# Avoiding hot-spots on two-level direct networks

Abhinav Bhatele
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551, USA
bhatele@llnl.gov

Nikhil Jain
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
nikhil@illinois.edu

William D. Gropp
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
wgropp@illinois.edu

Laxmikant V. Kale
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
kale@illinois.edu

## ABSTRACT

A low-diameter, fast interconnection network is going to be a pre-requisite for building exascale machines. A two-level direct network has been proposed by several groups as a scalable design for future machines. IBM's PERCS topology and the dragonfly network discussed in the DARPA exascale hardware study are examples of this design. The presence of multiple levels in this design leads to hot-spots on a few links when processes are grouped together at the lowest level to minimize total communication volume. This is especially true for communication graphs with a small number of neighbors per task. Routing and mapping choices can impact the communication performance of parallel applications running on a machine with a two-level direct topology. This paper explores intelligent topology aware mappings of different communication patterns to the physical topology to identify cases that minimize link utilization. We also analyze the trade-offs between using direct and indirect routing with different mappings. We use simulations to study communication and overall performance of applications since there are no installations of two-level direct networks yet. This study raises interesting issues regarding the choice of job scheduling, routing and mapping for future machines.

## Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—Network topology; C.4 [**Performance of Systems — Design studies**]; C.4 [**Performance of Systems — Modeling techniques**]; I.6 [**Simulation and Modeling**]

## General Terms

Algorithms, Performance

## Keywords

mapping, communication, performance, dragonfly network, exascale


*SC* '11, November 12-18, 2011, Seattle, Washington USA


## 1. INTRODUCTION

Parallel machines of the future will have a large number of fast cores on each node and a low network bytes-to-flop ratio. Computation will be cheap and communication will become expensive. Hence, scalable, low-diameter and fast networks are going to be a pre-requisite for building multi-Petaflop/s and Exaflop/s capability machines. New designs have been proposed recently by IBM (the PERCS topology [2]) and by the DARPA sponsored *Exascale Computing Study* on technical challenges in hardware (the dragonfly topology [15]). Both these topologies are multi-level direct networks with all-to-all connections at each level. Nodes are grouped logically to form cliques and cliques are grouped to form larger clusters. These logical groups at different levels of the hierarchy are named drawers and supernodes in the PERCS topology and groups and racks (or cabinets) in the dragonfly topology.

The main idea behind these topologies is to provide high bandwidth links within the groups at the lowest level of the hierarchy and then connect those groups with another level of links. Multiple levels also allow the use of high bandwidth copper links within the groups and optical links, which can be longer, for links spanning across groups. The hierarchical nature of this design poses a problem for applications where grouping processes together is optimal for restricting communication to the lower levels of the hierarchy – grouping at lower levels leads to hot-spots on some links connecting the higher levels, leading to performance degradation.

It has been suggested that non-minimal adaptive routing or indirect routing can solve this problem of overloading of a few links. Indirect routing leads to the use of more hops/links for each message, as compared to direct routing, leading to lower available bandwidth overall. Also, indirect routing adds another layer of implementation complexity for large networks. In this paper, we compare the impact of direct and indirect routing on network congestion and application performance.

We compare intelligent topology aware mappings with the default MPI rank-ordered mapping with respect to network utilization and application performance. We explore mapping of different communication patterns on to the network topology to identify scenarios which minimize hot-spots. We use three diverse communication patterns, representative of widely used parallel applications - a two-dimensional (2D) five-point stencil (Weather Research and Forecast Model [17]), a four-dimensional nine-point stencil (MIMD Lattice Computation [3]) and an $n$-targets multicast pattern (representative of NAMD [6]).

We use a simulation framework called BigSim [21] to study and predict the behavior of parallel applications on future architectures/

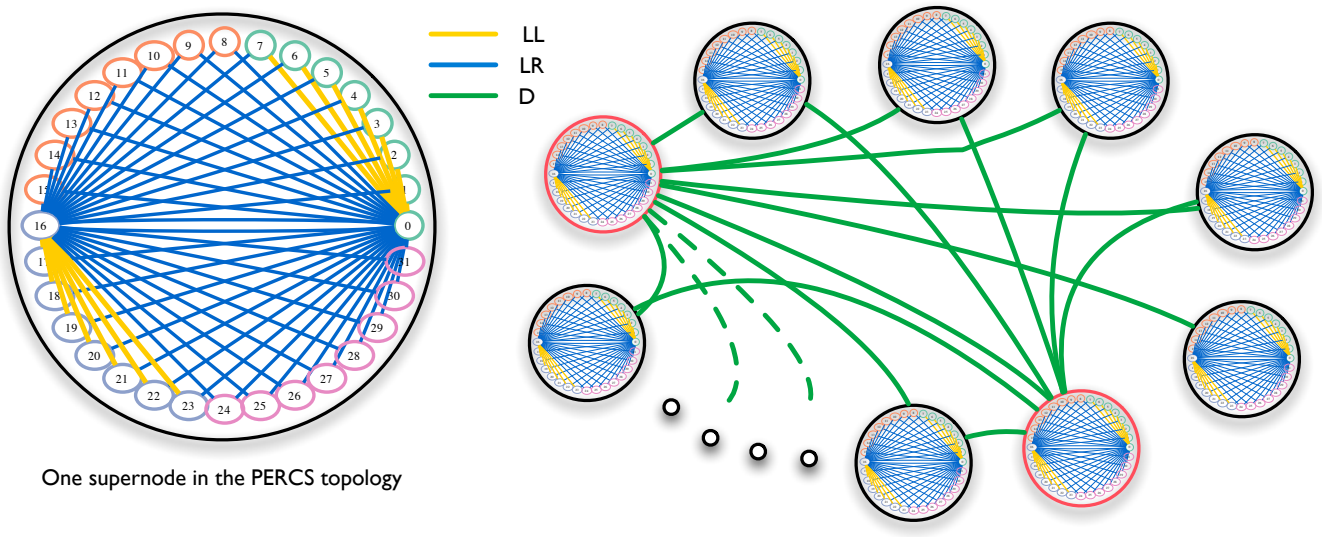


**Figure 1: The PERCS network – the left figure shows all to all connections within a supernode (connections originating from only two nodes, 0 and 16, are shown to keep the diagram simple). The right figure shows second-level all to all connections across supernodes (again D links originating from only two supernodes, colored in red, are shown).**

topologies. Using traces collected by our emulation-based technique, we simulate application runs on hundreds of thousands of cores. Non-uniform link bandwidths on different classes of links complicate the issue of identifying the weakest links. Interesting issues arise because of the imbalance in number of different types of links available when using a small subset of the entire topology. Hence, we do simulations for one quarter of the full system size (assuming 300 supernodes) and the full system as well.

The novel contributions of this paper are:

- To the best of our knowledge, this paper has the first analysis of congestion on a two-level direct topology due to routing and mapping choices. We present several solutions for avoiding hot-spots on such networks.
- The paper presents the largest packet-level detailed network simulations done so far (for 307,200 cores) for several communication patterns. These simulations help us analyze application performance in great detail through performance counter-based per-level link statistics, visualization tools and predicted application performance.
- We present several intelligent mappings for 2D, 4D and multicast patterns and compare their performance when coupled with direct and indirect routing on the PERCS network.

