Reconfigurable Hybrid Interconnection for Static and Dynamic Scientific Applications

Shoaib Kamil, Ali Pinar, Daniel Gunter,
Michael Lijewski, Leonid Oliker, John Shalf, David Skinner
CRD/NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Abstract

As we enter the era of petascale computing, system architects must plan for machines composed of tens of thousands or even hundreds of thousands of processors. Although fully connected networks such as fat-tree interconnects currently dominate HPC network designs, such approaches are inadequate for thousands of processors due to the superlinear growth of component costs. Traditional low-degree interconnect topologies, such as the 3D torus, have reemerged as a competitive solution because the number of switch components scales linearly with the node count, but such networks are poorly suited for the requirements of many scientific applications. We present our latest work on a hybrid switch architecture called HFAST that uses circuit switches to dynamically reconfigure a lower-degree interconnect to suit the topological requirements of each scientific application. This paper expands upon our prior work on the requirements of non-adaptive applications by analyzing the communication characteristics of dynamically adapting AMR code and presents a methodology that captures the evolving communication requirements. We also present a new optimization that computes the under-utilization of fat-tree interconnects for a given communication topology, showing the potential of constructing a "fit-tree" for the application by using the HFAST circuit switches to provision an optimal interconnect topology for each application. Finally, we apply our new optimization technique to the communication requirements of the AMR code to demonstrate the potential of using dynamic reconfiguration of the HFAST interconnect between the communication intensive phases of a dynamically adapting application.

1 Introduction

The performance of commodity microprocessor-based supercomputing systems has been reliant on clock frequency improvements that result from the scaling of microchip features due to Moore's Law. Since the introduction of 90nm chip technology, however, heat density and changes in the dominant physical properties of silicon at such small feature size have moderated the pace of clock frequency improvements. As a result, the industry is increasingly reliant on unprecedented degrees of parallelism to keep pace with the demand for HPC performance improvements. Consequently, in the approaching era of petaflops computing, systems are expected to require connecting together tens or even hundreds of thousands of CPUs.

However, we observe that fully connected networks, such as fat-tree and crossbar interconnects, which currently dominate today's HPC systems, have component costs that scale superlinearly with the number of nodes in the system. Consequently, HPC system architects are increasingly considering interconnection network designs such as 2D/3D torii that scale in cost linearly with system scale (eg. IBM BlueGene/L, Cray XT3/RedStorm). There is ample evidence that many applications can operate efficiently on networks with lower topological degree if mapped appropriately onto the interconnect. However, adoption of networks with a lower topological degree of connectivity (TDC) leads to considerable problems with application mapping. Unless the application's communication topology requirements are known before application processes are assigned to nodes, the mapping of the application process topology to the fixed network topology may be hopelessly inefficient. This kind of topological mismatch can be mitigated by sophisticated task migration and job-packing by the batch system, but such migration impacts overall system efficiency. Also, some applications make use of all-to-all communication, such as 3D FFT's, requiring full cross-section bandwidth, and therefore suffer greatly on mesh topologies.

In response to these requirements, we introduced a hybrid approach to interconnect architecture called the Hybrid Flexibly Adaptable Switch Topology (HFAST) that employs optical circuit switches (Layer-1) to dynamically provision packet switch blocks (Layer-2) at runtime. In our previous [16] paper, we described the HFAST interconnect approach in detail, presented an analysis of the communication requirements for a number of DOE applications, and finally presented a linear-time algorithm for organizing switch resources to match the communication requirements of those applications. Whereas the applications studied in the earlier paper presented static communication patterns, this paper expands our analysis to applications with dynamic communication requirements, investigates how much of the available bandwidth is typically utilized, and presents how our observations can be utilized to design next generation scalable networks.

We will begin with a brief overview of the HFAST architecture, then follow with a characterization of the communication in our static codes. In Section 4, we describe the communication involved in an Adaptive Mesh Refinement (AMR) calculation. Finally, in Section 5 we explore whether a fully-connected network is necessary for all applications.

2 Hybrid Switch Architecture

Given the superlinear cost of constructing fully connected networks (FCNs) such as fat-trees or crossbars, such interconnects will rapidly become the dominant cost in large-scale systems. As we move towards petaflops systems with tens or hundreds of thousands of processors, the industry will be hard-pressed to continue to build cost-effective fat-tree networks that offer balanced performance into the petascale era. For an alternative to fat-trees and traditional packet-switched interconnect architectures, we can look to recent trends in the high-speed wide area networking community, which has found that *lambda-switching* (hybrid interconnects composed of circuit switches together with packets switches) presents a cost-effective solution to a similar set of problems.

2.1 Circuit Switch Technology

Packet switches, such as Ethernet, Infiniband, and Myrinet, are the most commonly used interconnect technology for large-scale parallel computing platforms. A packet switch must read the header of each incoming packet in order to determine on which port to send the outgoing message. As bit rates increase, it becomes increasingly difficult and expensive to make switching decisions at line rate. Most modern switches depend on ASICs or some other form of semi-custom logic to keep up with cutting-edge data rates. Fiber optic links have become increasingly popular for cluster interconnects because they can achieve higher data rates and lower bit-error rates over long cables than is possible using low-voltage differential signaling over copper wire. However, optical links require a transceiver that converts from the optical signal to electrical so the silicon circuits can perform their switching decisions. The Optical Electrical Optical (OEO) conversions further add to the cost, latency, and power consumption of switches. Fully-optical switches that do not require an OEO conversion can eliminate the costly transceivers, but per-port costs will likely be higher than an OEO switch due to the need to use exotic optical materials in the implementation.

Circuit switches, in contrast, create hard-circuits between endpoints in response to an external control plane – just like an old telephone system operator's patch panel – obviating the need to make switching decisions at line speed. As such, they have considerably lower complexity and consequently lower cost per port. For optical interconnects, micro-electro-mechanical mirror (MEMS) based optical circuit switches offer considerable power and cost savings as they do not require expensive (and power-hungry) optical/electrical transceivers required by the active packet switches. Also, because non-regenerative circuit switches create hard-circuits instead of dynamically routed virtual circuits, they contribute almost no latency to the switching path aside from propagation delay. MEMS based optical switches, such as those produced by Lucent, Calient and Glimmerglass, are common in the telecommunications industry and their prices are dropping rapidly as the market for the technology grows larger and more competitive.

