

# InfiniBand MPI Network Design and Simulation Using IBSim

Paul G. Howard, Ph.D., Chief Scientist, Microway, Inc.  
Stephen S. Fried, President and Chief Technology Officer, Microway, Inc.  
Copyright © 2009 Microway, Inc.

## Abstract

By using Microway's InfiniScope™ real-time InfiniBand monitoring tool to monitor HPC (High Performance Computing) clusters performing real world computations, we have observed that in many HPC applications, the amount of time spent carrying out computations is much greater than the time spent exchanging data between nodes. InfiniBand switch manufacturers would like customers to believe that they require enough switches to guarantee the availability of full bisectional bandwidth, but when InfiniBand fabrics are used to support fine grained parallel computation they seldom produce enough traffic to saturate such a fabric. To quantify how network topology, fabric size, and algorithmic parameters affect the efficiency of large clusters, Microway has created an HPC simulator called IBSim. IBSim gives a network designer the ability to simulate the performance of classes of applications running on an InfiniBand cluster. It can quantify effects previously understood only anecdotally and enable cost-effective provisioning of HPC clusters. We demonstrate the use of IBSim in a case study based on simulation of a two-dimensional nearest neighbor computation model. This paper describes the simulator, presents the study, and explores the results.

## Cluster design

The basic issue in HPC cluster design is to balance computation, communication, and cost. You want to use processing power on compute nodes efficiently to get as much computation done as possible. However, as you build a cluster with more compute nodes, you need more network fabric (mainly switches) so that the nodes can communicate with each other. Communication is overhead. Switches do not do any computation.<sup>1</sup> You need enough switches to provide full connectivity and sufficient bandwidth to get data to and from the processors without network delays, but you don't want to purchase more switches than you need because you can buy three or four compute nodes for the cost of each switch.

From an algorithm design point of view, you want to reduce communication overhead by confining inter-process communication to individual nodes and reducing path lengths for the messages that have to travel through switches. You especially want to keep traffic from waiting for blocked ports on switches.

The difficulty in designing a cluster and developing the software for a cluster is that there are complicated interactions among the nodes, the switches, and the algorithms used to solve your problem. Once you have more nodes than you can connect to a single switch, the number of switches required for a fully non-blocking fabric increases by a factor of at least 3. Because worst case traffic patterns seldom occur in practice, most algorithms can't consume all the bandwidth available in a fully connected fabric.<sup>2</sup> Intra-node communication typically consists of infrequent messages with neighboring nodes. In such situations messages are never blocked.

---

<sup>1</sup> As the saying goes, "You can talk, or you can work."

<sup>2</sup> The feature of switch fabrics that enables fully non-blocking communication among all processors is called "Constant Bisectional Bandwidth", or CBB.

To help sort out the complexity and find out what factors really matter, Microway has developed a system simulator called IBSim. IBSim assists the network designer by estimating the expected performance of an MPI parallel programming algorithm running on various InfiniBand-based fabrics. It can compare performance of algorithms running on different fabrics described by text files. A companion program, Microway's InfiniBand Fabric Designer, can generate these files with various topologies. By varying the number of switches and other parameters used to construct these topologies and simulating algorithms running on the resulting fabrics, Microway can evaluate how algorithm performance is affected by the choice of network topology.

## **HPC and InfiniBand background**

The cluster approach to HPC is to have lots of multicore multiprocessor compute nodes, typically two quad-core processors per node. Because many problems involve non-local computations, the nodes have to communicate over a network. To reduce the total time to solution, you need a low-latency, high-bandwidth network. The most cost-effective high-performance network fabrics today employ InfiniBand switches and HCAs (Host Channel Adapters). Current InfiniBand technology is capable of end-to-end message latencies of under 1.2 microseconds and bandwidths of up to 4 gigabytes per second (QDR, or Quad Data Rate, bandwidth with 4X data width). The latest generation of switches is based on the Mellanox<sup>®</sup> Technologies InfiniScale<sup>®</sup> IV 36-port QDR crossbar switch. Large fabrics are built up using many of these crossbar switches and 4X or 12X interconnects.<sup>3</sup> Each compute node has one or more HCAs to interface with the InfiniBand network.

