Improving Data Movement Throughput by Balancing Network Load on the Blue Gene/Q Supercomputer

Huy Bui*, Preeti Malakar†, Eun-Sung Jung†, Venkatram Vishwanath†,
Todd S. Munson†, Sven Leyffer†, Andrew Johnson*, Jason Leigh‡, and Michael E. Papka†
*Dept. of Computer Sciennce
University of Illinois at Chicago, Chicago, IL
Emails: abui4, ajohnson@uic.edu
† Argonne National Laboratory, Lemont, IL
Emails: pmalakar@anl.gov, esjung@mcs.anl.gov, venkat, tmunson, leyffer, papka@anl.gov
‡Dept. of Computer Science
University of Hawaii at Manoa, Honolulu, HI
Email: leigh@hawaii.edu

***Abstract*—Achievable networking performance of applications in a supercomputer depends on the exact combination of communication patterns of the applications and routing algorithms used by the supercomputer. In order to achieve the highest networking performance for the applications, the routing algorithms need to be designed optimally for the communication patterns. However, while the communication patterns usually have a wide spectrum and vary from application to application and even from phase to phase in an application, the routing algorithms have a limited variation and usually are optimized for typical communication patters. This results in high networking performance for favored communication patterns but low networking performance for others. In this paper, we present approaches for improving networking performance by rebalancing load on physical links on the Blue Gene Q supercomputer. We realize our approaches into a framework called OPTIQ and demonstrate the efficacy of our framework via a set of benchmarks. The results show that we can achieve serveral times higher bandwidth than default mechannism MPI used in the Blue Gene Q supercomputer for certain communication patterns.**

***Keywords*-movement, load balancing, optimization, communication patterns**

I. Introduction

Simulation time in a supercomputer depends partially in achievable networking performance in the supercomputer. The achievable networking performance in its turn depends on the exact combination of communication patterns of applications and routing algorithms used by the supercomputer. For each communication pattern there exists routing algorithms resulting in high networking performance. However, while the communication patterns have wide spectrum and vary time to time, the routing algorithms have a limited variation and usually are optimized for typical communication patters. This results in high networking performance for favored communication patterns but low networking performance for other communication patterns. Low networking performance subsequentially leads to long simulation time. Reducing the communication time for these non-favored communication pattern reduces the running time of an application. Thus, improving the networking performance for these communication patterns are important in redecing simulation time.

Most of the routing algorithms are designed to perform well for certain communication patterns. However, for other communication patterns, they do not perform as well usually due to the unblanced on physically links caused by changing in the communication patterns. Rebalancing loads on the physical links in these situations can lead to better achievable performance.

The effect can be observed in MPI IO aggregation (give citation) and cesm. We show our work is motivated by those applications.

In this work, we propose a set of approaches that improve network performance for applications in supercomputers by rebalancing the load on physical links. Our approaches include using both heuristic algorithms and formal models with mathematical solvers to search for paths to move data from a set of sources to a set of destinations. The data then be divided into smaller messages and put into queues to move along found paths. The actual data movement can be done by using any available libraries for communication on the supercompter. We realize our approaches in a framework called OPTIQ. Our framework allows to easily expand the our work by adding algorithms for search paths, different way to schedule data transfer and different way to transfer data. It also allow to extend the framwork to other systems.

Our contributions include:

- Bring an insight understanding of how networking happens in the Blue Gene/Q supercomputer. By experiments, we show patterns that current routing algorithms favor and what it doesn not and explain why it happens it that way.
- Propose a set of approaches to improve networking

throughput by taking advantage of unused links or balancing network on links. We also explain when each approach should be used ot result highest possible throughputs.

Our paper includes the follows. In the next section, we present previous works in optimizing or improving data movement in various supercomputers or computing systems. In section III, we give a brief introduction to Mira - a Blue Gene Q supercomputer that we used to run our experiments. Section IV is the main section, where we explain about our framework and details of our approaches for each component of the framework. We demonstrate the efficacy of our approaches via a set of benchmarks in section VI. In section VII, we conclude our work and give some ideas about future work.