2.2 Prior Work

Circuit switches have long been recognized as a cost-effective alternative to packet switches, but it has proven difficult to exploit the technology for use in cluster interconnects because the circuit switches do not understand message or packet boundaries. It takes on the order of milliseconds to reconfigure an optical path through the switch, and one must be certain that no message traffic is propagating through the light path when the reconfiguration occurs. In comparison, a packet-switched network can trivially multiplex and demultiplex messages destined for multiple hosts without requiring any configuration changes. Please refer to the paper that introduced HFAST [16] and to a paper from the IBM DARPA-HPCS project [2] for a more in-depth discussion of the broad range of existing circuit-switch interconnect implementations that have been proposed over the past decade.

The hybrid interconnect architecture described in the IBM HPCS paper, which is based on Gupta and Shenfeld's earlier work on Interconnection Cached Networks (ICN) [8], has the most similarity to our work. Whereas the ICN would require task migration to preserve optimal graph embedding, the HFAST approach allows tasks to remain in-situ as the interconnect adapts to the evolving job requirements. This feature is particularly advantageous for the adaptive applications that are the focus of this paper.

2.3 HFAST: Hybrid Flexibly Assignable Switch Topology

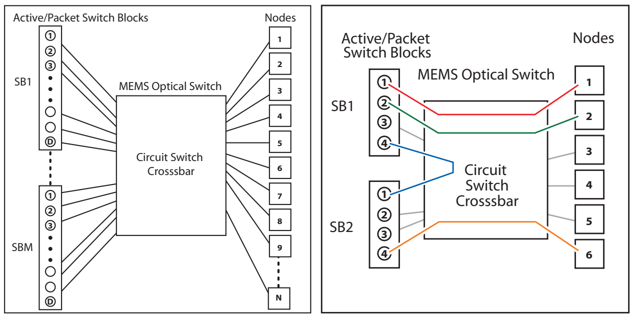


Figure 1: General layout of HFAST (left) and example configuration for 6 nodes and active switch blocks of size 4 (right). In this example, node 1 can communicate with node 2 by sending a message through the circuit switch (red) in switch block 1 (SB1), and back again through the circuit switch (green) to node 2.

We refer to our interconnect architecture as HFAST (Hybrid Flexibly Assignable Switch Topology). HFAST is composed of a mix of Layer-1 (passive/circuit switches) and Layer-2 (active/packet switch blocks) that operate in a coordinated fashion. HFAST derives its name from its ability to treat the packet switches as a flexibly assignable pool of resources that support adaptable formation of communication topologies without any job placement requirements. As shown in Figure 1, the circuit switches are used to dynamically provision packet switch blocks at runtime, resulting in an interconnect that requires far fewer of the more expensive packet switches than a fully connected (FCN) fat-tree or crossbar solution.

We envision using the circuit switch to set up dedicated circuits primarily for the bandwidth bound messages. To understand this design choice requires a short discussion of the bandwidth-delay product.

2.4 Handling of Small Messages

The product of the bandwidth and the delay for a given point-to-point connection describes precisely how many bytes must be "in-flight" to fully utilize available link bandwidth. This can also be thought of as the minimum size required for a non-pipelined message to fully utilize available link bandwidth. Vendors commonly refer to an $N_{1/2}$ metric, which describes the message size below which you will get only 1/2 of the peak link performance. The $N_{1/2}$ metric is typically the same as the bandwidth-delay product.

While RDMA can, in theory, support pipelining of messages that are smaller than the bandwidth-delay product, our measurements on a number of different existing interconnect architectures saw minimal improvements in performance of applications that were dominated by the smaller latency bound messages [16]. Therefore our analysis focuses on messages that are larger than the bandwidth-delay product, which is the minimum message size that can theoretically saturate the link. Our analysis filters out the latency-bound messages with the assumption that they can be carried either as transit traffic that requires several hops through the HFAST interconnect topology to reach its destination, or routed to a secondary low-latency low-bandwidth interconnect that uses much lower cost components for handling collective communications with small payloads. An example of such a low-latency secondary network can be found in the design of the BlueGene/L Tree network which is designed to handle fast synchronization and collectives where the message payload is typically very small.

3 Non-Adaptive Scientific Applications

In order to evaluate the potential effectiveness of utilizing hybrid interconnect networks, we must first develop an understanding of the communication requirements of scientific applications across a broad spectrum of parallel algorithms. Our previous work [16] investigated a number of applications that exhibited a runtime topology that is essentially static during the entire runtime of the code. In addition to several of the previously-studied applications, we add ELBM3D and BeamBeam3D, two applications used in physics research. Each of these applications is actively run at multiple supercomputing centers, consuming a sizable amount of computational resources. Table 3 summarizes the static applications studied in this work. Detailed descriptions of the algorithms and scientific impact of these codes have been detailed elsewhere [6, 10, 11, 14].

Name	Lines	Discipline	Problem and Method	Structure
Cactus [5]	84,000	Astrophysics	Einstein's Theory of GR via Finite Differencing	Grid
LBMHD [11]	1,500	Plasma Physics	Magneto-Hydrodynamics via Lattice Boltzmann	Lattice/Grid
GTC [10]	5,000	Magnetic Fusion	Vlasov-Poisson Equation via Particle in Cell	Particle/Grid
MADbench [4]	5,000	Cosmology	CMB Analysis via Newton-Raphson	Dense Matrix
ELBM3D	3,000	Fluid Dynamics	Fluid Dynamics vi Lattice Bolzmann	Lattice/Grid
BeamBeam3D [14]	23,000	Particle Physics	Poisson's equation via Particle in Cell and FFT	Particle/FFT

Table 1: Overview of scientific applications examined in this work.