Most HPC software uses MPI (Message Passing Interface) libraries. In general, you can run one MPI process per processor core, so you can have multiple processes running on a single node. Typical clusters have nodes with two quad core processors and can run 8 processes. Open source MPI implementations that support InfiniBand include MVAPICH and Open MPI.

## **The IBSim simulator**

IBSim simulates a workload of alternating computation and communication. The communication pattern (e.g., nearest-neighbor, computation-farming, random) is specified in a model, while the time required for computation and the amount of data that must be transmitted are specified as input parameters. The InfiniBand fabric and cluster properties are described in an input file. These properties include graphs describing how HCAs and switches are connected, data rates of the HCA to switch and switch to switch interconnects, and the number of processor cores per node. IBSim outputs an estimate of the wall clock time required to iterate the computation/communication pattern a specified number of times, as well as other statistics about network blocking and efficiency.

IBSim can be programmed to simulate a number of algorithmic models. One important and interesting class of HPC application is a nearest-neighbor computation, in which you are studying a physical system embedded in a two- or three-dimensional space with local interactions. Many computational fluid dynamics problems fall into this category. Often the system itself and its mathematical description are so complex that a closed-form solution is not possible.

To get insight into such a physical process, you can simulate it. You can model the physical space as a discrete grid of computation sites and perform one computation at each site for each time interval. The result of each site's computation depends only on the values of variables at nearby sites. You can trace the evolution of the system in time based on different initial conditions. More accurate results come

---

<sup>3</sup> The interconnects may be in the form of cables between external switch ports or in the form of wires on circuit boards. Even the largest InfiniBand switches are built up from smaller 24- or 36-port switches.

from subdividing the physical space into lots of computation sites that represent tiny parts of the physical space, and by using very short time intervals. If there are lots of sites and you are simulating short time intervals, the simulation becomes heavily compute-intensive.

What IBSim does is to simulate the computation and communication of the physical simulation to answer questions about how long the MPI simulation of the physical process will take under various hardware scenarios. The IBSim simulation model views the computation as an algorithm performing multiple iterations. In every iteration, each processor core does a fixed amount of work<sup>4</sup> for each of the computation sites in its cell. Then each cell exchanges data from its edge sites with each of its neighboring cells<sup>5</sup>. In this model, the input parameters are the amount of computation time per iteration for each site in a cell, the amount of data transferred for each site along the edge of a cell, and the number of iterations to simulate.

IBSim attempts to capture the essential features of an InfiniBand network in order to compare different fabric topologies. A simple topology file describes the network. It specifies the number of switches, the number of hosts, the number of cores per host, the maximum per-port bandwidth for each switch, the maximum bandwidth of each HCA, and the connection matrix for all switches and HCAs. IBSim computes InfiniBand forwarding tables for each switch according to a simple but realistic algorithm.

IBSim is a simulator, so it is only as accurate as the assumptions built into it. The assumptions are realistic, but they do not take all factors into account: latency is not considered directly but rather by adjusting link bandwidths according to the message size, asynchronous communication is not considered, and the forwarding table calculation in IBSim does not allow multiple paths between hosts. We run enough iterations to allow the simulation results to stabilize.

## Two-dimensional nearest neighbor model

The 2-D nearest neighbor model that we simulate assumes a square grid of just over a billion<sup>6</sup> computation sites. These sites are divided into equal-sized square cells<sup>7</sup>, one for each processor core in the cluster under investigation.

In the simulation model, computation and communication alternate. Each (simulated) core spends a specified time (with some random variation) doing the computation for its cell, and then sends data to its logically neighboring cells. The amount of data is proportional to the number of computation sites on the edge of the cell. If the neighboring cells are assigned to cores in the same host, communication is assumed to be very fast, proceeding at memory speeds. If not, messages are sent through the network, where they are routed according to the forwarding tables and subject to full or partial blocking if switch ports are congested.

