASSÓ, J., RADKE, T., SEIDEL, E., AND SHALF, J. The Cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science* (Berlin, 2003), Springer.

[36] GORLATCH, S. Send-receive considered harmful: Myths and realities of message passing. *ACM Trans. Program. Lang. Syst. 26*, 1 (Jan. 2004), 47–56.

[37] GREENBERG, R. I., AND LEISERSON, C. E. Randomized routing on fat-trees. In *Proc. of the* 26^{th} *Annual Symposium on the Foundations of Computer Science* (1985), pp. 241–249.

[38] HOEFLER, T., AND LUMSDAINE, A. Optimizing non-blocking collective operations for InfiniBand. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* (april 2008), pp. 1 –8.

[39] HOEFLER, T., LUMSDAINE, A., AND REHM, W. Implementation and performance analysis of non-blocking collective operations for MPI. In *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on* (nov. 2007), pp. 1 –10.

[40] HOEFLER, T., RABENSEIFNER, R., RITZDORF, H., DE SUPINSKI, B. R., THAKUR, R., AND TRAEFF, J. L. The Scalable Process Topology Interface of MPI 2.2. *Concurrency and Computation: Practice and Experience 23*, 4 (Aug. 2010), 293–310.

[41] HOEFLER, T., SCHNEIDER, T., AND LUMSDAINE, A. The impact of network noise at large-scale communication performance. In *IEEE International Symposium on Parallel and Distributed Processing, 2009. IPDPS 2009.* (May 2009), pp. 1–8.

[42] HOEFLER, T., SCHNEIDER, T., AND LUMSDAINE, A. Optimized routing for large-scale InfiniBand networks. In *Proceedings of the 2009 17th IEEE Symposium on High Performance Interconnects* (Washington, DC, USA, 2009), HOTI '09, IEEE Computer Society, pp. 103–111.

[43] HUSBANDS, P., AND HOE, J. C. MPI-StarT: delivering network performance to numerical applications. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)* (Washington, DC, USA, 1998), Supercomputing '98, IEEE Computer Society, pp. 1–15.

[44] JAJSZCZYK, A. Nonblocking, repackable, and rearrangeable Clos networks: fifty years of the theory evolution. *Communications Magazine, IEEE 41*, 10 (Oct. 2003), 28–33.

[45] JOHNSON, N. L., AND KOTZ, S. *Urn models and their application: an approach to modern discrete probability theory*. Wiley New York, 1977.

[46] JOKANOVIC, A., PRISACARI, B., RODRIGUEZ, G., AND MINKENBERG, C. Randomizing task placement does not randomize traffic (enough). In *Proceedings of the 2013 Interconnection Network Architecture: On-Chip, Multi-Chip* (New York, NY, USA, 2013), IMA-OCMC '13, ACM, pp. 9–12.

[47] KALÉ, L. V., KUMAR, S., AND VARADARAJAN, K. A framework for collective personalized communication. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing* (Washington, DC, USA, 2003), IPDPS '03, IEEE Computer Society, pp. 69.1–.

[48] KAMIL, S., SHALF, J., OLIKER, L., AND SKINNER, D. Understanding ultra-scale application communication requirements. *Proc. Workload Characterization Symposium* (Oct. 2005), 178–187.

[49] KARINIEMI, H. *On-Line Reconfigurable Extended Generalized Fat Tree Network-on-Chip for Multiprocessor System-on-Chip Circuits*. PhD thesis, Tampere University of Technology, 2006.

[50] KARONIS, N., DE SUPINSKI, B., FOSTER, I., GROPP, W., LUSK, E., AND BRESNAHAN, J. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International* (2000), pp. 377 –384.

[51] KARYPIS, G., AND KUMAR, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing 20*, 1 (1998), 359–392.