II. Related work

Improving data movement performance by balancing network load has been studied in previous works. In [1], Valiant proposed a randomized routing mechanism mathematically proved to be able to route data globally with no sharing links for what network. However the routing mechanism did not work well in local routing [2] . [3] proposed minimal routing optimization. However, it did not work well with global optimization [2] . To address the limitation of both approaches, [2] proposed GOAL globally oblivious adaptive locally to balance load using adaptive routing algorithm for torus networks.

GOAL paper [2] Several bgq paper

[4] proposed oblivious routing schemes in extended generalized fat tree networks. It extended 2 algorithms called S-mod-k and D-mod-k to provide a better oblivious solution for slimmed networks.

[5] showed that there is potential to optimize all-to-all collective exchange communication pattern at system level in fat tree networks and proved it mathematically and via simulations. The work is then extended in [6] to propose a generic method to determine optimal pattern-specific routing for eXtended Generalized Fat Tree (XGFT). The method used a hybrid combination of integer linear programming (ILP) and dynamic programming. Interconnection network is divided into small subdomains and ILP is used for each subdomain. The local solution is then combined using dynamic programming. The proposed method takes up to several hours for several thousands of compute nodes interconnect network.

In other supercomputers, [7] proposed two deadlock-free routing mechanisms that support o n-the-fly adaptive routing on Cray XC30 system for Dragonfly networks.

In Blue Gene Q, [8] proposed a heuristic routing for Blue Gene/Q supercomputing systems. The routing path is computed dynamically at the routing time based on coordinates of source and destination, partition shape and message size. The systems route a packet along the longest dimension of a partition first, shortest last. BG/Q systems route a message using single path. Different messages however might be routed using different paths.

Some optimizations are done based on systems specific characteristics such as [9] proposed optimization for MPI_Allreduce in BG/Q based on the observation on number links and other hardware supports on compute nodes.

Our previous works in [10] and [11] addressed a problem of data aggregation for I/O purposes very dense in I/O case. In these works, we proposed . In a more recent work [12] we presented our approaches for improving the performance very sparse. In this paper, we give solution for medium dense communication patterns between M and N.

In the next section, we briefly introduce Mira, a Blue Gene Q supercomputer that we used in our experiments in demonstrating the efficacy of our approaches.

III. Experiment system

In this section, we describe Mira - an IBM Glue Gene Q supercomputer in which we developed our algorithms and a framework for network load balancing and conducted our experiments. The Argonne Leadership Computing Facility (ALCF) maintains several compute-analysis systems used by the scientific community. Figure 1 depicts the architecture of the primary ALCF resources, consisting of the Blue Gene/Q compute cluster (Mira), the data analysis cluster (Tukey), and the file server nodes.

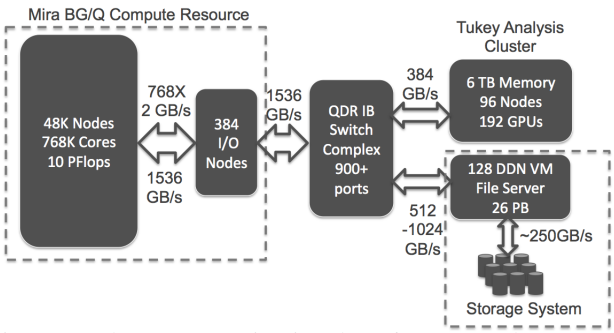


Figure 1: The ALCF maintains the 768K core Blue Gene/Q compute cluster (Mira), data analysis cluster (Tukey), and file server nodes.

Mira [8], with 48 compute racks (48K nodes and 768K cores) at the ALCF, provides 10 PFlops theoretical peak performance. Each node has a 16-core processor and 16 GB of memory.

The interprocess communications of Blue Gene/Q travel on a 5D torus network both for point-to-point and for collective communications. This 5D torus interconnects a compute node with its 10 neighbors at 2 GB/s theoretical peak over each link in each direction, making a total of 40 GB/s bandwidth in both directions for a single compute node. Because of packet and protocol overheads, however, only up to 90% of the raw data rate (1.8 GB/s) is available

for user data. The machine can be partitioned into non-overlapping rectangular submachines; these submachines do not interfere with each other except for I/O nodes and the corresponding storage system.