The model attempts to distribute cells in the computational grid to processor cores in a way that makes good use of the locality of cores in a host and of hosts connected to a switch. Essentially the “cell-assigner” works its way through the grid of computation sites following a space-filling curve that does a better job of clustering cells than a simple back-and-forth scan of the cells, and a much better job than a random mapping.

---

4 Or almost fixed: some random variation can be specified in the amount of work.

5 4 neighboring cells in a 2-D simulation or 6 in a 3-D simulation.

6 More precisely:  $32,768 \times 32,768 = 1,073,741,824$  sites.

7 The cell sizes and number of cores used are truncated to integers for simplicity. This doesn't affect the results much, since we're mainly looking for large, general effects.

# Simulation

## Assumptions

We have run 2-D nearest-neighbor IBSim simulations for a number of different cluster sizes and topologies. The simulations assume the use of 36-port QDR InfiniBand switches, and hosts with two quad-core processors running at 3 GHz. We vary a number of cluster and algorithm parameters. Every simulation is run for 10 iterations, producing stable results.

## Load balancing

We tested two different levels of load-balancing. For the well-balanced case, we assume that computation times are normally distributed with standard deviation of 1 percent of the mean. For the unbalanced case, the standard deviation is 10 percent of the mean. The amount of data transmitted from each edge site is normally distributed with standard deviation of 1 percent of the mean in all cases.

## Algorithm granularity

In order to model different duty cycles of time computing relative to time communicating, we chose a few representative values for computation times and data packet sizes. We ran simulations of computations at four different granularities:

Granularity	Duty Cycle	
	Calculation Time per site	Bytes of Communication per edge site
Medium grained	1 $\mu$ s	1024
Fine grained	100 ns	1024
Very fine grained	10 ns	1024
Ultra fine grained	10 ns	10240

Each iteration of the computation over the billion computation sites involves a given amount of computation time which we consider at a rate of 3000 floating point operations per  $\mu$ s for a 3 GHz processor core, assuming 1 floating point operation per clock<sup>8</sup>. Then each cell exchanges data with neighboring cells, transmitting a given number of bytes for each site on its edge. A good measure of the granularity of a computation is the ratio of the number of bytes transmitted to the number of floating point operations (“bytes per flop”). This number depends on the size of the cell and hence on the number of processor cores; more precisely, it's proportional to the square root of the number of cores.

## Cluster size and topology

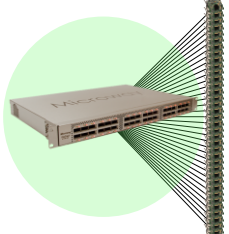
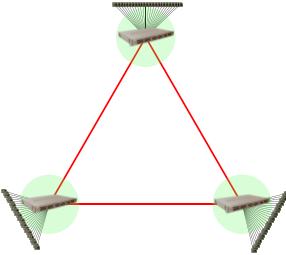
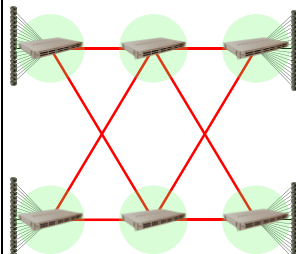
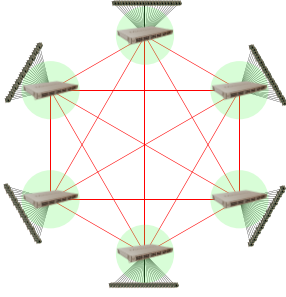
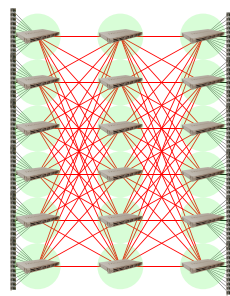
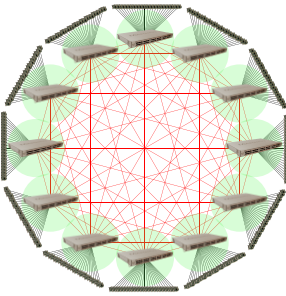
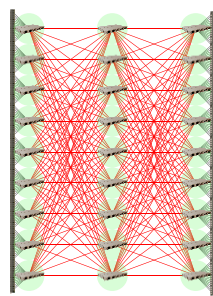
The simulated clusters range from 36 to 288 nodes. We simulated both FatTrees<sup>9</sup> and FasTrees<sup>10</sup> for the larger clusters. FasTrees use far fewer switches than FatTrees for the same number of nodes.

