Optimal Routing in Binomial Graph Networks

Thara Angskun [1], George Bosilca [1], Brad Vander Zanden [1], and Jack Dongarra [2]

[1]Department of Computer Science, The University of Tennessee, Knoxville
[2]University of Tennessee, Oak Ridge National Laboratory and University of Manchester
{angskun, bosilca, bvz, dongarra}@cs.utk.edu

Abstract

A circulant graph with n nodes and jumps $j_1, j_2, ..., j_m$ is a graph in which each node i, $0 \leq i \leq n-1$, is adjacent to all the vertices $i \pm j_k$ mod n, where $1 \leq k \leq m$. A binomial graph network (BMG) is a circulant graph where j_k is the power of 2 that is less than or equal to n. This paper presents an optimal (shortest path) two-terminal routing algorithm for BMG networks. This algorithm uses only the destination address to determine the next hop in order to stay on the shortest path. Unlike the original algorithms, it does not require extra space for routing tables or additional information in the packet. The experimental results show that the new optimal algorithm is significantly faster than the original optimal algorithm.

1 Introduction

Recently, several high performance computing platforms have been installed with more than 10,000 CPUs, such as Blue-Gene/L at LLNL, BGW at IBM and Columbia at NASA [1]. However, as the number of components increases, so does the probability of failure. To satisfy the requirements of such a dynamic environment (where the available number of resources is fluctuating), a scalable and fault-tolerant communication framework is needed. The communication framework is important for both runtime environments of MPI libraries and the MPI libraries themselves. In general, the communication framework is based on a logical network topology.

There are several existing logical network topologies that can be used in high performance computing (HPC). The fully connected topology is good in terms of fault-tolerance, but it is not scalable because of its high degree. The bidirectional ring topology is more scalable, but it is not fault-tolerant. Hypercube [2] and its variants [3–10], FPCN [11], de Bruijn [12] and its variants [13, 14], Kautz [15] and ShuffleNet [16] have a number of node restrictions. They are either not scalable or not fault-tolerant. The Manhattan Street Network (2D Torus) [17] is more flexible (no restriction in numbers of node) than Hypercube-like topologies. However, it has a much higher average hop-distance. Variants of k-ary tree, such as Hierarchical Clique (HiC) [18] and k-ary sibling tree (Hypertree [19]) used in SFTP [20, 21], are scalable and fault-tolerant. They are good for both unicast and broadcast messages. However, all nodes in their topologies are not equal (the resulting graph is not regular). Topologies, used in structured peer-to-peer networking based on distributed hash tables such as CAN [22], Chord [23], SkipNet [24], Kademlia [25], Viceroy [26], Pastry [27] and Tapestry [28], are also scalable and fault-tolerant. They were designed for resource discovery in highly dynamic environments. Hence they may not be efficiently used in HPC, owing to the overhead for managing highly dynamic applications.

Binomial graph (BMG) [29] provides desirable topological properties in terms of both scalability and fault-tolerance for high performance computing such as reasonable degree, regular graph (every node has the same degree), low diameter, symmetric graph (in the sense that an average inter-nodal distance is the same from any source node) and low cost factor. It also has low message traffic density, optimal connectivity, low fault-diameter, is strongly resilient and has good optimal probability in failure cases.

BMG is an undirected graph $G{:=}(V,E)$ where V is a set of nodes (vertices); $|V| = n$; and E is a set of links (edges). Each node i, where $i \in V$ and i=0,1,...,*n-1*, has links to a set of nodes U, where $U=\{i\pm1, i\pm2, ..., \pm2^k | 2^k \leq n\}$ in circular space, i.e., node i has links to a set of clockwise (CW) nodes $\{(i+1) \bmod n, (i+2) \bmod n, ..., (i+2^k) \bmod n \mid 2^k \leq n\}$ and a set of counterclockwise (CCW) nodes $\{(n+i-1) \bmod n, (n+i-2) \bmod n, ..., (n+i-2^k) \bmod n \mid 2^k \leq n\}$. The structure of BMG can also be classified in the Circulant graph family[1]. A Circulant graph with n nodes and jumps $j_1, j_2, ..., j_m$, where $m \in \mathbb{N}$, is a graph