For interconnect network traffic, BG/Q supports both deterministic and dynamic routing [8]. In deterministic routing, packets are routed based on dimension-ordered routing, from the longest first to the shortest last. In dynamic routing, routing is still dimension-ordered, but it is programmable, enabling different routing algorithms to be used. Given the size of a certain message, routing is always the same, and its path is known before it is routed. These are the default routing algorithms and cannot be changed during run time. However, one can set which routing zone id to use by using the PAMI_ROUTING environment variable. Since BG/Q uses single-path data routing, for sending/receiving a message only one link of the ten available is used. The details of routing can be found in [8].

PAMI is a low-level communication library for BG/Q [13]. PAMI provides low-overhead communication by using various techniques such as accelerating communication using threads, scalable atomic primitives, and lockless algorithms to increase the messaging rate. Since MPI is implemented on top of PAMI, direct use of PAMI would provide higher messaging rates as well as lower latencies in comparison with MPI.

IV. Multi-path Data Movement/Routing

In Blue Gene/Q, data is routed through its interconnect using the default routing algorithms. The default routing algorithms perform well for some communication patterns [8]. However, for certain communication patterns (shown later in this paper), they result in poor performance due to unbalanced load on the physical network links. This results in significantly larger amount of data being transferred over few links. This is because the default algorithms use a single path to transfer data between any two nodes in the system. In addition, data traverses along fixed paths on certain links using static routing regardless of the overall load of the system. Thus, some links are overloaded while other links have less data or may even be idle. This overloading is a major bottleneck to the data movement throughput. Balancing the load on the physical network links can improve the data transfer throughput.

The above problem can be formulated as a multicommodity integral flow problem, which is shown to be NP-complete [14]. Given a set of source and destination nodes, and the amount of data to be transferred from the sources to destinations, the problem is to find a set of paths from the source nodes to the destination nodes that results in high throughput. Additionally, the objective is to balance the overall system load in order to avoid congestion in the interconnect and to avoid overloading physical network links. In this paper (Sections IV-A and IV-B), we propose two approaches to solve this problem taking into consideration the system topology.

We use the idle or lightly-loaded links for data transfer in order to balance the load. For this, we need to search for multiple paths between source and destination nodes and assign appropriate load on each path. Present-day supercomputers have thousands of nodes and hundreds of thousands of edges due to complex interconnect topology. This implies a huge search space for multiple paths. Brute-force approach of searching for paths can lead to significant amount of time being spent on searching for paths, calculating and balancing load on the links. We prune the search space to reduce the search time by constraining the number of hops on each path.

In this paper, we propose two approaches aiming for balancing load on physical links: one heuristic algorithms and one model-based optimization approach. In both approaches, we use Yen's algorithm [15] to search for a set of shortest paths. After that, in the first approach, we use maximum load path constraint to prune the search space, while in the second approach, we use an optimization model with solvers to search for final paths. In this paper, we use Yen's algorithm, but any algorithms searches for K shortest paths should work as well.

In order to search for paths, we model the interconnect network as a graph. Each compute node is modeled as a vertex and each physical link is modeled as an edge. The bandwidth of a physical link is modeled as its corresponding edge's capacity. The need of data movement from source nodes to destination nodes is modeled as data movement from source vertices to destination vertices. The problem now becomes searching for paths to move data from source vertices to destination vertices to minimize transfer time. The next subsection, we briefly describe the K shortest paths generation based on Yen's algorithm.

K Shortest Path Generation

Algorithm 1 K shortest paths generation

Input: Set of pairs of source-destination (s_i, d_i). Graph of nodes. Number of shortest path k

Output: Set of paths: k paths for a pair of source-destination

```
for each pair of source-dest (s_i, s_i) do
    while (less than k paths discovered || still have paths to discover) do
        Use Yen's algorithm to search for the shortest path p.
        Add p into k_paths for later use.
    end while
end for
```

In this algorithm, we go through all pairs of sources and destinations and generate k shortest paths for each pair. We

do stop when either there is no more path to generate or we have enough k paths.