---

8 In fact, with the use of vector instructions like those in SSE, it is possible to perform more than one floating point operation per clock.

9 FatTrees guarantee full bisectional bandwidth. In a FatTree, two-thirds of the switches are called leaf switches, and the others are called spine switches. Leaf switches connect hosts to spine switches, but spine switches do not connect to hosts. Every nonlocal path within the network (a path that requires more than one switch) requires three hops through switches: leaf-spine-leaf. By introducing multiple levels of spine switches, we can create huge networks. These typically consist of “FatTrees of FatTrees” with leaf-spine-spine-spine-leaf connections.

10 A FasTree is essentially a fully connected network. Every switch is connected to some hosts and also to every other switch. This leads to fewer hops across the network and hence lower latency, as switch latency adds 120 ns, and copper cable latency adds 1 ns per meter (fiber cables have slightly greater latency than copper). All non-local paths are switch-switch, but there may be congestion over heavily used paths. A FasTrees generally requires far fewer switches than a FatTree for the same number of nodes. The switch saving becomes larger for larger clusters. The size of a FasTree is limited by the number of physical ports per switch.

Topology	Nodes	Number of switches	Active processor cores <sup>11</sup>	Fabric
Single switch	36	1	256	
FasTree	72	3	576	
FatTree	72	6	576	
FasTree	144	6	1089	
FatTree	144	18	1089	
FasTree	288	12	2304	
FatTree	288	27	2304	

<sup>11</sup> Because of the truncation-to-integer effect in the model, some cores in the simulated cluster will be inactive.

## Mapping algorithm to hardware

We also tested the effect of careful mapping of computation cells to processes. We find that by mapping nearby computation cells to processor cores that are physically close to each other on the cluster, we can sometimes achieve significantly improved performance. In the tables, “mapped” indicates the careful mapping, while “random” indicates a completely random assignment of computation cells to processes and processors.

## Example simulation parameters

Consider the 36 node single switch case with 256 cores where the cores are mapped carefully and the load is balanced so that the computation time has a standard deviation of 1 percent from the mean. For the medium grained case, the computation rate is 1  $\mu$ s per site on average and communication overhead is 1024 bytes per edge site. Each core computes a sub-matrix of  $2^{12} \times 2^{12}$  sites per core ( $1/256^{\text{th}}$  of the problem space of  $2^{15} \times 2^{15}$  sites used in this study) for a total computation of  $2^{24}$  sites per core per iteration. This calculation takes 1  $\mu$ s per site on average (each processor core is given a slightly different calculation rate to vary the load balance), so an average core would require  $2^{24} \times 1 \mu\text{s} = 16$  seconds. After computing, the core sends the specified amount of data to its neighbors according to the communication-per-edge-site parameter. For the example here, this is 1024 bytes per edge site – in this case,  $2^{12}$  edge sites per neighbor \* 1024 bytes = 4MB to each neighbor. Interior cores communicate with 4 neighbors, edge cores with 3, and corners with 2. The neighbors may be located within a processor core on the same node, in a remote node on the same switch, or in a remote node connected to another switch. The final iteration finishes after all cores receive all the messages from all neighbors. Because each core has a given computation speed determined by the load balancing parameter, fast nodes will have to wait for slow nodes to communicate their results. We use 30 GB/s to simulate local node memory bandwidth, and 3.15 GB/s on the InfiniBand fabric, adjusted to factor in latency.