3.1 Communication Topology Characteristics

In this section, we present the topological connectivity for each application by representing the volume and pattern of message exchanges between all tasks. By recording statistics on these message exchanges using the IPM [1] profiling layer, we can form an undirected graph which describes the topological connectivity required by the application. This graph is undirected because we assume that switch links are bi-directional, resulting in topology charts that are always symmetric about the diagonal. From this graph we can calculate certain reduced quantities which describe the communication pattern at a coarse level. In particular, we examine the maximum and average TDC (total degree of connectivity, which is the number of other tasks a particular task communicates with) of each code, a key metric for evaluating the potential of the HFAST approach. Additional details may be found in [16].

Figure 2 shows the communication topologies for the static applications in this study. In BeamBeam3D (left), the domain is decomposed such that the two beams involved in the communication are split over the processors. In this particular run, each processor communicates with its 127 neighbors in the same row of the beam, plus 3 neighbors in the column dimension, and 128 processors of the other beam. Thus we observe a TDC of 258 both as a max and as an average at $P = 1024$, or about 25% of processors.

In Figure 2 (second from left), we see that the ghost-zone exchanges of Cactus result in communications with "neighboring" nodes, represented by diagonal bands. In fact, each node communicates with at most 6 neighbors due to the regular computational structure of this 3D stencil code. The average and max TDC is 5 and 6 respectively, because some nodes are on the boundary and therefore

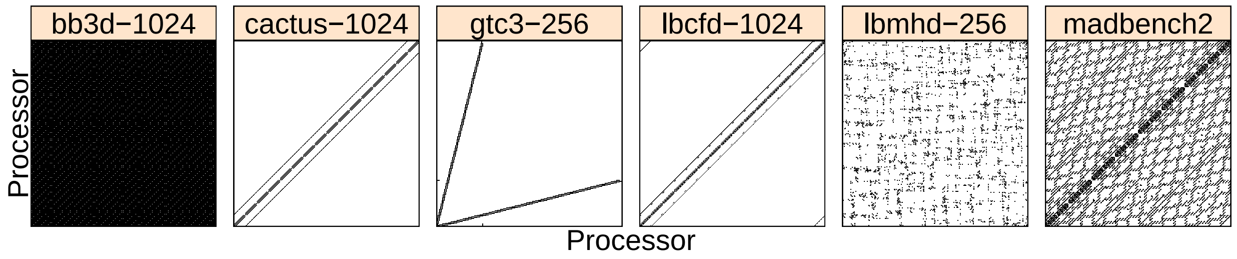


Figure 2: Communication topology of the static applications in our study (from left to right): BeamBeam3D, Cactus, GTC3, ELBM3D, LBMHD, and MadBench2

have fewer communication partners. Note however that the low TDC indicates limited utilization of an FCN architecture.

GTC has a regular communication structure reflected in Figure 2 (third from left). This particle-in-cell calculation uses a one-dimensional domain decomposition across the toroidal computational grid, causing each processor to exchange data with its two neighbors as particles cross the left and right boundaries. Additionally, there is a particle decomposition within each toroidal partition, resulting in an average TDC of 4 with a maximum of 17 for the 256 processor test case. These small TDC requirements clearly indicate that most links on an FCN are not being utilized for the GTC simulation.

The communication topology for ELBM3D is in in Figure 2 (fourth from left). At $P = 1024$, the code has a TDC of 6. Each node streams data to its neighbors in the three dimensions, yielding the extremely low TDC. At this concurrency, each process communicates with only 0.6% of its neighbors.

The connectivity of LBMHD is shown in Figure 2 (fifth from left). Structurally, we see that the communication, unlike Cactus, is scattered (not occuring on the diagonal). This is due to the interpolation between the diagonal streaming lattice and underlying structure grid. Note that although the 3D LBMHD streams the data in 27 directions, the code is optimized to reduce the number of communicating neighbors to 12. This degree of connectivity (12) represents both the aveage and maximum TDC and is insensitive to the concurrency level.

MADbench's connectivity is shown in Figure 2 (rightmost). This application uses ScaLAPACK's `pdgemr2d` function to remap subsets of the matrices to different processors before calling `pdgemm`. This communication is apparent in the clusters along the diagonal which result from the matrix distribution. At a concurrency of 256, the TDC is a max of 44, with the average degree of connectivity being 39.

As a whole, these codes reflect the class of applications with relatively static communication topology that vastly underutilize a fully connected network.

4 Adaptive Mesh Refinement Calculation

Adaptive mesh refinement (AMR) is a powerful technique that reduces the computational and memory resources required to solve otherwise intractable problems in computational science. Typically, AMR has been applied to physical systems that are modeled by a governing set of partial differential equations (PDEs). The AMR strategy solves the system of PDEs on a relatively coarse grid, and dynamically refines it in regions of scientific interest or where the coarse grid error is too high for proper numerical resolution. Without some form of adaptivity, naively increasing the grid resolution uniformly across the entire computational domain can result in computationally prohibitive expense for realistic cases.

However, adaptive codes tend to be far more complicated than their uniform grid counterparts. Significant software infrastructure must be developed to manage refined regions and ensure correctness between the various levels of refinement. A key component of an AMR calculation is dynamic mesh regridding, which dynamically changes the grid hierarchy to accurately capture the physical phenomena of interest. Cells requiring enhanced resolution are identified and tagged using a user-supplied error indicator, and then grouped into rectangular patches that sometimes contain a few cells that were not tagged for refinement. These rectangular patches are subdivided to form the grids at the next level. The process is repeated until either the error tolerance criteria is satisfied or a specified maximum level of refinement is reached. When new grids are created at level $l + 1$, the data on these grids are copied from previous grids at the same level (where possible) using an efficient point-to-point protocol;

otherwise the data is interpolated from the underlying level l grids.