A. Heuristic Approach

Given the set of k shortest paths *k_paths* for each pair of communication, we need to select paths to be used for data movement to result in high performance. In this approach, we assume that each path carries similar amount of data. Thus, number of paths using a link can represent data load of the link. In order to avoid overloading on physical links and achieve high performance, we select paths in such a way that satisfies 2 conditions: (1) the number of paths for any pair of source/destination is as many as possible and (2) the maximum number of paths using any physical link is less than a given *maxload* value. While the first condition gives us more choices of paths, the second condition gives us control over max load on physical links. One exhaustive approach is to go through all combinations of paths for all pairs to examine the 2 conditions to find the best combination. However, its time complexity is exponential. We propose a heuristic algorithm that can produce the set of paths while examining much less combinations. Our approach keeps iterating through all pairs of source/destination to search for more paths to use until the maximum of loads on links reach to the *maxload* value. The approach is presented in **Algorithm** 2. The detail of the algorithm is explained as follows.

In the heuristic algorithm, inputs of the algorithm include pairs of source and destination together with their k shortest paths *k_paths* and an allowed *maxload*. Output of the approach is a set of selected paths used for data movement. We maintain a table *load*[][] of loads on all physical links, whenever a link (u,v) is used by a selected path its entry in the load table i.e. *load*[u][v] is increased by 1. We save selected path in a queue *selected_paths*.

We iterate through all pairs of source/destination. For each pair of source/destination, we get the first path out of its set of *k_paths*. We check if adding the path to its *selected_paths* would make the maximum load in the *load* table over *maxload*. We check the load by going through all links used by the path and examine their current loads in the *load* table in comparison with *maxload*. If their current load adding by 1 is not over *maxload* we then add the path *p* into *selected_paths*, update the *load* table and remove the path from *k_paths*. We iterate through all pairs and add at most one path per pair at a time. The algorithm completes when we either running out of paths to add or the maximum load of physical links in *load* table is over *maxload*.

In the algorithm, we also check if under the current *maxload* a pair of source/destination is not able to add any paths for transferring data. If so, we increase the *maxload* by 1 until we can add a path to the pair. We do so to make sure that we have at least 1 path to transfer data from its source to its destination.

Algorithm 2 Heuristic Algorithm based on k shortest paths

Input: Set of pairs of source-destination (s_i, d_i) and their *k_paths*. *maxload*
Output: Set of paths: *selected_paths* for data movement
Init:

queue<struct path> *selected_paths*;
int[][] loads: loads of all physical links, init to 0s.

Main:

```
function Heuristic_search
    while !(run out of paths —— go over maxload) do
        for each pair of source-dest (s_i, s_i) do
            Get the first path p out of set k_paths.
            Check if adding p make the current load over maxload.
            if (Not over maxload) then
                remove p from k_paths.
                add p into selected_paths.
                update load[][] with links used by p.
            else
                check if there is at the current pair has at least one path.
                if (Not having any path) then
                    increase the maxload by 1
                else
                    remove p from k_paths.
                end if
            end if
        end for
    end while
end function
```

In the **Agorithm** 2 we aim on balancing the number of paths using physical link. We identify number of paths on a link to actual amount of data transferred on the link. Thus, the data movement performs well if each path carries similar amount of data. In real applications, it is not always the situation. Due to different data sizes and different number of paths, data size per path can vary. In order to gain better performance, we need a better way to determine the amount of data to be transferred on each path. In the next section, we propose another approach that employs a mathematical model and solvers to in determining amount of data to be transferred on each path.

B. Model Optimization Approach

In this approach, we need to search for an assignment of data amount for each path that result in highest performance of data movement. Inputs are a set of pairs of source and destination, for each pair we have an amount of data needed to transfer from the source to the destination and k shortest paths between the source and destination.

We model the problem as searching for flow values in order to minimize transfer time for multiple commodities using pre-determined multiple flows in a network given the amount of comodities and the capacities in links of the network. The model is written in AMPL (A Mathematical Programming Language). The model is descrbied in **Model** 1.