## Results

The following tables show some representative results for the two-dimensional nearest neighbor model for different granularities. We use the standard definitions:

$$\text{parallel efficiency} = \text{sequential time} / (\text{number of active cores} \times \text{parallel time})$$

$$\text{parallel speedup} = \text{sequential time} / \text{parallel time}$$

with both sequential and parallel time being the actual wall clock time to complete the entire run.

### Medium grained

1  $\mu$ s computation per site, 1024 bytes sent per edge site

Load balanced (computation time standard deviation: 1 percent)

Topology	Nodes	Bytes/Flop	Mapped		Random	
			Efficiency	Speedup	Efficiency	Speedup
Single switch	36	0.00067	98.3	252	98.5	252
FasTree	72	0.00100	98.2	566	98.4	567
FatTree	72	0.00100	98.3	566	98.4	567
FasTree	144	0.00138	98.2	1069	98.3	1071
FatTree	144	0.00138	98.4	1072	98.4	1072
FasTree	288	0.00200	98.2	2262	98.4	2267
FatTree	288	0.00200	98.2	2262	98.4	2266

Unbalanced (computation time standard deviation: 10 percent)

Topology	Nodes	Bytes/Flop	Mapped		Random	
			Efficiency	Speedup	Efficiency	Speedup
Single switch	36	0.00067	86.1	220	86.4	221
FasTree	72	0.00100	85.4	492	84.6	487
FatTree	72	0.00100	85.7	494	84.7	488
FasTree	144	0.00138	85.5	931	84.8	924
FatTree	144	0.00138	84.2	916	84.7	922
FasTree	288	0.00200	85.0	1959	84.5	1946
FatTree	288	0.00200	84.3	1943	83.6	1927

In the medium grained simulation, probably the most common scenario in real world HPC, we see that load-balancing is the factor that most affects performance. This makes sense: at each iteration, the computation is at least partially delayed by the slowest processing cell, dragging down the behavior of the whole cluster in the long run. This is an algorithm issue. It's important not to require different cells to do different amounts of work. Mapping the computation cells to processors doesn't make much difference at this granularity: you can just connect the network and go. Note that FasTrees work just as well as FatTrees, while using one third to one half of the total number of switches.

### ***Fine grained***

100 ns computation per site, 1024 bytes sent per edge site  
Load balanced (computation time standard deviation: 1 percent)

Topology	Nodes	Bytes/Flop	Mapped		Random	
			Efficiency	Speedup	Efficiency	Speedup
Single switch	36	0.00667	97.9	251	96.5	247
FasTree	72	0.01000	97.4	561	94.1	542
FatTree	72	0.01000	97.3	560	94.1	542
FasTree	144	0.01375	96.1	1047	91.2	993
FatTree	144	0.01375	96.4	1050	92.4	1006
FasTree	288	0.02000	91.5	2107	87.2	2009
FatTree	288	0.02000	94.3	2172	88.7	2043

Unbalanced (computation time standard deviation: 10 percent)

Topology	Nodes	Bytes/Flop	Mapped		Random	
			Efficiency	Speedup	Efficiency	Speedup
Single switch	36	0.00667	86.4	221	85.1	218
FasTree	72	0.01000	84.4	486	84.7	488
FatTree	72	0.01000	84.9	489	84.1	484
FasTree	144	0.01375	85.1	926	84.1	915
FatTree	144	0.01375	84.8	923	84.2	917
FasTree	288	0.02000	84.4	1944	81.9	1886
FatTree	288	0.02000	84.3	1942	83.7	1928

In the fine grained simulation, which is about as far as you would expect to go in today's environment, we note that again load balancing makes some difference. Other differences are small. Parallel efficiency suffers a little bit as the clusters become larger. FasTrees are still as good as FatTrees.