4.1 HyperCLaw Overview

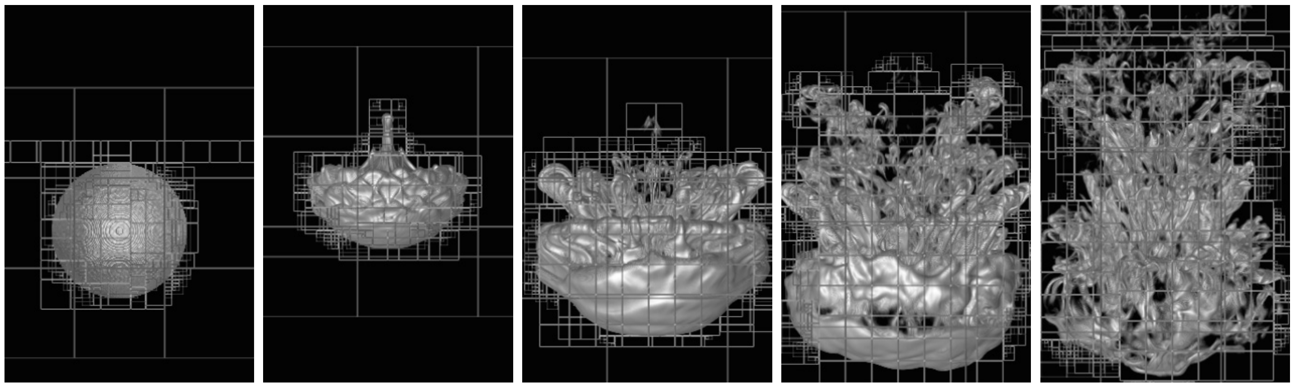


Figure 3: Deformation of Helium bubble as it passes through shock front in the AMR simulation.

Our work examines HyperCLaw, a hybrid C++/`Fortran` AMR code developed and maintained by CCSE at LBNL [7,15] where it is frequently used to solve systems of hyperbolic conservation laws using a higher-order Godunov method. The HyperCLaw code consists of an application layer containing the physics classes defined in terms of virtual functions. Data blocks are managed in C++, in which ghost cells are filled and temporary storage is dynamically allocated so that when the calls to the physics algorithms (usually finite difference methods implemented in `Fortran`) are made, the same stencil can be used for all points and no special treatment is required. By structuring the software in this manner, the high level objects that encapsulate the functionality for AMR and its parallelization are independent of the details of the physics algorithms and the problem being solved. This simplifies the software engineering process of adding/replacing physics modules. In HyperCLaw most of the communication overhead occurs in the FillPatch operation. FillPatch starts with the computational grid at a given level, adds ghost cells five layers thick around each grid, and then fills those cells either by copying from other grids at that level, or by interpolating from lower level (coarser) cells that are covered by those ghost cells. This gives FillPatch a very complicated nonlinear communication pattern. Once all the ghost cells for each grid are filled with valid data, a higher-order Godunov solver is applied to each grown grid. This solver is very compute-intensive, taking upwards of a full second for the problems we ran in this study, during which time no interprocessor communication occurs.

4.2 Evolution of Communication Topology

The HyperCLaw problem examined in this work profiles a hyperbolic shock-tube calculation, where we model the interaction of a Mach 1.25 shock in air hitting a spherical bubble of helium. This case is analogous to one of the experiments described by Haas and Sturtevant [9]. The helium is a factor of 0.139 less dense than the surrounding air which causes the shock to accelerate as it enters the bubble and subsequently generates vorticity that dramatically deforms the bubble. An example of the kind of calculation HyperCLaw performs is seen is Figure 3, along with an overlaid representation of the grids used.

For this paper, we limit the refinement to three levels (0, 1 & 2), with 0 being the lowest (or base) level and 2 being the highest (or finest) level, where most of the computation time is spent. Level 1 is effectively the base grid at level 0, refined by a factor of two in each of the three coordinate directions, with Level 2 being similarly refined from Level 1. If this were a uniform grid calculation, each level beyond 0 would represent eight times the computation of the level less refined than it. With AMR technology, only refining where there is interesting physics, each level may only be 2-4 times as costly as the previous level.

In Figure 4 we see the communication topologies for communication at level 1 and at level 2. These two levels exhibit very different communication. Because level 1 represents a coarser problem than

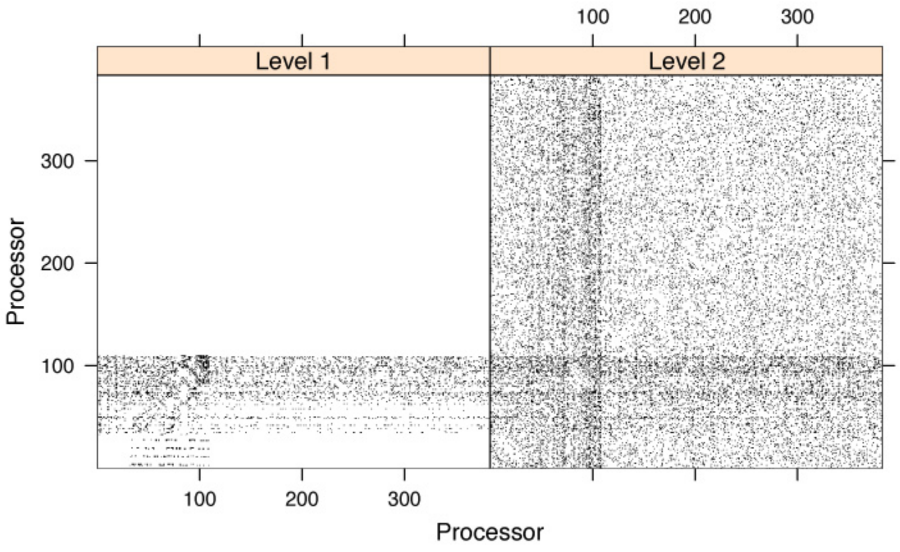


Figure 4: Topology of HyperCLaw for AMR levels 1 and 2.

that at level 2, there are far fewer grids at level 1 than at level 2. Since each 3D grid can potentially communicate with at least six tasks, the more grids at a level (and hence the more grids that a task owns), the potentially higher number of processes that task must communicate with.

The structure of the AMR calculation provides opportunity for HFAST to adapt its configuration to the evolving topology, because the timesteps each contain distinct communication and computation phases separated by a regridding operation that takes on the order of hundreds of milliseconds. In this space, the optical switches can be reconfigured to take advantage of changes in the most significant commmunicating partners.