Model 1 Data movement optimization

```
set Nodes;
set Arcs within Nodes cross Nodes;

set Jobs;
set Paths{Jobs};
set Path_Arcs{job in Jobs, p in Paths[job]}
    within Arcs;

param Capacity{Arcs} >= 0 default Infinity;
param Demand {Jobs} default 0;

var Flow {job in Jobs, Paths[job]} >= 0;
var Z >= 0;

maximize obj: Z;

subject to

demand_con {job in Jobs}: sum {p in Paths[job]}
Flow[job,p] = Demand[job]*Z;

capacity_con {(i,j) in Arcs}:
  sum {job in Jobs, p in Paths[job]:
    (i,j) in = Path_Arcs[job,p]} Flow[job,p]
<= Capacity[i,j];
```

The notions used in **Model** 1 are explained as follows:

- sets:
  - *Nodes*: set of nodes in the network, each node represent a compute node in the supercomputer.
  - *Arcs*: set of arcs in the network. Each arc represent a physical link in the supercomputer.
  - *Jobs*: set of jobs. Each jobs has a source and a destination.
  - *Paths*: set of paths for each job.
  - *Path_Arcs*: set of arcs on each path of each job.
- params:
  - *Capacity*: capacity of each arc i.e. bandwidth of the physical link.
  - *Demand*: amount of data to be transferred of each job between a pair of source and destination.
- vars:
  - *Flow*: flow of each job on a path. It can be seen as the proprotional bandwidth assigned for the job on that path.
  - *Z*: is reversed of total time.
- objective function: we want to minimize the time or maximize its reversed value i.e. maximize *Z*.
- constraints(subject to):
  - *demand_con*: flow of a job on equals to the demand of the job divided by the transfer time.
  - *capacity_con*: total flow on an arc is less than its capacity.

The model takes a set of nodes, a set of arcs and their corresponding capacity, a set of jobs (source/destination pairs), a demand for each job, a set of paths for each job, and a set of arcs for each path as inputs. It searches for an assignment of flow values (proportional capacity) for paths of all the jobs such that the transfer time for demands of all jobs is minimum.

We feed the model into solvers together with data of nodes, arcs, capacity, paths for jobs and get the paths with given proportional bandwidth. Based on proportional bandwidth, each path can take proportional demand of a job.

We realize algoirthms and other implementation in a framework named OPTIQ.

V. Framework

Our framework has 3 main components: Path searching algorithms, Schedule and Transport and an extra component depicted in Figure 2.

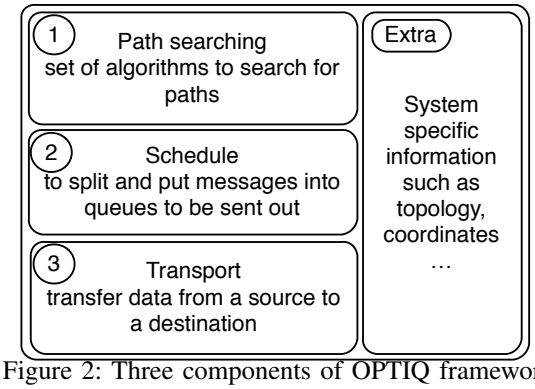


Figure 2: Three components of OPTIQ framework

The functionarity of each component is as following:

- Path searching: search for path to transfer data from a set of sources to a set of destination. Multiple or single paths can be found using a set of algorithm. User can decide what algorithm to be used or let the framework use a default algorithm.
- Schedule: Split a buffer data that needed to tranfer into smaller messages and put those messages into a queue of transport layer to be transferred. It also handles incoming messages for itself and for forwardig them to its neighbors on a way to a message final destination. As we route data in our own ways, we search for the paths, and we also need to schedule messages transfer. It includes sending local messages, forwarding messages form other sources, receiving data as the intermediate node or the destination node.
  Order of messages into sending queue: 3 types of messages: local messages (needed to send), fowarding

messages (needed to send), its receiving messages. first come first serve, local messages first. forwa
When there are multiple ranks per node, which one will be choosen to receive data at the next dest (forwarding). Single rank to do or many rank to do, currently every rank executes data transfer.