[52] KATHAREIOS, G., MINKENBERG, C., PRISACARI, B., RODRÍGUEZ, G., AND HOEFLER, T. Cost-effective diameter-two topologies: analysis and evaluation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015* (2015), ACM, pp. 36:1–36:11.

[53] KIM, J., DALLY, W., SCOTT, S., AND ABTS, D. Cost-Efficient Dragonfly Topology for Large-Scale Systems. *Micro, IEEE 29*, 1 (Feb. 2009), 33–40.

[54] KIM, J., DALLY, W. J., SCOTT, S., AND ABTS, D. Technology-driven, highly-scalable dragonfly topology. *SIGARCH Comput. Archit. News 36*, 3 (June 2008), 77–88.

[55] KINSY, M. A., CHO, M. H., WEN, T., SUH, E., VAN DIJK, M., AND DEVADAS, S. Application-aware deadlock-free oblivious routing. In *Proceedings of the 36th annual international symposium on Computer architecture* (New York, NY, USA, 2009), ISCA '09, ACM, pp. 208–219.

[56] KINSY, M. A., CHO, M. H., WEN, T., SUH, E., VAN DIJK, M., AND DEVADAS, S. Application-aware deadlock-free oblivious routing. *SIGARCH Comput. Archit. News 37*, 3 (June 2009), 208–219.

[57] KUMAR, S., MAMIDALA, A., HEIDELBERGER, P., CHEN, D., AND FARAJ, D. Optimization of MPI collective operations on the IBM Blue Gene/Q supercomputer. *Int. J. High Perform. Comput. Appl. 28*, 4 (Nov. 2014), 450–464.

[58] KURMANN, C., RAUCH, F., AND STRICKER, T. M. Cost/performance tradeoffs in network interconnects for clusters of commodity PCs. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing* (Washington, DC, USA, 2003), IPDPS '03, IEEE Computer Society, pp. 196.2–.

[59] LEIGHTON, T., MAKEDON, F., PLOTKIN, S., STEIN, C., TARDOS, E., AND TRAGOUDAS, S. Fast approximation algorithms for multicommodity flow problems. In *Journal of Computer And System Sciences* (1991), pp. 487–496.

[60] LEISERSON, C., ABUHAMDEH, Z. S., DOUGLAS, D. C., FEYNMAN, C. R., GANMUKHI, M. N., HILL, J. V., HILLIS, W. D., KUSZMAUL, B. C., PIERRE, M. A. S., WELLS, D. S., WONG-CHAN, M. C., WEN YANG, S., AND ZAK, R. The network architecture of the Connection Machine CM-5. In *Proc. of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures* (San Diego, CA, USA, June 1992), pp. 272–285.

[61] LIN, X.-Y., CHUNG, Y.-C., AND HUANG, T.-Y. A multiple LID routing scheme for fat-tree-based InfiniBand networks. *Proc. of the* 18^{th} *International Parallel and Distributed Processing Symposium* (2004), 11–.

[62] LIN, Z., ETHIER, S., HAHM, T. S., AND TANG, W. M. Size scaling of turbulent transport in magnetically confined plasmas. *Phys. Rev. Lett. 88*, 19 (2002), 195004–.

[63] MAKHORIN, A. GNU linear programming kit, 2013. [Online; accessed 06-June-2013].

[64] MAMIDALA, A., KUMAR, R., DE, D., AND PANDA, D. MPI collectives on modern multicore clusters: Performance optimizations and communication characteristics. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on* (may 2008), pp. 130 –137.

[65] MARTÍNEZ, J. C., FLICH, J., ROBLES, A., LÓPEZ, P., AND DUATO, J. Supporting fully adaptive routing in InfiniBand networks. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing* (Washington, DC, USA, 2003), IPDPS '03, IEEE Computer Society, pp. 44.1–.

[66] MCKAY, B. D., MILLER, M., AND ŠIRÁŇ, J. A note on large graphs of diameter two and given maximum degree. *Journal of Combinatorial Theory, Series B 74*, 1 (1998), 110–118.