For this to work, two requirements must be met: first, the proportion of communicating partners (for messages over the bandwidth-delay product) must be less than P in order to justify use of a lower degree interconnect topology. Secondly, the set of communicating partners must not change dramatically at each step. The results show that these are both true for the HyperCLaw code.

In the left panel of Figure 5, we can see the average percent of communicating partners in a typical (Level 2) timestep, with message size cutoffs on the x-axis and a separate curve for each of six cuts of the message volume. Thus, although fully 25% of the communication adjacency matrix is full when all messages are considered, 85% of the communication volume is accounted for by ony 15% of the communicating partners.

The change in the number of communicating partners as the AMR computation evolves closely tracks the progression from Level 1 to Level 2 timesteps. This is illustrated in the right panel of Figure 5, which shows percent of communicating partners across 60 timesteps, ignoring messages smaller than 16KB (which can be dealt with using multiple hops and/or a secondary network for collectives and small messages) and for two message volumes of 85% (lower bar) and 100%. Every fifth timestep is at AMR Level 1, the rest are Level 2. This graph shows a gradual evolution in the number of communicating partners within each AMR Level. Although the total number of communicating partners is very similar within a level, Figure 6 also shows that these are, with the exception of transitions between AMR levels, over 80% of the *same* partners from one step to the next. In this graph, each bar represents the percentage of communicating partners in the current timestep that were also communicating partners in the previous timestep. This implies that reconfiguration of the topology does not need to be performed between every AMR timestep, since, for example, an optimal topology at timestep 2 will also be optimal or nearly optimal for timesteps 3 and 4. Instead, reconfiguration need only occur when going between levels of an AMR calculation.

In the next section, we explore whether fully-connected networks are necessary to meet the requirements of our dynamic and static applications.

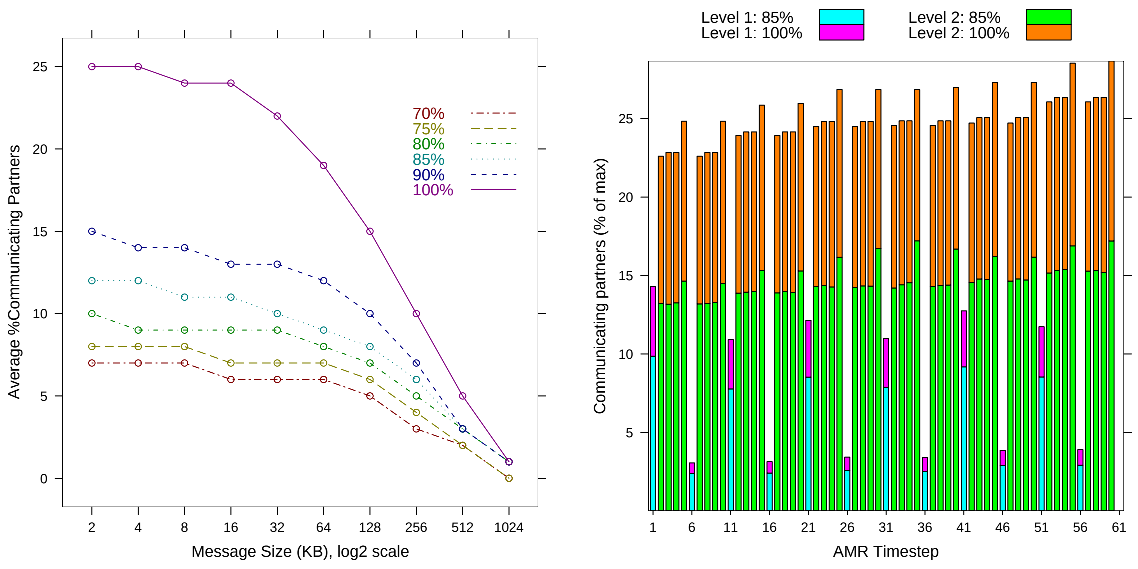


Figure 5: Number of Partners for varying volume and message size cutoffs in a Level 2 timestep (left) and for 85% volume of messages greater than 16K across 60 Level 1 and Level 2 timesteps (right)

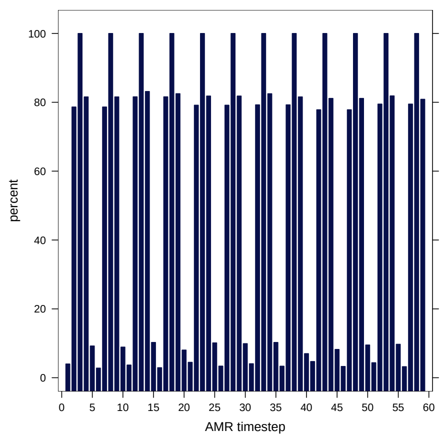


Figure 6: Same AMR Partners

5 Optimizing the Interconnection Topology: *The Fit-Tree Approach*

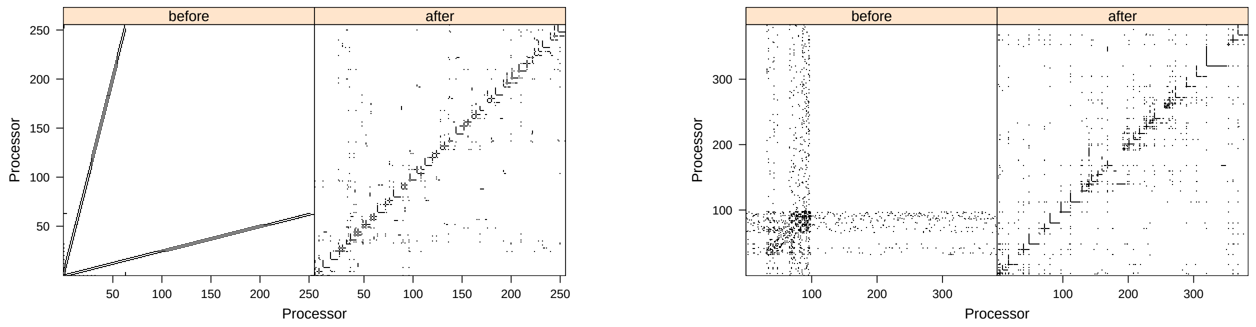
In this section we investigate whether the full bandwidth of a fat-tree network is required for scalable applications that target hundreds of thousands or even millions of processors. First, we describe a processor allocation strategy for fat-trees that can be implemented easily with our reconfigurable interconnection network and show the effectiveness of this strategy in reducing the average number of hops per message. Then we show that four our applications, required bandwidth is much smaller than the full bandwidth available in a fat tree, especially for large numbers of processors. Finally, we discuss how this observation can be exploited to design an interconnection network that is as effective as a fat-tree, but more efficient in terms of switching resources.