### ***Very fine grained***

10 ns computation per site, 1024 bytes sent per edge site  
Load balanced (computation time standard deviation: 1 percent)

Topology	Nodes	Bytes/Flop	Mapped		Random	
			Efficiency	Speedup	Efficiency	Speedup
Single switch	36	0.06667	85.9	220	74.5	191
FasTree	72	0.10000	78.1	450	62.1	358
FatTree	72	0.10000	79.1	456	62.6	361
FasTree	144	0.13750	69.9	762	45.1	492
FatTree	144	0.13750	72.6	791	54.8	596
FasTree	288	0.20000	33.3	767	30.6	706
FatTree	288	0.20000	55.9	1287	42.3	974

Unbalanced (computation time standard deviation: 10 percent)

Topology	Nodes	Bytes/Flop	Mapped		Random	
			Efficiency	Speedup	Efficiency	Speedup
Single switch	36	0.06667	83.7	214	73.7	189
FasTree	72	0.10000	78.5	452	61.3	353
FatTree	72	0.10000	78.4	452	61.2	353
FasTree	144	0.13750	70.0	762	44.7	487
FatTree	144	0.13750	71.1	774	50.6	551
FasTree	288	0.20000	33.2	765	32.7	754
FatTree	288	0.20000	55.1	1270	42.3	974

Generally as we move to larger and faster clusters, we are hoping to reduce the granularity of our algorithmic solutions. In the very fine grained simulation, we see some new effects. Algorithm load balancing becomes unimportant, while processor mapping begins to make a difference. At this granularity, cluster size does make a difference. There are so many messages that must be sent in a large cluster that some will be blocked, and FasTrees just don't have enough bandwidth available.

### ***Ultra fine grained***

10 ns computation per site, 10240 bytes sent per edge site  
Load balanced (computation time standard deviation: 1 percent)

Topology	Nodes	Bytes/Flop	Mapped		Random	
			Efficiency	Speedup	Efficiency	Speedup
Single switch	36	0.66667	33.1	85	18.2	47
FasTree	72	1.00000	21.3	123	11.0	63
FatTree	72	1.00000	21.6	124	11.9	68
FasTree	144	1.37500	11.9	129	4.5	49
FatTree	144	1.37500	13.3	145	8.7	95
FasTree	288	2.00000	3.5	81	3.2	73
FatTree	288	2.00000	9.1	211	5.9	135

Unbalanced (computation time standard deviation: 10 percent)

Topology	Nodes	Bytes/Flop	Mapped		Random	
			Efficiency	Speedup	Efficiency	Speedup
Single switch	36	0.66667	33.9	87	18.1	46
FasTree	72	1.00000	21.7	125	10.8	62
FatTree	72	1.00000	22.0	127	11.9	69
FasTree	144	1.37500	11.9	129	5.1	55
FatTree	144	1.37500	13.5	147	8.7	95
FasTree	288	2.00000	3.5	81	3.2	74
FatTree	288	2.00000	9.2	211	5.8	134

For an ultra-fine grained computation, the HPC cluster model begins to break down. Adding more nodes doesn't even necessarily improve the wall clock time, as seen in the speedup column. At this point so much bandwidth is required that FatTrees are necessary. Note that this level of granularity is scarcely realistic by today's standards: we're computing for 10 ns for each site (that is, 30 clocks at 3 GHz, or 30 floating point operations at one flop per clock), then sending 10K of data for each edge site!

## Summary

By using the IBSim InfiniBand network simulator, we can determine which factors are important in designing a cluster. For the 2-D nearest neighbor computation model at normal granularity, we find that we can scale the cluster size with no loss in efficiency, and we don't have to worry about carefully mapping the algorithm to the processors. The main item of concern is to balance the computation so that each process does about the same amount of work in each iteration. The most important finding is that at this granularity, FasTrees run just as efficiently as FatTrees while using far fewer switches. This can save you money.

Other computation models can be simulated as well, such as a compute-farming application, in which one central master node hands out work to a number of compute nodes, which report back. In such a case, simulations confirm the obvious: blocking occurs at the master node, and you would want a large amount of variance in the amount of work farmed out so that the compute nodes don't compete for bandwidth to and from the master node.

Using IBSim, Microway can design highly efficient balanced fabrics. By configuring a network with as many processing nodes as possible and only just enough switches, we can save you money and provide you with a maximally cost-effective HPC cluster.