[1]The family of Circulant graphs includes fully connected, ring, Recursive Circulants [30] and Midimew [31].

in which each node i, $0 \leq i \leq n-1$, is adjacent to all the vertices $i \pm j_k \bmod n$, where $1 \leq k \leq m$. BMG is a Circulant graph where j_k is the power of 2 that is less than or equal to n. For a BMG of size n (having n nodes), each node has a degree δ (number of neighbors) as shown in Equation (1).

$$\delta = \begin{cases} (2 \times \lceil \log_2 n \rceil) - 1 & \text{For } n = 2^k, \text{where } k \in \mathbb{N} \\ (2 \times \lceil \log_2 n \rceil) - 2 & \text{For } n = 2^k + 2^j, \\ & \text{where } k, j \in \mathbb{N} \wedge k \neq j \\ 2 \times \lceil \log_2 n \rceil & \text{Otherwise} \end{cases} \tag{1}$$

Figure 1(a) illustrates an example of a 12-node binomial graph, where the thin lines represent all the connections in the network. The other way to look at the binomial graph is that it is a topology, which is constructed from merging all necessary links being able to create binomial trees from each node in the graph. Figure 1(b) shows an example of a binomial tree where node 0 is the root node. The arrows point in the direction of the leaf nodes. Obviously, BMG is able to deliver broadcast messages optimally from any node within $\lceil \log_2(n) \rceil$ steps because of its capability to create the binomial tree out of each node.

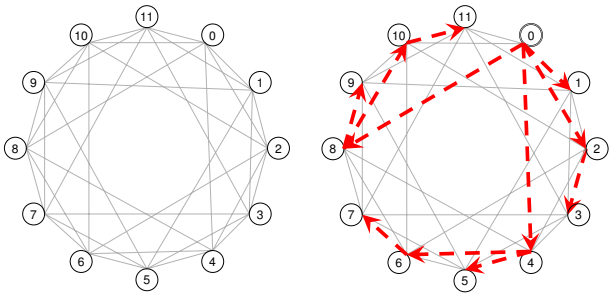


Figure 1. Binomial graph structure. (a) 12-node BMG. (b) Binomial tree from node 0.

The distance $d(x,y)$ between node x and node y in a graph is defined as the length of the shortest path from x to y in the graph. The diameter D of a graph is given by $\max(d(x,y))$ over all possible pairs (x,y) of nodes in the graph. The diameter D is the longest of the shortest paths between any two nodes in the graph. The average distance $\bar{d}$ of a graph is given by Equation (2).

$$\bar{d} = \frac{\sum_{x=0}^{n-1} \sum_{y=0}^{n-1} d(x,y)}{n \times (n-1)}, \text{ where } x \neq y \tag{2}$$

The average distance and diameter of BMG, along with related network topologies, are shown in Figure 2(a) and Figure 2(b), respectively. The results indicate that BMG has the lowest average distance ($\approx \frac{\log_2(n)}{3}$) and diameter ($O(\lceil \frac{\lceil \log_2(n) \rceil}{2} \rceil)$) among them.

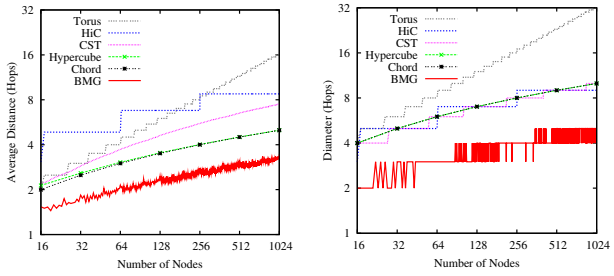


Figure 2. Distance Comparison. (a) Average distance ($\bar{d}$). (b) Diameter (D).