## 2. THE PERCS TOPOLOGY

The PERCS interconnect topology is a fully connected two-tier network [2]. Figure 1 (left) shows one supernode of the PERCS topology as a large circle. Within the large circle, a small circle represents a quad chip module (QCM) which consists of four 8-core Power7 chips. We will refer to a QCM as a node in rest of the paper. Eight nodes in one color in each quadrant constitute a drawer. Each node has a hub/switch which has three types of links originating from it - LL, LR and D links. There are seven LL links (24 GB/s) that connect a node to seven other nodes in the same drawer. In addition, there are 24 LR links (5 GB/s) that connect a node to the remaining 24 nodes of the supernode. LL and LR links constitute the first tier connections that enable communication between any two nodes in one hop. To maintain simplicity, LL and LR links originating from only two nodes, numbered 0 and 16 are shown in Figure 1 (left).

On the right, in Figure 1, the second tier connections between supernodes are shown. Every supernode is connected to every other supernode by a D link (10 GB/s). These inter-supernode connections originate and terminate at hub/switches connected to nodes; a given hub/switch is directly connected to only a fraction ($\leq 16$) of the 512 supernodes (full system size). For simplicity, D links originating from only two supernodes (in red) have been shown. 32 cores of a node can inject on to the network at a rate of 192 GB/s through a hub/switch directly connected to them.

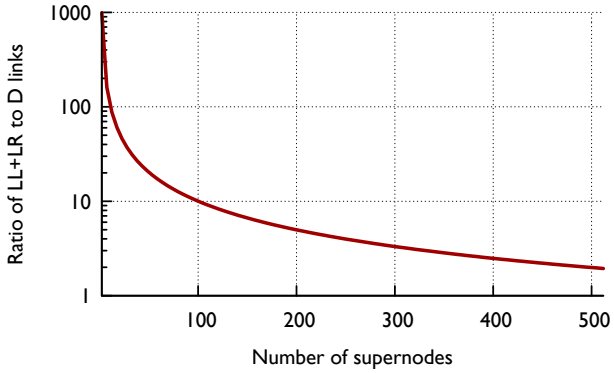


**Figure 2: The number of D links reduces significantly compared to that of LL and LR links as one uses fewer and fewer supernodes in the PERCS topology.**

An important thing to note about the PERCS topology is the ratio of the number of first level connections to that of second level connections. For a system with $n$ supernodes, the number of D links is $(n \times (n-1))$. There are $(32 \times 31 \times n)$ LL and LR links in total. Hence, there are $(992/(n-1))$ first tiers links for every second tier link as shown in Figure 2. One can observe that as the number of supernodes used by an application gets smaller, there is

a scarcity of D links in comparison to LL and LR links. This may be a bottleneck in applications with comparable intra-supernode and inter-supernode traffic if they are running on a small subset of supernodes. Hence, we simulate two different system sizes (64 supernodes and 300 supernodes) to compare them.

The next section will present a case study of a 2D Stencil showing that a default mapping of this application with direct routing can lead to significant congestion on the network. Hence, interesting research questions arise with respect to reducing hot-spots on two-level direct networks. Random versus contiguous job scheduling, direct versus indirect routing and intelligent mapping techniques present opportunities to minimize congestion.

# 3. MOTIVATION

Let us look at a relatively simple yet prevalent communication pattern — a two-dimensional five-point stencil computation. We will consider a case where the application uses 16 supernodes or $16,384$ cores of the machine. Placement at various levels can play an important role for this pattern in deciding which MPI processes to put together on one node (32 cores) on one hand to which nodes or drawers should be placed on "virtual" supernode boundaries on the other hand. Let us assume that the virtual cartesian topology for this example is $128 \times 128$ and the communication is wrapped around on all four sides.

The default placement of processes by the job scheduler will divide the 2D topology of $128 \times 128$ tasks along one dimension and each supernode gets a block of $8 \times 128$ tasks. Within a supernode, each drawer gets a block of $2 \times 128$ tasks in order and each node gets a block of $1 \times 32$ tasks. Figure 3 (left) shows the default placement of the $16,384$ tasks on the 16 supernodes. As is obvious from the diagram, 128 cores at the boundaries share a D link (shown in red) with a capacity of 10 GB/s leading to an effective bandwidth of $10 \div 128$ GB/s.

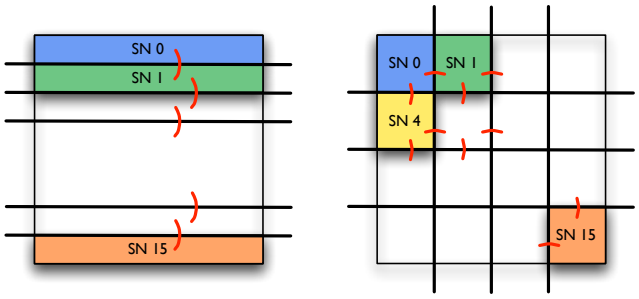


**Figure 3: Default (linear) and blocked (square) mapping of tasks performing 2D communication on 16 supernodes**

A simple square decomposition of the domain at each level (supernodes, drawers and nodes) can improve the utilization of links significantly. Assigning a square block of $32 \times 32$ tasks to each supernode leads to the use of more D links (see Figure 3, right) – 32 instead of 16 D links are used now. Also, each D link is used for only one-fourth of the data in this setup and hence the effective bandwidth per D link is $10 \div 32$ GB/s.

Next, let us consider the grouping of MPI processes within a drawer and a node, for optimizing communication further using topology aware mapping (Figure 4). As opposed to the default mapping, within a supernode, tasks can be grouped into blocks of $16 \times 16$ to be placed on the four drawers and further into blocks of $4 \times 8$ to be placed on each node. In this case, the effective bandwidth available on LR links is $5 \div 8$ GB/s and that on each LL link

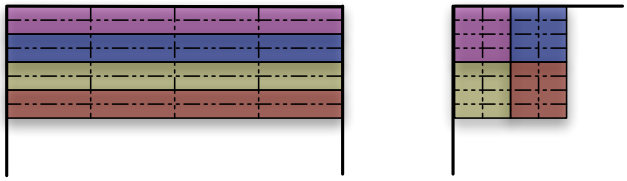

**Figure 4: Default (linear) and blocked (square) mapping of tasks on to drawers and nodes within a supernode**

is $24 \div 8$ GB/s. These improvements in link utilization using an intelligent mapping are summarized in Table 1.

| Link | Default Mapping | Good Mapping |
|---|---|---|
| D | $10/128 = 0.078$ | $10/32 = 0.313$ |
| LR | $5/32 = 0.156$ | $5/8 = 0.625$ |
| LL | $24/32 = 0.75$ | $24/8 = 3.0$ |

**Table 1: Effective link bandwidth (in GB/s) when using the default and an intelligent mapping for a 2D Stencil of 16K tasks**

As we can see, an intelligent mapping can increase the number of links used and reduce the load on each link. This can lead to significant performance improvements. However, it is important to note that even a good blocked mapping utilizes only 32 D links whereas the number of D links among 16 supernodes is $16 \times 15 = 240$. A random mapping at the level of nodes or drawers will increase the number of D links used although it might lead to hot-spots. We will look at these issues in detail in Section 6.

# 4. APPROACHES TO MINIMIZING CONGESTION ON THE NETWORK

Topology aware mapping of MPI tasks to physical cores/nodes on a machine can minimize contention and impact application performance [4, 9, 19, 20]. Intelligent mapping can be used to carefully distribute traffic over the various links on two-level direct networks. This section outlines the different mappings that we evaluate for minimizing hot-spots on the network. We also explore indirect routing coupled with some of the mappings to analyze if it can be used as an alternative to intelligent mapping.