5.1 Effect of Processor Assignment

The effect of processor allocation has been well-observed in the literature and has recently redrawn attention due to increasing numbers of processors in state of the art supercomputers [3]. Processor allocation aims at relocating frequently communicating processes such that their positions in the processor topology are closer, resulting in smaller latency, as well as ensuring the network is not congested due to messages consuming bandwidth on more links than necessary. While the processor allocation problem is very hard for the general case (and in fact is NP-Complete [13] even for the special case of a binary tree), in this section we will restrict ourselves to a special case in processor allocation on a fat-tree using a heuristic based on graph partitioning.

Given a graph $G = (V, E)$, graph partitioning decomposes the vertex set V into two or more components to minimize the number of edges that connect vertices from different partitions, while preserving specified balance criteria. We say an edge is *cut* if it connects two vertices from different components, and a vertex is a *boundary* vertex if it is connected to a cut edge. In our model, each processor is represented by a vertex of the graph, and two vertices are connected if the associated processors communicate. As the balance criterion, we want the parts to include equal numbers of vertices so the resulting decomposition fits onto a fat-tree. This criterion must reflect the HFAST networking model which provisions a set of layer-2 packet switches such that the communication topology required by the application can be met by the resulting network. An economical HFAST implementation must use commodity packet switches, which tend to be of small radix. In this work, we use 4×4 switches, so that the root of the tree has 4 branches, and all other intermediate nodes have 2 branches. The packet switch radix determines the number of branches from a node of the tree, and thus the number of parts for each partitioning.

After this mapping, messages that go to the top of the tree correspond to the edges that are on the boundary after the first partitioning. Similarly, messages that are routed back to a processor at the second level from the top correspond to cut edges after a second partitioning during recursive bisection. Minimizing the number of cut edges in partitioning pushes the communication to the lower levels of the fat-tree, and thus can be used as a heuristic to minimize the average number of hops per message.



gtc-256 hyperclaw

Figure 7: Communication topologies before and after our processor allocation algorithm.

We implemented our technique for processor allocation, using the MeTiS graph partitioning package [12], and experimented with communication graphs of various applications. The effects of permutations are depicted in Figure 7. In this figure, processors are ordered so that after a partitioning, all processors of the first part are ordered before those in the second part, and this is repeated recursively.

Figure 8 shows the effects of processor allocation on the average number of hops per message. The number of hops for each message decreases by 22% percent on average and as high as 49% as in GTC with $P = 256$. The results on the Cactus datasets demonstrate the increasing effectiveness of our techniques as processor size increases, which reveals how crucial processor allocation will be when the number of processors reaches the order of millions.

Another advantage of the HFAST reconfigurable network is the ability to remap processes to processors and to interconnection topology requirements. This ability can be especially effective for adaptive applications, where the computation adapts itself to the physical phenomena being simulated. We have applied our techniques to the HyperCLaw communication graphs, and our results are presented in Figure 9. In these experiments, we used a three-way initial decomposition to map 384 (3×128) processors to a fat-tree. The results show that the average number of hops can be consistently decreased by 12% to 20% after each timestep. Results also show that permutation needs to be repeated before each communication step, since the communication patterns change significantly after each time step. There will be enough communication-free computation time to reconfigure the network between two

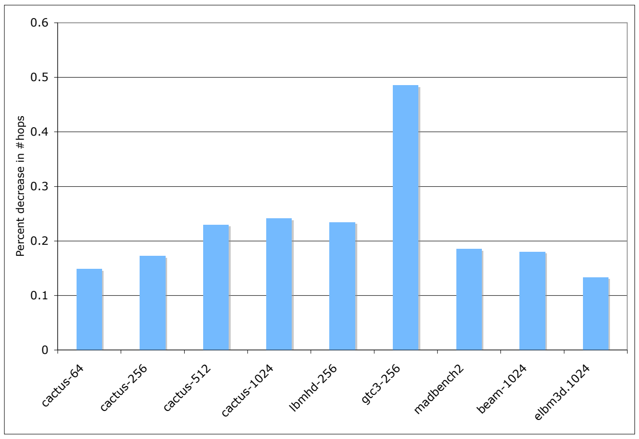


Figure 8: Average number of hops. The x-axis corresponds to different applications and the y-axis shows percent decrease in the number of hops. Note here 0.1 means 10%.

timesteps, since the time for computation is on the order of a second, while reconfiguration time is estimated on the order of milliseconds.

The fit-tree optimization in Section 5.3 is equally applicable to optimizing the embedding of communication topologies for the IBM hybrid OCS interconnect [8]. However, analysis of HyperCLaw's communication requirements show that any lower-degree interconnect with a fixed topology or even IBM's OCS approach would require a considerable amount of task migration in order to maintain an optimal embedding of the evolving communication topology. No such task migration is required for HFAST to reconfigure for the evolving communication topology requirements.