This paper introduces an optimal (shortest path) two-terminal routing algorithm for BMG networks. This algorithm uses only the destination address to determine the next hop in order to stay on the shortest path. Unlike the original algorithm, it does not require extra space for routing tables or additional information in the packet. In fact, the shortest-path problem for Circulant graphs of arbitrary degree is NP-hard [32]. There are several optimal algorithms for specific types of Circulant networks such as 2-Circulant [33, 34] and Recursive Circulant Networks [35]. However, these algorithms can not be directly applied with the BMG, due to their restriction with constant degree. Actually, BMG and the undirected Chord [36] are exactly the same topology when the number of nodes is a power of two[2]. Hence, the optimal routing in the undirected Chord [36] may also be used in BMG. Figure 3(a) illustrates an average distance ($\bar{d}$) overhead (in percent) of an undirected chord routing algorithm. The percent overhead was calculated from $\frac{\bar{d}_{\text{Chord}} - \bar{d}_{\text{Optimal}}}{\bar{d}_{\text{Optimal}}} \times 100$.

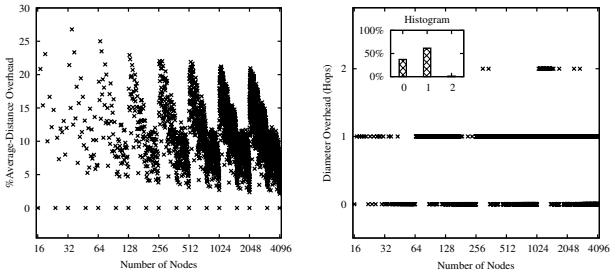


Figure 3. Performance of undirected chord routing algorithm on BMG topology. (a) $\bar{d}$ overhead (%). (b) D overhead (hops).

This figure indicates that an undirected chord rout-

[2]if the number of nodes in the undirected Chord is not power of two, it will create roughly the same small groups of nodes such that the number of groups is a power of two and will use their routing algorithm [36] between those groups.

ing algorithm is not optimal for BMG unless the number of nodes is a power of 2 or middle points between the power of 2 (e.g., 24, 48, 96, ...). Figure 3(b) illustrates diameter (D) overhead in the number of hops (i.e., $D_{\text{Chord}} - D_{\text{Optimal}}$).

The structure of this paper is as follows. Section 2 describes the original routing algorithm. The new optimal routing algorithms are discussed in section 3. Section 4 presents the experimental evaluations, followed by conclusions and future work in section 5.

2 Original routing algorithms

This section presents three original two-terminal routing algorithms [29]. One optimal routing algorithm is based on breadth-first search, and the two sub-optimal routing algorithms are called basic and variant. Each node in the graph may run the same routing algorithm because all nodes in BMG are equal (both regular and symmetric).

2.1 Breadth-First Search Optimal Algorithm

The optimal routing algorithm can use a breadth-first search technique with a modified graph coloring algorithm. Although this algorithm gives the optimal result, the complexity of the algorithm is $O(\delta^D)$.

Instead of recomputing the next hop in every message transmission, the breadth-first search technique can use a routing table to keep the result of the next hop sorted by the shortest path from the node itself to all other nodes in BMG. However, the routing table requires an extra space of $O(n^2)$.

2.2 Basic Sub-Optimal Algorithm

A basic algorithm to estimate the shortest path between nodes is to use a rule-based method that sends the unicast messages to a neighbor that has the closest ID to the destination ID as shown in Algorithm 2. The complexity of the basic unicast routing algorithm is $O(\delta)$. The experimental results indicate that the basic is a sub-optimal algorithm (i.e., average distance overhead $\neq 0$) as shown in Figure 4.

2.3 Variant Sub-Optimal Algorithm

This algorithm is the variant of the basic algorithm that allows messages to go forward to a neighbor of which ID is not the closest ID to the destination ID if the destination is directly connected to the neighbor. The complexity of the variant unicast routing algorithm is $O(\delta^2)$.

Figure 4(a) and Figure 4(b) present the overhead of both sub-optimal algorithms. They depict that the variant algorithm is marginally better than the basic algorithm in terms of $\bar{d}$ and D.