## 4.1 Topology aware mapping

A default MPI rank-ordered mapping of processes on to nodes of a two-level direct network can lead to significant hot-spots and extremely low effective bandwidth on some links (as shown in Section 3). Intelligent mapping of the virtual communication topology on to such networks can spread the communication over more links instead of concentrating it over a few and result in reduced contention and better application performance. Below, we present different mapping techniques for near-neighbor communication patterns that will be compared using simulations:

**Default Mapping (DEF)**: Default mapping refers to a "contiguous" MPI rank-ordered mapping where rank 0 is placed on the first core in the allocated job partition, rank 1 on the second and so on. Let us use a concrete example of mapping a 2D near-neighbor communication pattern originating from a 5-point stencil on to 64 supernodes of the PERCS topology to understand different mappings. We assume a virtual topology of $256 \times 256$ tasks for this example since there are $65,536$ cores on 64 supernodes. The default mapping by the job scheduler will place the first 1024 tasks *i*.e. a block

| Communication Pattern | Number of Supernodes | Number of Elements | Number of Messages | Message Size (KB) | Sequential Computation (ms) |
|---|---|---|---|---|---|
| 2D 5-point Stencil | 64 | $8192 \times 8192$ | 4 | 64 | 479 |
| 4D 9-point Stencil | 64 | $64 \times 64 \times 64 \times 64$ | 8 | 2048 | 224 |
| Multicast Pattern | 64 | – | 14 | 1024 | – |
| 4D 9-point Stencil | 300 | $64 \times 32 \times 64 \times 32$ | 8 | 1024 | 50 |

**Table 2: Details of the experimental setup for different communication patterns and different number of supernodes**

of $4 \times 256$ tasks on the first supernode, the next block of $4 \times 256$ tasks on the second supernode and so on. Within each supernode, the first 256 tasks *i*.e. a block of $1 \times 256$ will be placed on the first drawer and so on. Each node in a drawer will get a block of $1 \times 32$ tasks. As we have seen in Section 3, this can be very inefficient.

**Blocked Nodes Mapping (BNM)**: The first obvious thing to try with respect to near-neighbor communication is to cluster communicating processes on a node, which has 32 cores, to minimize inter-node traffic and hence reduce the communication volume being sent on the network links. Hence, for all intelligent mappings in the paper, we place $4 \times 8$ tasks in 2D and $4 \times 2 \times 2 \times 2$ in 4D on a node.

**Blocked Drawers Mapping (BDM)**: Next, we can attempt blocking processes at the level of drawers. For the 2D example, instead of the default $1 \times 256$ processes being placed on a drawer, we try different blocks of $4 \times 64$, $8 \times 32$ and $16 \times 16$. The smallest possible dimension size of the drawer blocks is decided by the smaller dimension of the node blocks which is 4. Hence we cannot map a block of $2 \times 128$ tasks, for example, on to the drawer.

**Blocked Supernodes Mapping (BSM)**: In this particular mapping, in addition to blocking for nodes and drawers, we also block MPI tasks for supernodes. Similar to the block size restriction on drawers in BDM, the smallest possible dimension size for the supernode block is decided by the smallest dimension of the drawer block for a particular mapping. For example, in the 2D case, for a drawer block of $16 \times 16$, we cannot map a block of $8 \times 128$ or $4 \times 256$ on the supernode. The only possible choices are $16 \times 64$ and $32 \times 32$.

**Random Nodes Mapping (RNM)**: The goal with intelligent mapping is to increase the number of links used in this two-level direct network and reduce the volume of data sent on each link. Mapping MPI processes randomly on to the supernodes (still maintaining a block of $4 \times 8$ on each node to minimize inter-node communication volume) is one possible solution to achieve this. Randomizing the placement of MPI tasks at the level of nodes increases the number of D links used for sending data across supernodes and in turn, reduces the amount of data being sent on each link. Also, this mapping requires no programming effort from the application end-user.

**Random Drawers Mapping (RDM)**: We also try random mapping at the level of drawers where each drawer gets a block of $16 \times 16$ tasks (for the 2D case, for example). The intuition for this mapping is to restrict most of the communication within a drawer where we can use the high-bandwidth LL links to the fullest. This would minimize data being sent over the lower bandwidth LR and D links.

## 4.2 Direct versus indirect routing

When using direct routing on the PERCS topology, each message travels 1 to 3 hops on the network. The 3 hop routes consist of an LL or LR link as the 1st and 3rd hop and a D link as the middle hop. Indirect routing aims to reduce the load on the D links by jumping to a random supernode first and then going to the destination supernode through that supernode. This avoids using the direct D link between a pair of supernodes for sending all the traffic between them, which can lead to congestion. The maximum number of hops in an indirect route is 5 (LL/LR – D – LL/LR – D – LL/LR). We simulated two of the mappings above with indirect routing:

**Default Mapping with Indirect Routing (DFI)**: This is representative of the case where an application user lets the job scheduler map MPI ranks on to the supernodes contiguously, but the routing is indirect to minimize congestion on the network.

**Random Drawers Mapping with Indirect Routing (RDI)**: This approach is representative of the scenario where the user maps to random drawers or the job scheduler assigns random drawers from all supernodes on the machine to a particular job and an indirect routing is used on the system.

# 5. SIMULATION METHODOLOGY

In this section, we describe the experimental methodology and setup that have been used to predict the link utilization and performance on the PERCS topology. We present results for three benchmarks – a 2D Stencil, a 4D stencil and a multicast pattern. We use a three step method that consists of compute time prediction, trace collection through emulation and then, simulation.

## 5.1 Problem sizes

Simulations in this paper were done for two different allocation sizes – 64 supernodes and 300 supernodes, both being a fraction of the largest possible machine size (512 supernodes) with a two-level PERCS network. The use of fewer supernodes than the full system size leads to different ratios of D to LL and LR links and hence, interesting contention issues (see Section 2).

We use MPI codes run as AMPI programs [12] in all our experiments. The details of the experiments are summarized in Table 2. For the 2D Stencil, a block decomposition of the data array to the MPI tasks is done wherein each MPI task holds a 2D array of $8192 \times 8192$ elements. In each iteration, each MPI task sends and receives four messages of 64 KB each to/from its four neighbors and updates its elements using a 5-point stencil computation. In the 4D Stencil on 64 supernodes, the array on each MPI task has $64 \times 64 \times 64 \times 64$ elements with MPI tasks also arranged in a 4-dimensional virtual topology. Every MPI task exchanges 2 MB of data with each of its eight neighbors and updates its elements using a 9-point stencil in every iteration. For simulating 4D Stencil on 300 supernodes, we used an array of $64 \times 32 \times 64 \times 32$ elements per MPI task. For the multicast pattern, every MPI task sends 1 MB messages to fourteen other tasks. The tasks are supposed to represent the recipients of a multicast tree and are chosen accordingly for each root.

## 5.2 Prediction methodology

Predicting performance of an application for a future machine using BigSim [21] involves emulation on an existing architecture

to obtain traces for the future machine and then simulation using those traces. If the computational characteristics of the emulating architecture and the target architecture are different, then a "compute time" replacement step is also involved. The three step process for performance prediction is outlined below:

**Compute Time Prediction:** We first obtain the computation time for the various kernels (2D and 4D Stencil) on Blue Drop, an IBM Power7 processor. Blue Drop, located at NCSA, has four 8-core chips running at 3.8 GHz. We ran a multi threaded version (4-way SMT) of the kernels and obtained execution times as shown in Table 2. These are used to replace the actual computation times during trace collection using BgAdvance, a function call that helps in incrementing the runtime clock by user defined values.

**Trace Collection:** BigSim [21], a part of the Charm++ distribution [13], is used for emulating the PERCS network with $307,200$ cores. We use Blue Print, an IBM Power5 cluster (also at NCSA), to generate the traces. Each node of Blue Print has 16 cores and 64 GB of memory running at 1.9 GHz. We were able to emulate 2D Stencil, 4D Stencil and the multicast pattern for a 64 supernode target system on 32 nodes of Blue Print in approximately two minutes. Emulations of 4D Stencil for the 300 supernode target system were done on 85 nodes of Blue Print in approximately ten minutes.