[67] MELLOR-CRUMMEY, J., ADHIANTO, L., SCHERER, III, W. N., AND JIN, G. A new vision for coarray fortran. In *Proceedings of the* 3^{th} *Conference on Partitioned Global Address Space Programing Models* (New York, NY, USA, 2009), PGAS '09, ACM, pp. 5:1–5:9.

[68] MILLER, M., AND SIRAN, J. Moore graphs and beyond: A survey of the degree/diameter problem. *Electronic Journal of Combinatorics*, DS14 (2005).

[69] MINKENBERG, C., DENZEL, W., RODRIGUEZ, G., AND BIRKE, R. 11 end-to-end modeling and simulation of high-performance computing systems. *Springer Proceedings in Physics: Use Cases of Discrete Event Simulation: Appliance and Research* (2012), 201.

[70] MINKENBERG, C., AND RODRIGUEZ, G. Trace-driven co-simulation of high-performance computing systems using OMNeT++. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques* (ICST, Brussels, Belgium, 2009), Simutools '09, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[71] MOIN, P., AND MAHESH, K. Direct numerical simulation: a tool in turbulence research. *Annual Review of Fluid Mechanics 30*, 1 (1998), 539–578.

[72] MORITZ, C. A., AND FRANK, M. I. LoGPC: modeling network contention in message-passing programs. *SIGMETRICS Perform. Eval. Rev. 26*, 1 (June 1998), 254–263.

[73] MPI FORUM. MPI: A Message-Passing Interface Standard. Version 3.0, September 2012.

[74] NVIDIA. Summit and Sierra Supercomputers: An Inside Look at the U.S. Department of Energy's New Pre-Exascale Systems. http://www.teratec.eu/actu/calcul/Nvidia_Coral_White_Paper_Final_3_1.pdf, Nov. 2014.

[75] ÖHRING, S. R., IBEL, M., DAS, S. K., AND KUMAR, M. J. On generalized fat trees. In *Proceedings of the 9th International Symposium on Parallel Processing* (Washington, DC, USA, 1995), IPPS '95, IEEE Computer Society, pp. 37–.

[76] ÖHRING, S. R., IBEL, M., DAS, S. K., AND KUMAR, M. J. On generalized fat trees. In *Proceedings of the* 9^{th} *International Parallel Processing Symposium* (Washington, DC, USA, 1995), IEEE Computer Society, p. 37.

[77] PARK, H., HONG, S.-Y., CHEONG, H.-B., AND KOO, M.-S. A double fourier series (DFS) dynamical core in a global atmospheric model with full physics. *Monthly Weather Review 141* (2013), 3052–3061.

[78] PETRINI, F., KERBYSON, D. J., AND PAKIN, S. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing* (New York, NY, USA, 2003), SC '03, ACM, pp. 55–.

[79] PETRINI, F., AND VANNESCHI, M. A comparison of wormhole-routed interconnection networks. In *Proc. Third International Conference on Computer Science and Informatics* (Research Triangle Park, NC, USA, Mar. 1997).

[80] PETRINI, F., AND VANNESCHI, M. k-ary n-trees: High performance networks for massively parallel architectures. In *Proceedings of the 11th International Symposium on Parallel Processing* (Washington, DC, USA, 1997), IPPS '97, IEEE Computer Society, pp. 87–.

[81] PJEŠIVAC-GRBOVIĆ, J., ANGSKUN, T., BOSILCA, G., FAGG, G., GABRIEL, E., AND DONGARRA, J. Performance analysis of mpi collective operations. *Cluster Computing 10*, 2 (2007), 127–143.

[82] PRISACARI, B., RODRIGUEZ, G., GARCIA, M., VALLEJO, E., BEIVIDE, R., AND MINKENBERG, C. 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* (New York, NY, USA, 2014), INA-OCMC '14, ACM, pp. 5:1–5:4.