- Transport: actually transfer an amount of data from one point to another point in the system.
- Extra component: To get system specific information such as partition size, topology, coordinates, torus, and to compute neighbors of available nodes given to an application. Topology reading, coord, neighbors, torus, size, routing order, graph generated. Also set of benchmarks, tests.
  The framework has various options to allow users to tune the framework for optimal performance. For example, the framework allows to select messages from queue to either forward a message from another node first or to send its own message first, to select algorith to search for paths or to set chunk size to transfer a message, to easily add new transport layers on different machine.

VI. Exeperiments and Results

In this section, we demonstrate the efficacy of our approaches via a set of benchmarks.

A. Experiment setup

Wne carried our experiments on Mira, a Blue Gene/Q supercomputer. In our experiments, we varied paritition size from 512 nodes up to 8012 nodes. The experiments involved a subset or entire set of nodes for each partition. The number sources and destinations and the distance between them are also varied depending on each experiment. We also varied the number of ranks from 1 rank per node up to 8 ranks per node to show the efficacy of our work if multiple ranks per node are used. Data size to be exchanged is also varied from 512 KB up to 8 MB per pair of communication. Our experiments covered 3 communication patterns: disjoint, overlap and subset. For the commnication patterns, we demonstrated the efficacy of our algorithms in comparions with MPI_Alltoallv.

B. MPI Paths Reconstruction

In our experiment, we need to measure not only the performance of MPI routines but also loads on physical links and number of hops of path that MPI takes to move data from a set of sources to a set of destination. The load and hops informaiton can reveal insight of performance difference between MPI and our framework OPTIQ. Thus reconstructing MPI's paths is necessary to get load and hops information.

We reconstruct MPI's paths based on our understanding of default routing algorithms described in [8]. For each pair of source and destination, we start at a source node and follow the rules of the routing algorithm to move data from the source node to its destination. We then record paths for all the pairs and use them to calculate load and number of hops.

C. Communication Patterns

In this paper we demostrate data movement performance of our OPTIQ framework and existing MPI's routines on the following communication patters:

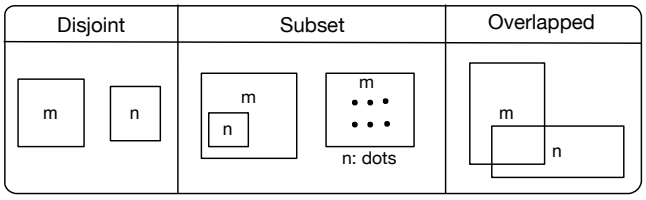


Figure 3: Communication patterns

- Subgroup Data Aggregation: a special case of All to many (or many to many.)
- Many to many: disjoint sets.
- Many to many: overlapped sets.
- Many to many: subsets. In this category, there are 2 sub types: (Type 1) concetrated subset of destinations and (Type 2) distributed destination nodes as in I/O data aggregation.

We carried a set of experiments to study the system's behavior in various patterns and demontrate throughput improvement.

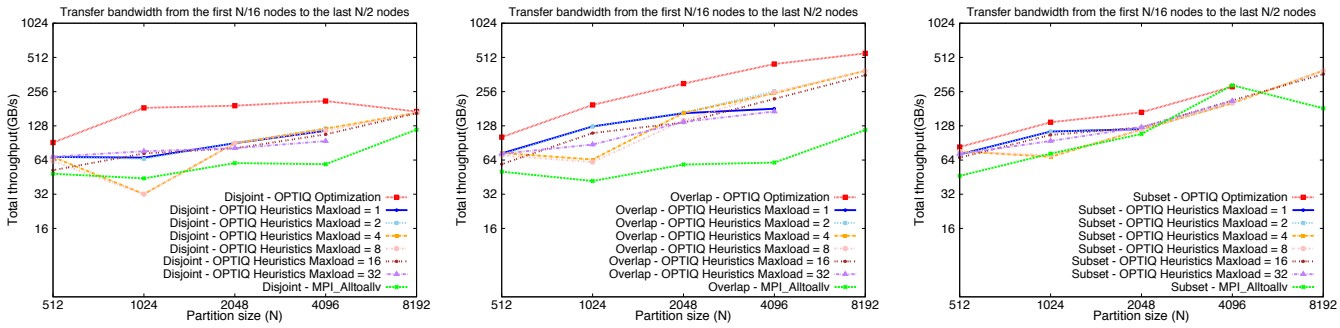
D. Experimental results

In the experiments, we collected the throughput and loading related information while varying communication patterns, partition sizes, message sizes, number of MPI ranks per node. In the next subsections, we go into each communication and study in details the system's behaviors and performance.