**Simulation:** We use the simulation component of BigSim with a detailed PERCS network model [2] to simulate the applications using the traces collected from the previous step. These simulation runs were done on Ember, which is a SGI Altix UV 1000 distributed shared memory system at NCSA.

# 6. SIMULATIONS FOR 64 SUPERNODES

We now present simulation results for the three communication patterns described in detail below and also summarized in Table 2. All simulations in this section were done for 64 supernodes.

## 6.1 Mapping a 2D 5-point Stencil

A two-dimensional five-point stencil is a prevalent communication pattern in parallel applications in various domains. The two-level direct topology, by nature of its inherent design, can suffer from hot-spots for a default MPI rank-ordered mapping of a communication pattern as simple as near-neighbor 2D. We choose this as our first communication pattern, the mapping for which is easier to illustrate and the results easier to analyze.

| Mapping | Node | Drawer | Supernode |
|---|---|---|---|
| DEF | $32 \times 1$ | $256 \times 1$ | $256 \times 4$ |
| BNM | $8 \times 4$ | $64 \times 4$ | $256 \times 4$ |
| BDM | $8 \times 4$ | $16 \times 16$ | $64 \times 16$ |
| BSM | $8 \times 4$ | $16 \times 16$ | $32 \times 32$ |

**Table 3: Dimensions of blocks at different levels (node, drawer and supernode) for different mappings of 2D Stencil**

The data array for this 2D Stencil is $2097152 \times 2097152$ and each MPI task is given a sub-domain of $8192 \times 8192$ elements. This gives us a logical 2D array of MPI tasks of dimensions $256 \times 256$ which is to be mapped to $65,536$ cores (64 supernodes). We deploy the six mappings with direct routing and two others with indirect routing as explained in Section 4 to map a 2D Stencil pattern on to 64 supernodes. The aspect ratios of the blocks were chosen in the same fashion as we did for 16 supernodes in Section 3 and the choices for the various mappings are summarized in Table 3. For

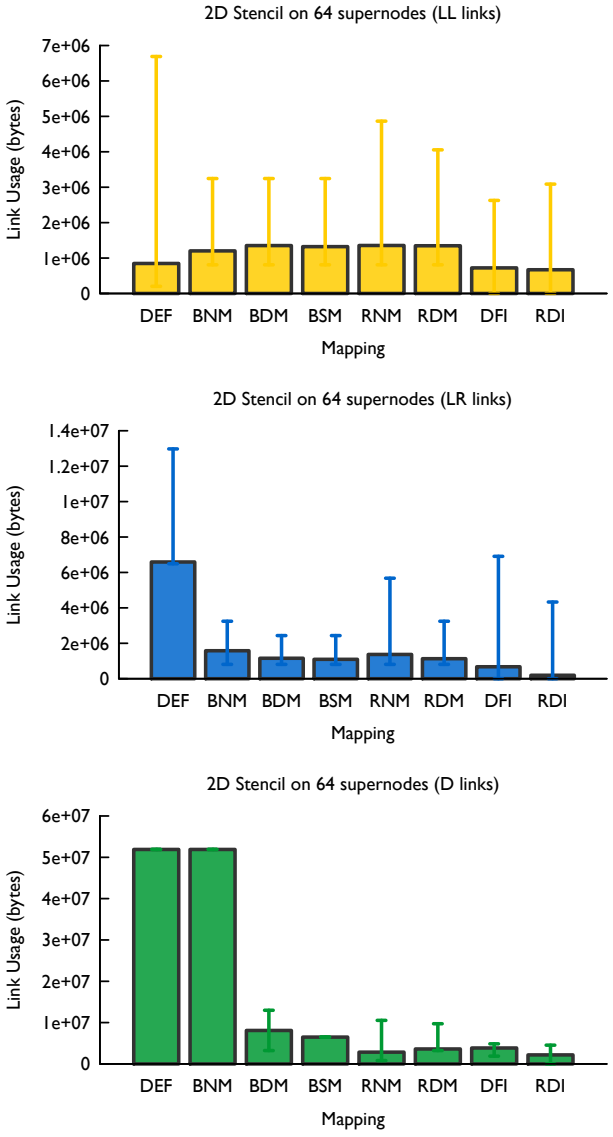


**Figure 5: Average number of bytes sent over LL, LR and D links for 2D Stencil on 64 supernodes**

the random nodes mapping, a block of $4 \times 8$ tasks is placed on each node and for the random drawers mapping, a block of $16 \times 16$ tasks is placed on each drawer.

Independent simulations were done for each mapping and we obtained the execution times and link statistics (number of packets sent) on different network links. Figure 5 shows the average, minimum and maximum number of bytes passing through each type of link for different mappings. As we increasingly block for nodes, drawers and supernodes, the average bytes sent over the D links decreases, which correlates well with decrease in execution time (Table 4). Blocking leads to an increase in the usage of LL links which is favorable since they have high bandwidth. Simulations that use indirect routing lower the D link usage even further and a very small fraction of the total time is spent in communication (given that the computation time is 479 ms).

The overall improvements in execution time for 2D Stencil are not significant because the message size is small (64 KB) and hence there is negligible load on the high bandwidth links. We shall see

| DEF | BNM | BDM | BSM | RNM | RDM | DFI | RDI |
|---|---|---|---|---|---|---|---|
| 481.70 | 481.74 | 480.07 | 480.90 | 480.71 | 481.03 | 480.07 | 479.74 |

**Table 4: Execution time per iteration (in ms) for 2D Stencil for different mappings on 64 supernodes**

| Mapping | Node | Drawer | Supernode |
|---|---|---|---|
| DEF | $16 \times 2 \times 1 \times 1$ | $16 \times 16 \times 1 \times 1$ | $16 \times 16 \times 4 \times 1$ |
| BNM | $4 \times 2 \times 2 \times 2$ | $16 \times 4 \times 2 \times 2$ | $16 \times 16 \times 2 \times 2$ |
| BDM | $4 \times 2 \times 2 \times 2$ | $4 \times 4 \times 4 \times 4$ | $16 \times 4 \times 4 \times 4$ |
| BSM | $4 \times 2 \times 2 \times 2$ | $4 \times 4 \times 4 \times 4$ | $8 \times 8 \times 4 \times 4$ |

**Table 5: Dimensions of blocks at different levels (node, drawer and supernode) for different mappings of 4D Stencil**

that mapping can result in significant improvements when communication is higher, in the next few sections.

## 6.2 Mapping a 4D 9-point Stencil

A four-dimensional nine-point stencil is representative of the communication pattern in MILC, a Lattice QCD code. For the same amount of data assigned to each task in a two- and four-dimensional stencil computation, say $x^4$, the computation is $5x^4$ in 2D versus $9x^4$ in 4D and the size of each message is $x^2$ in 2D and $x^3$ in 4D. Hence, we expect more congestion, given larger messages and better improvement from mapping for 4D Stencil.

For 4D Stencil simulations, we consider an array of $1024 \times 1024 \times 1024 \times 1024$ doubles. The 4D array is distributed among MPI tasks by recursively dividing along all four dimensions, with each task being assigned $64 \times 64 \times 64 \times 64$ elements. This leads to a logical 4D grid of MPI tasks of dimensions $16 \times 16 \times 16 \times 16$. In each iteration, every MPI task sends eight messages of size $64 \times 64 \times 64$ elements to its eight neighbors. Table 5 lists the dimensions of the blocks of tasks placed on a node, drawer and supernode for different mappings. For the random nodes mapping, we place $4 \times 2 \times 2 \times 2$ tasks on a node and for the random drawers mapping, we place $4 \times 4 \times 4 \times 4$ tasks on a drawer.