[83] PRISACARI, B., RODRÍGUEZ, G., HEIDELBERGER, P., CHEN, D., MINKENBERG, C., AND HOEFLER, T. Efficient task placement and routing of nearest neighbor exchanges in dragonfly networks. In *The 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC'14, Vancouver, BC, Canada - June 23 - 27, 2014* (2014), ACM, pp. 129–140.

[84] PRISACARI, B., RODRÍGUEZ, G., JOKANOVIC, A., AND MINKENBERG, C. Randomizing task placement and route selection do not randomize traffic (enough). *Design Automation for Embedded Systems 18*, 3-4 (2014), 171–182.

[85] PRISACARI, B., RODRÍGUEZ, G., AND MINKENBERG, C. Generalized hierarchical all-to-all exchange patterns. In *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013* (2013), IEEE Computer Society, pp. 537–547.

[86] PRISACARI, B., RODRÍGUEZ, G., MINKENBERG, C., AND HOEFLER, T. Bandwidth-optimal all-to-all exchanges in fat tree networks. In *International Conference on Supercomputing, ICS'13, Eugene, OR, USA - June 10 - 14, 2013* (2013), ACM, pp. 139–148.

[87] PRISACARI, B., RODRÍGUEZ, G., MINKENBERG, C., AND HOEFLER, T. Fast pattern-specific routing for fat tree networks. *ACM Transactions on Architecture and Code Optimization (TACO) 10*, 4 (2013), 1–25.

[88] RAAB, M., AND STEGER, A. Balls into bins - a simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science*. Springer, 1998, pp. 159–170.

[89] RAECKE, H. Minimizing congestion in general networks. In *In Proceedings of the* 43^{rd} *IEEE Symposium On Foundations of Computer Science (FOCS)* (2002), pp. 43–52.

[90] RANKA, S., WANG, J.-C., AND FOX, G. C. Static and run-time algorithms for all-to-many personalized communication on permutation networks. *IEEE Trans. Parallel Distrib. Syst. 5*, 12 (Dec. 1994), 1266–1274.

[91] REQUENA, C. G., VILLAMÓN, F. G., GÓMEZ, M. E., LÓPEZ, P., AND DUATO, J. Deterministic versus adaptive routing in fat-trees. *Proc. of the* 21^{st} *Parallel and Distributed Processing Symposium, 2007* (Mar. 2007), 1–8.

[92] RODRIGUEZ, G., BADIA, R. M., AND LABARTA, J. An evaluation of marenostrum performance. vol. 22, Sage Publications, Inc., pp. 81–96.

[93] RODRIGUEZ, G., BEIVIDE, R., MINKENBERG, C., LABARTA, J., AND VALERO, M. Exploring pattern-aware routing in generalized fat tree networks. In *ICS '09: Proceedings of the 23rd international conference on Supercomputing* (New York, NY, USA, 2009), ACM, pp. 276–285.

[94] RODRIGUEZ, G., MINKENBERG, C., BEIVIDE, R., LUIJTEN, R. P., LABARTA, J., AND VALERO, M. Oblivious routing schemes in extended generalized fat tree networks. In *CLUSTER* (2009), IEEE, pp. 1–8.

[95] SACK, P., AND GROPP, W. Faster topology-aware collective algorithms through non-minimal communication. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2012), PPoPP '12, ACM, pp. 45–54.

[96] SCHROEDER, M., BIRRELL, A., BURROWS, M., MURRAY, H., NEEDHAM, R., RODEHEFFER, T., SATTERTHWAITE, E., AND THACKER, C. Autonet: a high-speed, self-configuring local area network using point-to-point links. *Selected Areas in Communications, IEEE Journal on 9*, 8 (oct 1991), 1318 –1335.

[97] SCOTT, D. Efficient all-to-all communication patterns in hypercube and mesh topologies. In *Distributed Memory Computing Conference, 1991. Proceedings., The Sixth* (apr-1 may 1991), pp. 398 –403.