The number of sources: m, number of destination n, ratio between m and n is r, total number of node is p, average distance between sources and destinations is d

1) Constant ration r, while varying m and n together with the number of nodes p: In this experiment, we keep the ratio between number of sources and destination constant while varying the number of sources, destinations together with total number of nodes. We keep the ration as 1/8. The first m = p/16 nodes send data to the last p/2 nodes. Each source has 8 destination We tested the framework for 3 patterns: subset, disjoint and overlap with 3 methods of transferring data OPTIQ optimization, OPTIQ heuristics and MPI_Alltoallv. We use 1 rank/node. The data size is 8 MB per pair. We vary the number of nodes from 512 to 8192.

In this experiment, we keep the number sources and destinations as 128 to 256 while varying the total number



(a) Disjoint (b) Overlap (c) Subset

Figure 4: Varying the number of sources and destinations while keeping the ratio constant

of nodes. The sources are first 128 nodes while destination are last 256. Each source node has 2 destination nodes. We use 1 MPI rank per node. The total number of nodes vary from 512 to 8K. The message size is 8 MB.

2) Consant the number of node p and sources m, vary number of destinations n: This is to show that with more number of destination, we have less paths to explore

3) Constant the number of nodes p, sources m and destination n, vary the average distance d between sources and destinations: This is to show that with longer distance, we have more paths to explore.

4) Benchmark for chunk size: To transfer dat through intermediate, we split a message into smaller chunks and keep sending the chunks into the network. It is to reduce the waiting time at the intermediate, thus, reduce the total transfer time. We do an experiment to see what is the optimal chunk size for each pattern. In this experiment, we generated 91 patterns (16 disjont patterns[expriment id 0-15], 48 overlap patterns [16-63], 27 subset patterns [64-90]), in which we vary the number of sources, destination and their locaiton in the 2048-node partition. We varied the chunk size from 8 KB to 1 MB to find the most optimal chunk size. However for the sake of simplicity, we only show the results of chunk sizes from 32 KB to 512 KB while not losing the accuracy of the experiment. The results are shown in Figure 5.

We can see that, for disjoint and overlap patterns, most of the time the chunk size of 64 KB and/or 128 have dominated performance. For the subset patterns, chunk size 128 KB is clearly optimal choice. In order to be consistent, in the above experiments, we used 64 KB chunk size for disjoint and overlap pattern and 128 KB chunk size for subset patterns.

5) Message size scaling:

6) Number of ranks scaling:

7) Number of paths fed into heuristics II: Quality i.e the performance based on the number of paths

Time to search for paths

8) Number of paths fed into model: Quality

Time to search for paths

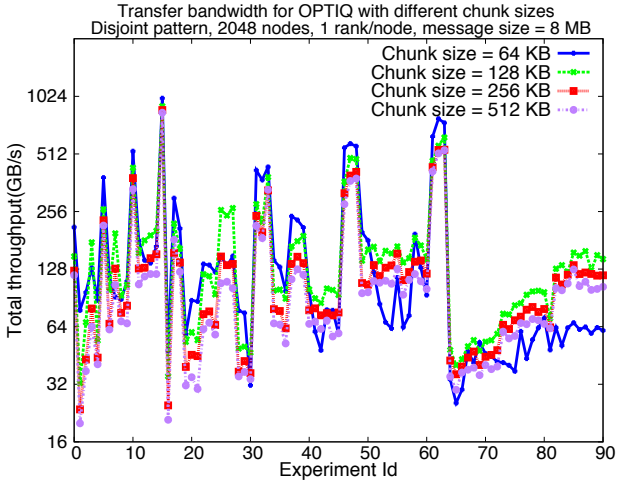


Figure 5: Chunk sizes and their performance in 2048-node partition

VII. Conclusion

In this paper we have ... expand the work to the Cray XE6 supercomputer and tested in real applications

By understanding better communication and rounting - ¿ better design routing in future system, optimizing for application.