Figure 6 shows histograms based on the amount of data (in bytes) sent over the LL, LR and D links (note, that the bin sizes and y-axis ranges for the LL, LR and D link plots are different). The counts only include links with non-zero utilization. The amount of data being sent over D links is much higher (bin size of 108.3 MB) and hence, we expect that lowering the amount of data sent on D links will have a positive impact on the performance. Let us focus on the right column first which shows the D link usage for different mappings. For the default mapping, a large number of links are in the last bin *i.e.* they are heavily utilized. As we progressively block tasks using different mappings (BNM, BDM and BSM), number of links in the lower numbered bins increases, signifying fewer bytes passing through each link and fewer hotspots. Random nodes mapping (RNM) is successful in spreading the load evenly on more D links and also in reducing the maximum number of bytes passing through any given link. Even though the random nodes and random drawer mappings increase the usage of LL and LR links, since the data being sent over them is small, this does not have an adverse affect on performance.

Figure 7 presents similar histograms for indirect routing coupled with default mapping and random drawers mapping. These present the best scenarios for link usage – for the D link histograms, more D links are used but the amount of data being sent over each link reduces further compared to direct routing (Figure 6). The random drawers mapping with indirect routing (RDI) reduces the maximum LL and LR link utilization also (which the other mappings are unsuccessful at).

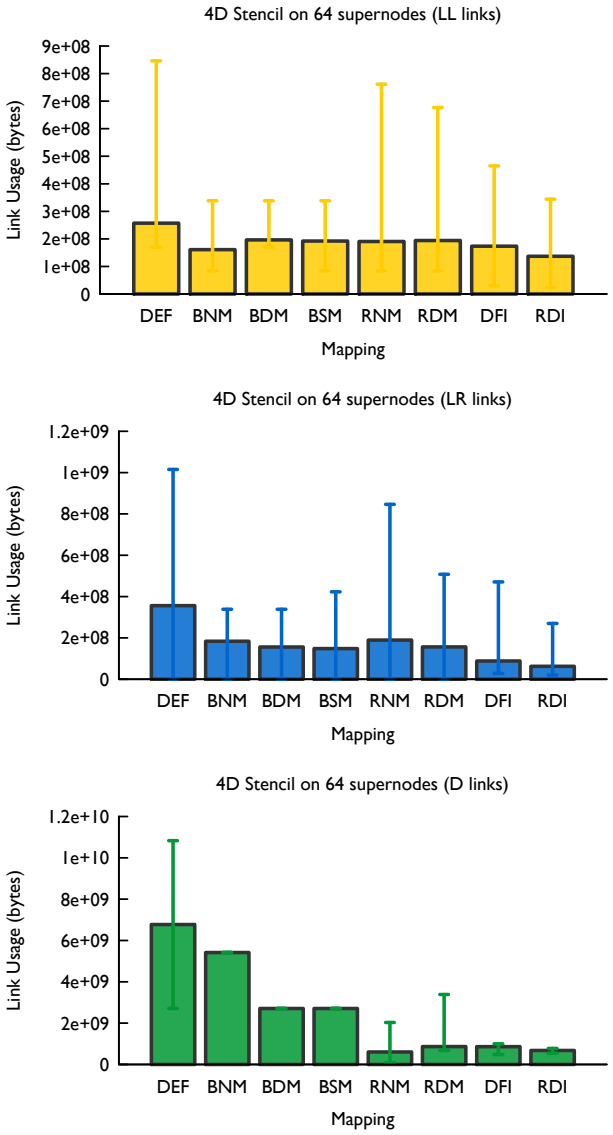


**Figure 8: Average number of bytes sent over LL, LR and D links for 4D Stencil on 64 supernodes**

The information presented in these histograms is summarized as average, minimum and maximum number of bytes sent over LL,

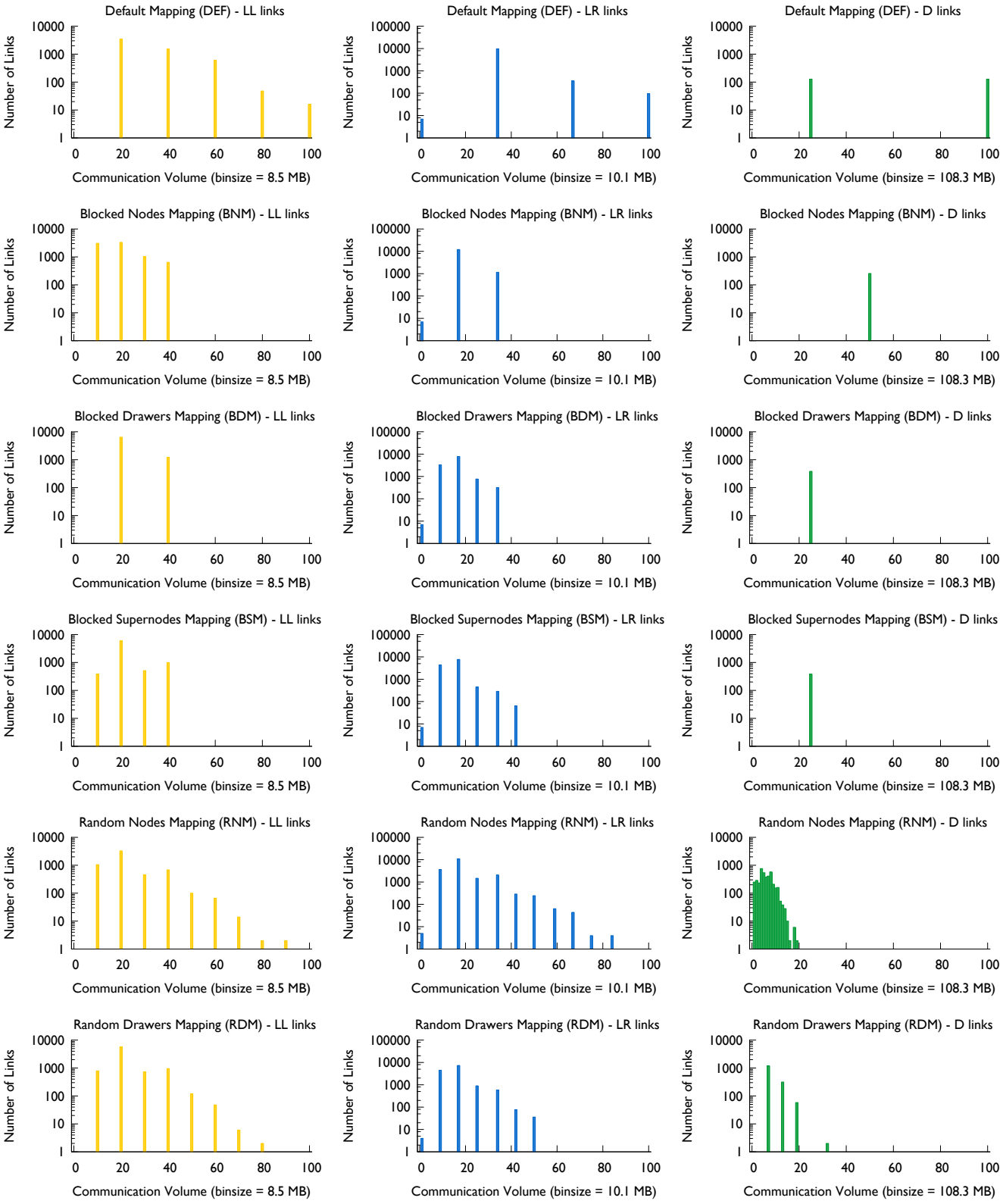


**Figure 6: Histograms showing link utilization in terms of number of bytes sent over different links for different mappings of a 4D Stencil on to 64 supernodes of the PERCS Topology. Each row represents utilization of the LL, LR and D links for a different mapping for direct routing, from top to bottom they are – default mapping, blocked nodes, blocked drawers, blocked supernodes, random nodes and random drawers.**