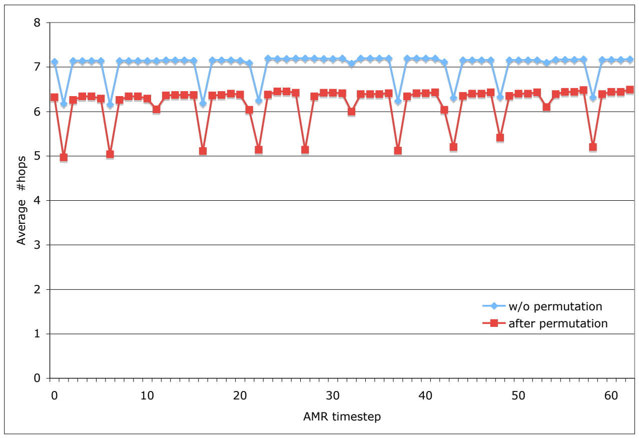


Figure 9: Average number of hops for hyperclaw. The y-axis corresponds to the average number of hops, and the x-axis corresponds to the AMR time step.

5.2 Fat-tree Utilization

In this section we will investigate how much of the fat-tree bandwidth is utilized by typical applications. We will look at the average and the worst case scenarios, as well as effects of processor allocation. In the average case scenario, each processors picks one of its neighbors to send uniformly at random, whereas, in the worst case scenario, each processors sends to its furthest neighbor in the fat-tree to maximize the total bandwidth being used. Such a scenario is only a worst case, not likely to happen, but our results show that even at this extreme case, the bandwidth of a fat-tree is not fully utilized by these applications.

The results for the average case fat-tree utilization are presented in Figure 10. As seen in this figure, the bandwidth utilization degrades sharply as we go to the higher levels of the tree and is uniformly

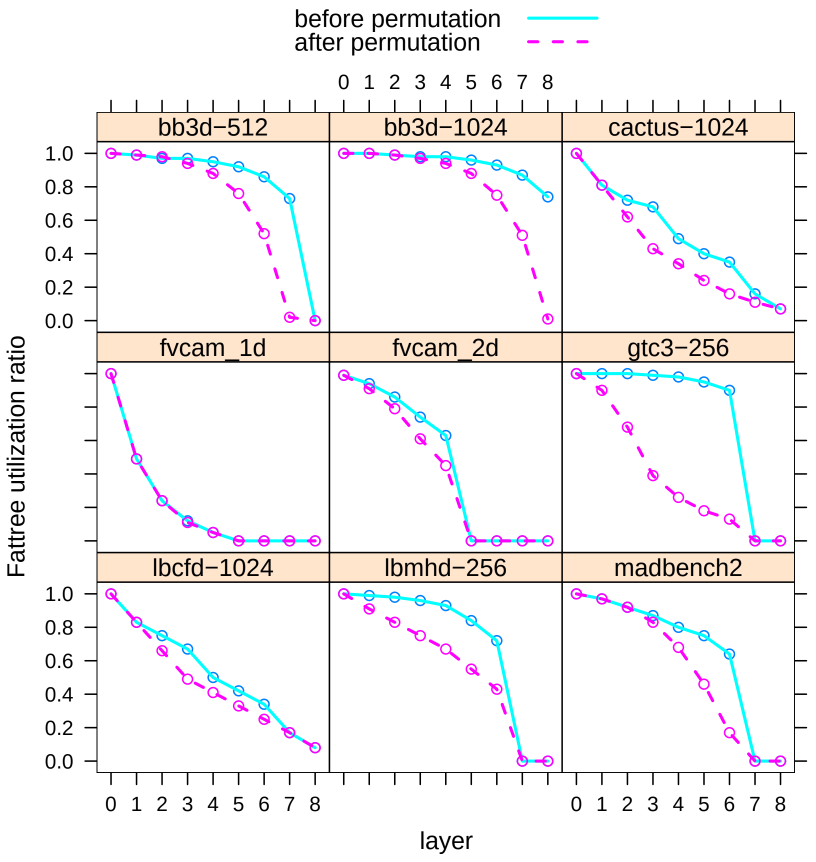


Figure 10: Fattree utilization before and after permutation for the static applications BeamBeam3D p=512 and p=1024, (2) CACTUS p=256,512, and 1024, GTC3 p=256, LBCFD p=1024, LBMHD p=256, and MADBench2 p=256.

below 20% at the highest level, if permutation is used. Although the bandwidth utilization is still low without them, permutations bring consistency, as seen in BeamBeam3D at $P = 1024$.

In Figure 11, we present the results for the worst case scenario at the highest level of the fat-tree, which correspond to maximum bandwidth that can possibly be utilized at that level. As the results show, the full bandwidth can possibly be utilized in only 3 out of 9 applications in our study, at least for some period of time. In the 5 other applications, however, at most 50% of the available bandwidth can be utilized at any time. The trend of decreasing maximum utilization with increasing processors numbers can be observed in the Cactus group. These results clearly show that applications that can scale to millions of processors do not require the full bandwidth of a fat-tree.

5.3 Fit-Tree: A New Interconnection Topology Optimization

We are currently working on designing a new interconnection network that will adapt the recursive structure of the fat-tree, but will better capture applications communication requirements. Fat-trees have a layered structure, where the total bandwidth between any two stages is a constant, and thus in the same order of the number of processors. The number of layers is $O(\lg P)$, where P is the number of processors. All together, the cost of a fat-tree scales as $P \lg P$, for $\lg P$ layers of bandwidth $O(P)$. While the fully- onnected interconnect topology offer much simpler job scheduling and communication characteristics for smaller scale systems, fat-trees are not regarded as a cost-effective option for petascale systems that are expected to scale to hundreds of thousands of processors.