Acknowledgment

The authors would like to thank... more thanks here

References

[1] L. G. Valiant and G. J. Brebner, "Universal Schemes for Parallel Communication," in *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '81. New York, NY, USA: ACM, 1981, pp. 263–277. [Online]. Available: http://doi.acm.org/10.1145/800076.802479

[2] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles, "GOAL: a load-balanced adaptive routing algorithm for torus networks," *ACM SIGARCH Computer Architecture News*, vol. 31, no. 2, pp. 194–205, 2003.

[3] G. D. Pifarré, L. Gravano, S. A. Felperin, and J. L. C. Sanz, "Fully-adaptive Minimal Deadlock-free Packet Routing in Hypercubes, Meshes, and Other Networks," in *Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '91. New York, NY, USA: ACM, 1991, pp. 278–290. [Online]. Available: http://doi.acm.org/10.1145/113379.113405

[4] G. Rodriguez, C. Minkenberg, R. Beivide, R. P. Luijten, J. Labarta, and M. Valero, "Oblivious routing schemes in extended generalized Fat Tree networks," in *CLUSTER*. IEEE, 2009, pp. 1–8.

[5] B. Prisacari, G. Rodriguez, C. Minkenberg, and T. Hoefler, "Bandwidth-optimal All-to-all Exchanges in Fat Tree Networks," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ser. ICS '13. New York, NY, USA: ACM, 2013, pp. 139–148. [Online]. Available: http://doi.acm.org/10.1145/2464996.2465434

[6] ——, "Fast Pattern-specific Routing for Fat Tree Networks," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, pp. 36:1–36:25, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2555289.2555293

[7] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, and M. Valero, "Efficient Routing Mechanisms for Dragonfly Networks," in *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 2013, pp. 582–592.

[8] D. Chen, N. Eisley, P. Heidelberger, S. Kumar, A. Mamidala, F. Petrini, R. Senger, Y. Sugawara, R. Walkup, B. Steinmacher-Burow, A. Choudhury, Y. Sabharwal, S. Singhal, and J. J. Parker, "Looking under the hood of the IBM blue gene/Q network," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 69:1–69:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389090

[9] S. Kumar and D. Faraj, "Optimization of MPI_Allreduce on the Blue Gene/Q Supercomputer," in *Proceedings of the 20th European MPI Users' Group Meeting*, ser. EuroMPI '13. New York, NY, USA: ACM, 2013, pp. 97–103. [Online]. Available: http://doi.acm.org/10.1145/2488551.2488557

[10] V. Vishwanath, M. Hereld, V. Morozov, and M. E. Papka, "Topology-aware data movement and staging for i/o acceleration on blue gene/p supercomputing systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 19:1–19:11. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063409

[11] H. Bui, E. Jung, V. Vishwanath, J. Leigh, and M. Papka, "Improving data movement performance for sparse data patterns on blue gene/q supercomputer," in *7th International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2) held in conjunction with the 43rd International Conference on Parallel Processing*, September 2014.

[12] H. Bui, H. Finkel, V. Vishwanath, S. Habib, K. Heitmann, J. Leigh, M. Papka, and K. Harms, "Scalable parallel i/o on a blue gene/q supercomputer using compression, topology-aware data aggregation, and subfiling," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, Feb 2014, pp. 107–111.

[13] S. Kumar, A. R. Mamidala, D. A. Faraj, B. Smith, M. Blocksome, B. Cernohous, D. Miller, J. Parker, J. Ratterman, P. Heidelberger, D. Chen, and B. Steinmacher-Burrow, "PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*, ser. IPDPS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 763–773. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2012.73

[14] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Foundations of Computer Science, 1975., 16th Annual Symposium on*. IEEE, 1975, pp. 184–193.

[15] J. Y. Yen, "An algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Quart. Applied Math*, vol. 27, pp. 526–530, 1970.