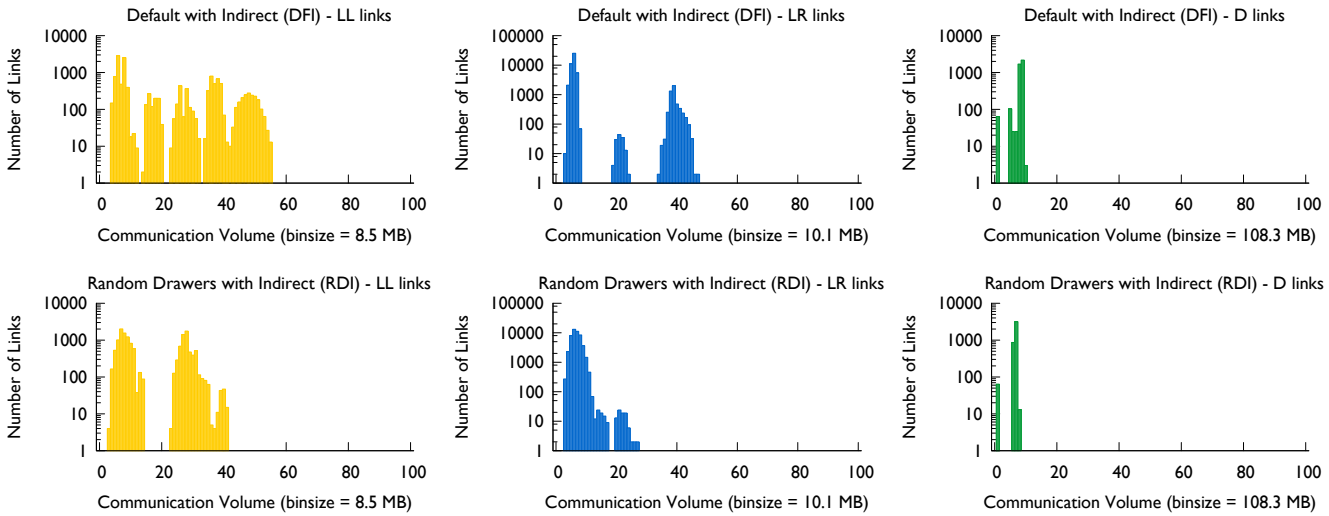


**Figure 7: Histograms showing link utilization for mapping of a 4D Stencil on to 64 supernodes of the PERCS Topology. The two rows represents utilization of LL, LR and D links for the default and random drawers mapping, respectively (both with indirect routing.)**

LR and D links for the different mappings in Figure 8. The random nodes mapping (RNM) and random drawers mapping (RDM) with direct routing and random drawers mapping with indirect routing (RDI) have the similar usage for D links and also lead to similar execution times (Figure 9). It is important to note that indirect routing achieves performance comparable to an intelligent mapping but at the cost of increasing overall traffic on the network. The black horizontal line in Figure 9 represents the lower bound for the execution time assuming that each MPI task does its sequential computation and sends its ghost messages over the lowest bandwidth LR links in a no contention scenario. The best mappings come very close to this lower bound.

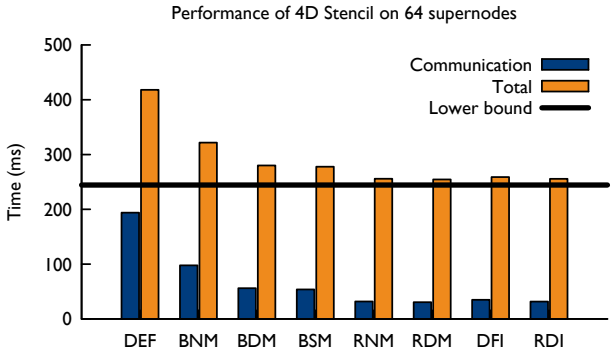


**Figure 9: Time spent in communication and overall execution per iteration for different mappings on 64 supernodes**

The BigSim simulation framework also has capabilities to output event logs which can be visualized through a performance visualization tool (Projections [14]). Figure 10 shows a histogram view of activity added across all processors for different time bins (note, the bin size on the top plot is 2 ms whereas on the bottom plot is 1 ms, so they are showing different time periods). The blue represents computation and green represents communication. In the top plot, there are gaps between computation when most processors are idle, waiting for messages before they can begin the next

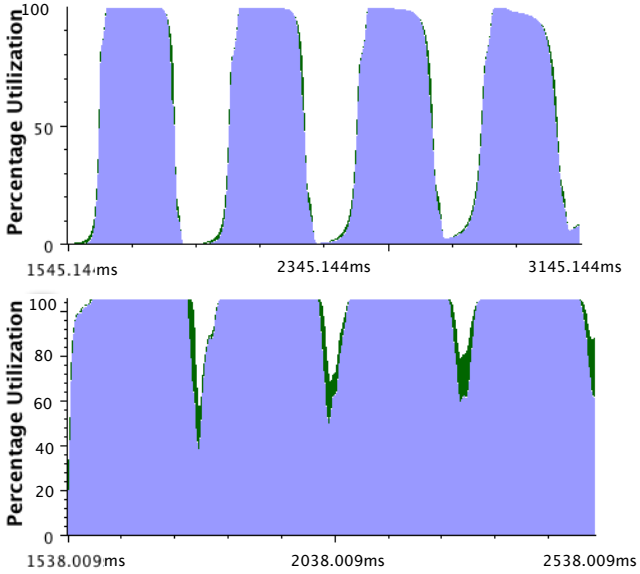


**Figure 10: Projections time profile view showing the utilization of processors over time for the DEF and RNM mapping**

iteration. In the bottom plot, by virtue of an intelligent mapping, the wait time becomes negligible and hence, most of the idle time disappears.

## 6.3 Mapping a multicast pattern

NAMD is a molecular dynamics application with a multicast communication pattern where a subset of processors build spanning trees and the root of each tree sends messages along the tree to several processors. We wrote a simple MPI benchmark to simulate this multicast pattern, where, in each iteration, every MPI task builds a spanning tree with 14 other tasks whose ranks differ from its own by $..., -2x, -x, x, 2x, 3x, ...$, where, x is a parameter. For example, for $x = 5$, MPI task with rank 50 sends messages to processors with MPI ranks 20, 25, 30, 35, 40, 45, 55, 60, 65, 70, 75, 80 and 85. This benchmark performs no computation. We compare

the default mapping of MPI tasks with four mapping and routing configurations – BNM, BDM, DFI and RDI.