We believe the core idea of a fat-tree can be used for scalable interconnection designs after tuning its structure to match scalable applications. Our key observation is that for most applications that target millions of processors, the required bandwidth decreases as we go higher in the fat-tree, and is only a small portion of the available bandwidth at the top level. This is expected since the maximum

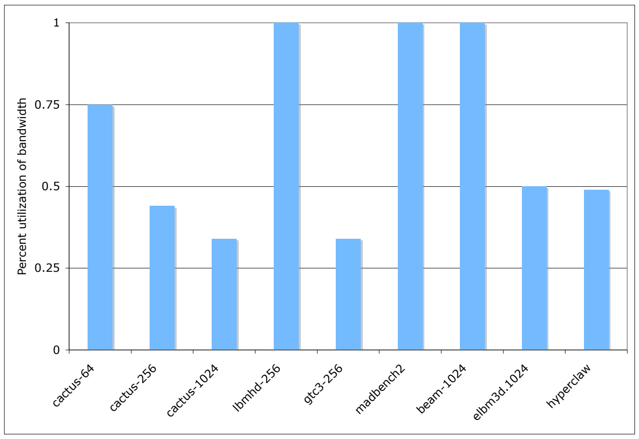


Figure 11: Maximum utilization of the top level of a fat tree

bandwidth that can be utilized at the highest level is determined by the bisection bandwidth of the processor communication graph, or more precisely the number of vertices on the boundary of the bisection. This bisection bandwidth is much smaller than the number of vertices, as observed in our experiments. For instance the bisection bandwidth of a $P \times P$ 2D-regular grid is only $O(P)$, and a $P \times P \times P$ 3D-regular grid is only $O(P^2)$. Just as an AMR code applies refinement to the subset of the problem domain that requires it, the reconfigurable interconnect is able to apply the dedicated high-bandwidth links only to the portions of the job that require it – delivering efficient application performance with a lower global bisection bandwidth.

This observation can be exploited to design a new interconnect, with available bandwidth that decreases at higher levels of the tree, which we call a *fit-tree*. A *fit-tree* is a layered network just like a fat-tree, but unlike a fat-tree total bandwidth between layers decreases as we move to higher levels. This not only decreases the cost by reducing the necessary hardware, but can also improve performance by reducing the number of layers of the tree, and thus the number of hops required.

While the details of the fit-tree paradigm are beyond the scope of this paper, the authors believe this brief discussion gives the readers a flavor of where our research is directed and how our results in this paper can be used to design interconnection networks for next generation petascale computers. Future work will further explore the feasibility of fit-tree networks and the relative cost and resource savings when compared to a fat-tree.

6 Summary and Conclusions

This work has extended our previous work on HFAST in two important ways. First we have shown the this networking concept is applicable to applications with dynamically changing connectivities. Second, we introduced the fit-tree approach to switch provisioning which makes HFAST dramatically more effcient in its allocation of switch resources. Thirdly, we have shown that optimizations to the communication topology for any fixed-topology interconnect would require task migration in order to match the evolving communication topology of adaptive applications. HFAST overcomes the need for task migration through its ability to dynamically optimize the interconnect topology during the compute-intensive phase of a bulk-synchronous adaptive application such as HyperCLaw.

There is a crisis looming in parallel computing driven by rapidly increasing concurrency and the scaling of switch costs. It is important to investigate more cost-effective alternatives to interconnect architecture in order to ensure future HPC systems support the communication requirements of the broadest range of scientific disciplines. We believe that an adaptive interconnection architecture such as HFAST offers the best combination of flexibility and cost-scaling to meet these challenges. In future work we hope to expand the coverage of DOE applications for which we have high quality communication profiles and also examine the particular technological components from which one might bring an HFAST network into existence.

References

[1] IPM Homepage. `http://www.nersc.gov/projects/ipm`, 2005.

[2] Kevin J. Barker, Alan Benner, Ray Hoare, Adolfy Hoisie, Alex K. Jones, Darren J. Kerbyson, Dan Li, Rami Melhem, Ram Rajamony, Eugen Schenfeld, Shuyi Shao, Craig Stunkel, and Peter A. Walker. On the feasibility of optical circuit switching for high performance computing systems. In *Proc. SC2005: High performance computing, networking, and storage conference*, 2005.

[3] 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 on Research and Development*, 49, March/May 2005.

[4] Julian Borrill, Jonathan Carter, Leonid Oliker, David Skinner, and R. Biswas. Integrated performance monitoring of a cosmology application on leading hec platforms. In *Proceedings of the International Conference on Parallel Processing (ICPP), to appear*, 2005.

[5] Cactus Homepage. `http://www.cactuscode.org`, 2004.

[6] A. Canning, J. Carter, J. Shalf, and S. Ethier. Scientific computations on modern parallel vector systems. In *Proceedings of the IEEE Conference on Supercomputing*, 2004.

[7] Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory. `http://seesar.lbl.gov/CCSE`.

[8] Vipul Gupta and Eugen Schenfeld. Performance analysis of a synchronous, circuit-switched interconnection cached network. In *ICS '94: Proceedings of the 8th international conference on Supercomputing*, pages 246–255, New York, NY, USA, 1994. ACM Press.

[9] J.-F. Haas and B. Sturtevant. Interaction of weak shock waves with cylindrical and spherical gas inhomogeneities. *Journal of Fluid Mechanics*, 181:41–76, 1987.

[10] Z. Lin, S. Ethier, T.S. Hahm, and W.M. Tang. Size scaling of turbulent transport in magnetically confined plasmas. *Phys. Rev. Lett.*, 88, 2002.

[11] A. Macnab, G. Vahala, P. Pavlo, , L. Vahala, and M. Soe. Lattice boltzmann model for dissipative incompressible MHD. In *Proc. 28th EPS Conference on Controlled Fusion and Plasma Physics*, volume 25A, 2001.

[12] Metis Homepage. `http://glaros.dtc.umn.edu/gkhome/views/metis`.

[13] Burkhard Monien. The complexity of embedding graphs into binary trees. In L. Budach, editor, *Fundamentals of Computation Theory. Proceedings,*, pages 300–309, 1985.

[14] J. Qiang, M. Furman, and R. Ryne. A parallel particle-in-cell model for beam-beam interactions in high energy ring collide rs. *J. Comp. Phys.*, 198, 2004.

[15] C. A. Rendleman, V. E. Beckner, M. L., W. Y. Crutchfield, and John B. Bell. Parallelization of structured, hierarchical adaptive mesh refinement algorithms. *Computing and Visualization in Science*, 3(3):147–157, 2000.

[16] J. Shalf, S. Kamil, L. Oliker, and D. Skinner. Analyzing ultra-scale application communication requirements for a reconfigurable hybrid interconnect. In *Proc. SC2005: High performance computing, networking, and storage conference*, 2005.