[98] SINGH, A. *Load-balanced routing in interconnection networks*. PhD thesis, Stanford University, 2005.

[99] SISTARE, S., VANDEVAART, R., AND LOH, E. Optimization of MPI collectives on clusters of large-scale SMP's. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)* (New York, NY, USA, 1999), Supercomputing '99, ACM.

[100] SRINIVASAN, A. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on* (1997), pp. 416–425.

[101] SUR, S., JIN, H.-W., AND PANDA, D. K. Efficient and scalable all-to-all personalized exchange for infiniband-based clusters. In *Parallel Processing, 2004. ICPP 2004. International Conference on* (2004), pp. 275–282 vol.1.

[102] (TACC), T. A. C. C. Press release: "stampede's" comprehensive capabilities to bolster u. s. open science computational resources.

[103] THAKUR, R., RABENSEIFNER, R., AND GROPP, W. Optimization of collective communication operations in MPICH. *IJHPCA 19*, 1 (2005), 49–66.

[104] VALERIO, M., MOSER, L., AND MELLIAR-SMITH, P. Using fat-trees to maximize the number of processors in a massively parallel computer. In *Proceedings of the 1993 International Conference on Parallel and Distributed Systems* (1993), pp. 128–134.

[105] VALERIO, M., MOSER, L., AND MELLIAR-SMITH, P. Recursively scalable fat-trees as interconnection networks. In *Phoenix Conference on Computers and Communications* (1994), vol. 13, pp. 40–46.

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

[107] VALIANT, L. G. A bridging model for parallel computation. *Commun. ACM 33*, 8 (Aug. 1990), 103–111.

[108] WILLCOCK, J. J., HOEFLER, T., EDMONDS, N. G., AND LUMSDAINE, A. Active pebbles: parallel programming for data-driven applications. In *Proceedings of the international conference on Supercomputing* (New York, NY, USA, 2011), ICS '11, ACM, pp. 235–244.

[109] WILLIAMS, S., CARTER, J., OLIKER, L., SHALF, J., AND YELICK, K. Lattice Boltzmann simulation optimization on leading multicore platforms. In *Proc. of the International Conference on Parallel and Distributed Computing Systems (IPDPS)* (2008).

[110] XIE, M., LU, Y., LIU, L., CAO, H., AND YANG, X. Implementation and evaluation of network interface and message passing services for TianHe-1A supercomputer. In *Proceedings of the 2011 IEEE 19th Annual Symposium on High Performance Interconnects* (Washington, DC, USA, 2011), HOTI '11, IEEE Computer Society, pp. 78–86.

[111] YANG, Y., AND WANG, J. Optimal all-to-all personalized exchange in multistage networks. In *Parallel and Distributed Systems, 2000. Proceedings. Seventh International Conference on* (2000), pp. 229 –236.

[112] YU, W., PANDA, D. K., AND BUNTINAS, D. Scalable, high-performance nic-based all-to-all broadcast over Myrinet/GM. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing* (Washington, DC, USA, 2004), CLUSTER '04, IEEE Computer Society, pp. 125–134.

[113] ZAHAVI, E. Fat-trees routing and node ordering providing contention free traffic for MPI global collectives. In *IPDPS Workshops* (2011), IEEE, pp. 761–770.

[114] ZAHAVI, E., JOHNSON, G., KERBYSON, D. J., AND LANG, M. Optimized InfiniBand fat-tree routing for shift all-to-all communication pattern. In *International Supercomputing Conference (ISC07)* (Dresden, Germany, June 2007).

[115] ZAHAVI, E., JOHNSON, G., KERBYSON, D. J., AND LANG, M. Optimized InfiniBand(TM) fat-tree routing for shift all-to-all communication patterns. *Concurr. Comput. : Pract. Exper. 22*, 2 (Feb. 2010), 217–231.