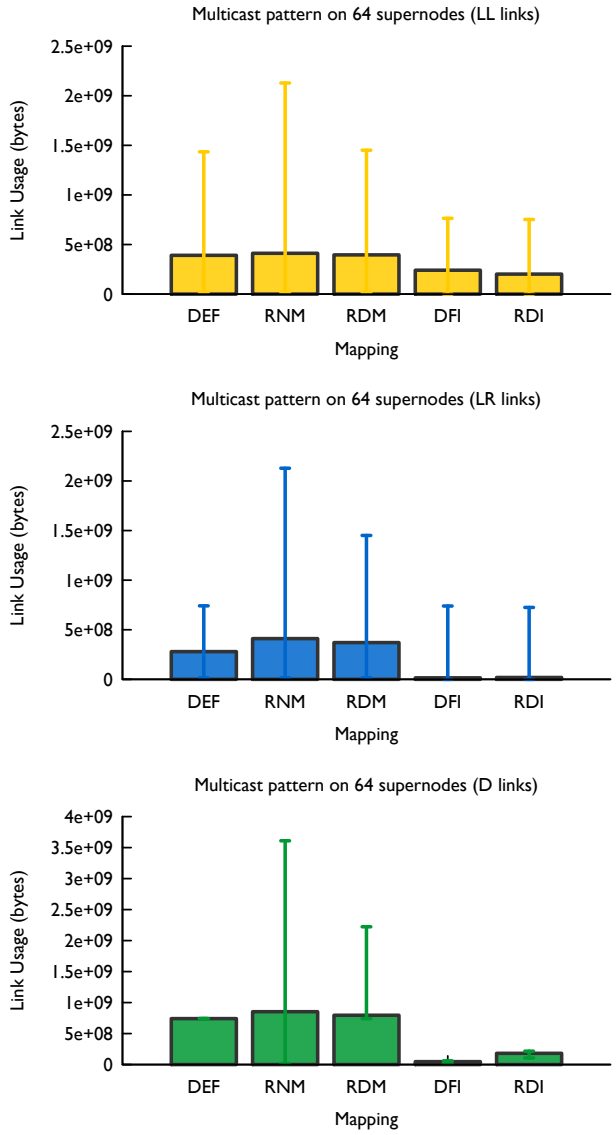


**Figure 11: Average number of bytes sent over LL, LR and D links for the multicast pattern on 64 supernodes**

In Figure 11, we present link usage statistics for the three types of links. This is a different communication pattern from the 2D and 4D near-neighbor patterns we have seen so far. The random nodes and random drawers mapping with direct routing do not get better link utilization compared to the default mapping because it is difficult to find a blocking that is optimized for this multicast pattern. However, the indirect routing cases (DFI and RDI) succeed in lowering the average and maximum usage on the D links significantly compared to the other mappings. This is also reflected in the reduction in the execution time per iteration as shown in Table 6.

## 7. SIMULATIONS FOR 300 SUPERNODES

Predictions for the sustained Petaflop/s Blue Waters machine, to be installed at Illinois, indicated that the machine would have more than 300 supernodes connected by the PERCS network (the actual number is not public). We now present results of running a 4D Stencil on 307,200 cores using a detailed packet-level PERCS network simulation. To the best of our knowledge, this is the first attempt at simulating a parallel machine at this scale.

| DEF | RNM | RDM | DFI | RDI |
|---|---|---|---|---|
| 54.64 | 87.73 | 44.24 | 17.81 | 17.64 |

**Table 6: Execution time per iteration (in ms) for the multicast pattern for different mappings on 64 supernodes**

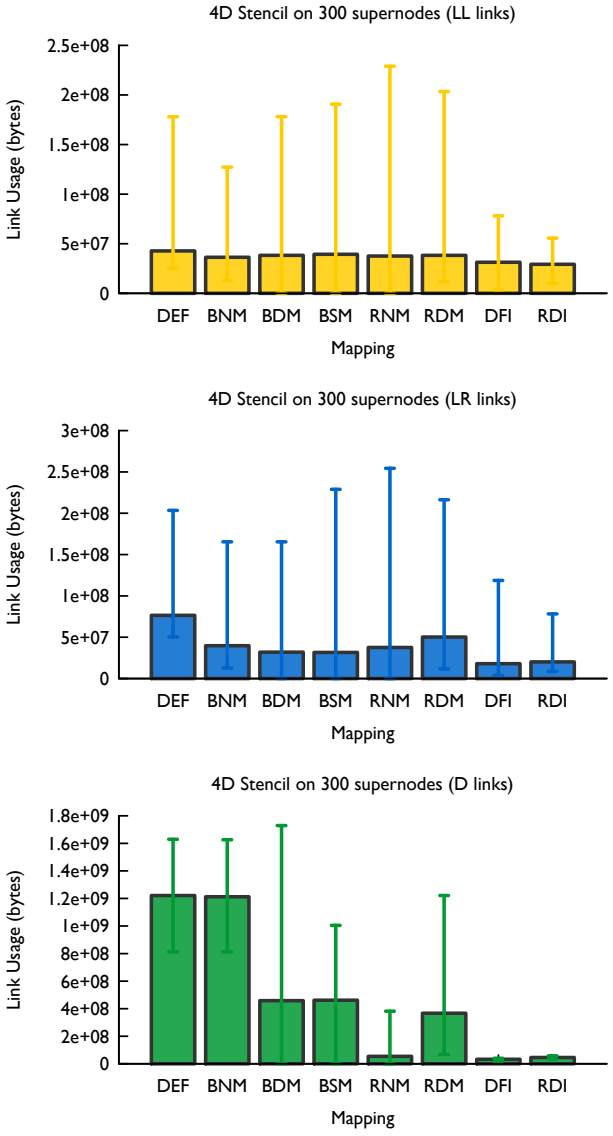


**Figure 12: Average number of bytes sent over LL, LR and D links for 4D Stencil on 300 supernodes**

For 300 supernode simulations, we consider a data array of $512 \times 512 \times 1024 \times 4800$ doubles. The 4D array is distributed among 307,200 MPI tasks with each task being assigned $32 \times 32 \times 64 \times 64$ elements. This leads to a logical 4D grid of MPI tasks of dimensions $16 \times 16 \times 16 \times 75$. We use mapping configurations similar

to those used for the 64 supernode case (see Table 5, Section 6.2). Since one dimension is a non-power-of-2 and significantly bigger than the other three, mapping on 300 supernodes is more challenging than on 64 supernodes. It is also impossible to pack all the supernodes exactly as per the mappings in the table. To handle this, we continue to map MPI tasks in the described shapes as long as it is possible to pack them neatly within the supernodes, and for the remaining supernodes (that are generally small in number), we do a random drawer assignment. For random nodes and random drawers mappings, we choose the node dimensions to be $4 \times 4 \times 2 \times 1$ tasks and the drawer dimensions to be $8 \times 8 \times 4 \times 1$ tasks.

Figure 12 presents the average, minimum and maximum data sent over the LL, LR and D links. Similar to the the 2D and 4D mappings on 64 supernodes, the default mapping leads to heavy traffic on all types of links. We observe similar LL and LR link usage for all mappings but the differences are significant for the usage of D links. Again, it is important to note that the communication volume on D links is almost ten times higher than the communication on LL and LR links (see y-axis). One difference from the 64 supernode mapping of 4D Stencil is that the blocked node mapping does not lower link utilization as compared to the default mapping. Also, the random drawers mapping does not perform as well as the random nodes mapping. The random nodes mapping and the mappings with indirect routing lead to the lowest D link usage which also translates to improvements in performance (Figure 13).

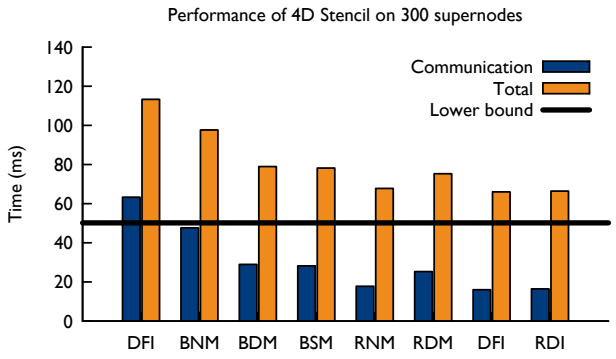


**Figure 13: Time spent in communication and overall execution per iteration for different mappings on 300 supernodes**

The performance results, in terms of execution time per iteration are as expected (Figure 13). As observed for the 64 supernode mapping of 4D Stencil, random nodes mapping and indirect routing cases give the best performance, followed closely by the random drawers mapping. The benefit is substantial, not only in terms of the communication time (which is reduced by 75% for the best mapping), but also for the per iteration time. We see a reduction of 42% in the application run time relative to the default mapping. The best mapping is worse by 24% when compared to the lower bound which indicates that there is still some room for improvement.

# 8. RELATED WORK

Mapping of guest graphs on to host graphs has been a subject of interest in mathematics, VLSI design and parallel computing since the 1980s. In parallel computing, several techniques were developed to map communication graphs to hypercubes in the 1980s [7, 18, 19] and to torus networks in the early 2000s [4, 20]. More recently, several application and runtime system developers have studied techniques for mapping [1, 5, 8, 10] to three-dimensional torus topologies with the emergence of supercomputers like the IBM Blue Gene and Cray XT/XE series.

Two-level direct networks were proposed recently by independent groups [2, 15, 16] and are being considered as an alternative to the more popular torus and fat-tree designs for building exascale machines. Hoefler *et al.* discuss mapping algorithms to minimize contention and demonstrate their applicability to the PERCS network through mapping simulations of sparse matrix-vector multiplication up to $1,792$ nodes [11]. Our work considers both regular and irregular communication graphs and presents simulation results on up to $307,200$ cores. Use of the BigSim simulation framework allows us to present detailed link utilization and timing information for different applications. We also discuss the interplay of mapping and routing and present best choices for both.

In this paper, we did not consider hybrid codes (MPI + OpenMP or pthreads). Mapping of hybrid codes is a specific instance of the general mapping problem since one can assume one core per node and one MPI task being mapped to each core. We also restricted our discussion to static communication patterns in this work. Changes in communication within an application can be handled by a dynamic load balancer, which in turn can deploy the discussed mapping algorithms. Considering inter-job contention, both static and dynamic is beyond the scope of this work and will be discussed in a future publication.

# 9. CONCLUSION

Multi-level direct networks have emerged as a new technology to connect a large number of processing elements together. Default MPI rank-ordered mapping with direct routing on such networks leads to significant hot-spots, even for simple two and four-dimensional near-neighbor communication patterns. This paper discusses techniques and analyzes various choices for congestion control on these networks.

We use detailed packet-level network simulations for up to three hundred thousand MPI tasks and three different communication patterns to compare various mappings – default mapping, blocked mapping to nodes, drawers, or supernodes and mapping to random nodes and drawers. We also compare direct versus indirect routing for some of the mappings. We show performance improvements of up to 42% for some mapping and routing combinations. For the communication patterns simulated in this paper, we find that if direct routing is used, mapping blocks of MPI tasks to random nodes gives the best performance and evenly distributed usage of D links. We also observe that indirect routing can achieve performance comparable to an intelligent mapping and obviates the need for mapping, at the cost of increasing overall traffic on the network.

This paper also highlights the utility of simulation-based predictions to analyze algorithms and make design choices before a parallel machine is installed and available for use. This will become increasingly important as machine sizes grow, making it essential to do application and hardware co-design.

# Acknowledgments

This research was supported in part by the Blue Waters sustained-petascale computing project (which is supported by the NSF grant OCI 07-25070 and the state of Illinois) and by a DOE Grant DE-SC0001845 for HPC Colony II. Runs for this paper were done on Blue Print and Ember, resources at NCSA. The authors would like to thank Ryan Mokos for building the Blue Waters network simulation model used in this paper. This document was released by Lawrence Livermore National Laboratory for an external audience as LLNL-CONF-491454.

## 10. REFERENCES


[1] T. Agarwal, A. Sharma, and L. V. Kalé. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*, April 2006.

[2] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony. The PERCS High-Performance Interconnect. In *2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI)*, pages 75 –82, August 2010.

[3] C. Bernard, T. Burch, T. A. DeGrand, C. DeTar, S. Gottlieb, U. M. Heller, J. E. Hetrick, K. Orginos, B. Sugar, and D. Toussaint. Scaling tests of the improved Kogut-Susskind quark action. *Physical Review D*, (61), 2000.

[4] G. Bhanot, A. Gara, P. Heidelberger, E. Lawless, J. C. Sexton, and R. Walkup. Optimizing task layout on the Blue Gene/L supercomputer. *IBM Journal of Research and Development*, 49(2/3):489–500, 2005.

[5] A. Bhatelé, E. Bohm, and L. V. Kalé. Optimizing communication for Charm++ applications by reducing network contention. *Concurrency and Computation: Practice & Experience*, 23(2):211–222, February 2011.

[6] A. Bhatele, S. Kumar, C. Mei, J. C. Phillips, G. Zheng, and L. V. Kale. Overcoming scaling challenges in biomolecular simulations across multiple platforms. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008*, pages 1–12, April 2008.

[7] F. Ercal and J. Ramanujam and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. In *Proceedings of the 3rd conference on Hypercube concurrent computers and applications*, pages 210–221. ACM Press, 1988.

[8] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, and M. C. Pitman. Blue matter: Approaching the limits of concurrency for classical molecular dynamics. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2006. ACM Press.

[9] Francine Berman and Lawrence Snyder. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing*, 4(5):439–458, 1987.

[10] F. Gygi, E. W. Draeger, M. Schulz, B. R. D. Supinski, J. A. Gunnels, V. Austel, J. C. Sexton, F. Franchetti, S. Kral, C. Ueberhuber, and J. Lorenz. Large-Scale Electronic Structure Calculations of High-Z Metals on the Blue Gene/L Platform. In *Proceedings of the International Conference in Supercomputing*. ACM Press, 2006.

[11] T. Hoefler and M. Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proceedings of the international conference on Supercomputing*, ICS '11, pages 75–84, New York, NY, USA, 2011. ACM.

[12] C. Huang, G. Zheng, S. Kumar, and L. V. Kalé. Performance Evaluation of Adaptive MPI. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006*, March 2006.

[13] L. Kalé and S. Krishnan. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In A. Paepcke, editor, *Proceedings of OOPSLA'93*, pages 91–108. ACM Press, September 1993.

[14] L. V. Kale, G. Zheng, C. W. Lee, and S. Kumar. Scaling applications to massively parallel machines using projections performance analysis tool. In *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, volume 22, pages 347–358, February 2006.

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

[16] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick. Exascale computing study: Technology challenges in achieving exascale systems, 2008.

[17] Michalakes, J., J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. The Weather Research and Forecast Model: Software Architecture and Performance. In *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, October 2004.

[18] S. Wayne Bollinger and Scott F. Midkiff. Processor and Link Assignment in Multicomputers Using Simulated Annealing. In *ICPP (1)*, pages 1–7, 1988.

[19] Soo-Young Lee and J. K. Aggarwal. A Mapping Strategy for Parallel Processing. *IEEE Trans. Computers*, 36(4):433–442, 1987.

[20] H. Yu, I.-H. Chung, and J. Moreira. Topology mapping for Blue Gene/L supercomputer. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 116, New York, NY, USA, 2006. ACM.

[21] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. V. Kalé. Simulation-based performance prediction for large parallel machines. In *International Journal of Parallel Programming*, volume 33, pages 183–207, 2005.