Algorithm 1 Finding an Optimal Route with Breadth-First Search

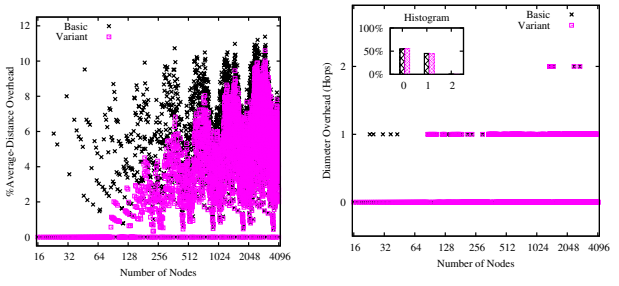
Require: $1 \leq myID \leq n \wedge 1 \leq destID \leq n, n \in \mathbb{N}$

```
for i = 0 to n do
    State[i] ⇐ INIT
end for
State[myID] ⇐ START
enQueue(myID)
while Queue is not empty do
    nodeID ⇐ deQueue()
    if nodeID = destID then
        break
    end if
    Get neighborID of the nodeID
    for i = 0 to (Numbersofneighbor) − 1 do
        if State[nodeID] = INIT then
            State[neighborID[i]] ⇐ START
            Parent[neighborID[i]] ⇐ nodeID
            enQueue(neighborID[i])
        end if
    end for
    State[nodeID] ⇐ DONE
end while
nodeID ⇐ destID
while Parent[nodeID] ≠ myID do
    nodeID ⇐ Parent[nodeID]
end while
Return nodeID
```

Algorithm 2 Find neighborID which has the estimated shortest distance to destID

Require: $1 \leq myID \leq n \wedge 1 \leq destID \leq n, n \in \mathbb{N}$

```
Min ⇐ ∞
Get neighborID of myID
for i = 0 to (Numbersofneighbor) − 1 do
    Distance ⇐ |destID − neighborID[i]|
    if Distance < Min then
        Min ⇐ Distance
        nextHopID=neighborID[i]
    end if
end for
Return nextHopID
```



(a) (b)

Figure 4. Sub-optimal routing performance. (a) $\bar{d}$ overhead (%). (b) D overhead (hops).

3 New routing algorithms

In order to always stay on the shortest path from a source to a destination, messages must be delivered through a neighbor that has the estimated *shortest hop* to the destination unlike the original basic sub-optimal algorithm that estimates the *shortest distance* between neighbors and destination (i.e., a greedy algorithm). The key to success of this algorithm is how well we can estimate the number of hops that is used for sending messages between two nodes. Several methods to calculate the number of hops between two nodes have been explored as follows.

3.1 Bit Counting method

The bit counting method represents the distance between a source and a destination in a binary format. The bit-1 represents the number of hops that messages can travel, e.g., if a distance between a source and a destination ($|destID - srcID$) is 9 (binary is 1001), a message is forwarded to nodes with distance 8 (1000) and 1 (0001). Hence the message can be delivered within two hops. From the above example, it does not matter which of the distances is selected as the first hop. Thus, load balancing of both links and neighbors can be implemented by a node if the next neighbor is randomly selected from all those candidates. Quality of service (QOS) can also be implemented by a node simply by selecting the next hop based on the priority of its candidate neighbors.

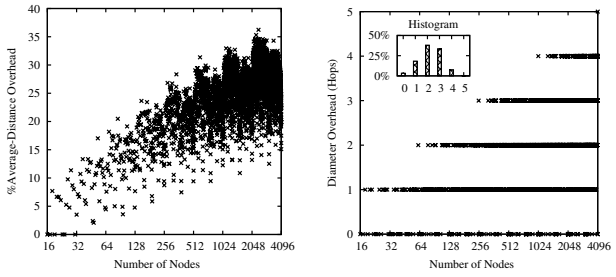
The bit counting method can be used to estimate the number of hops by counting the number of bit-1 of distance between the source and the destination in both clockwise and counter-clockwise directions. The estimated number of hops is the minimum number of bit of both directions. The complexity of this algorithm is O(1).

In practice there are several fast bit counting algorithms (O(1)). They can be divided into two classes. The principal idea of the first class is to count the number of bits in parallel fashion. These algorithms re-arrange an original binary number into several small groups of bits, then count the number of bits in each group simultaneously and finally sum the results. An example of an algorithm in this class is the MIT HAKMEM item 169 [37]. The second class is based on a lookup table of pre-computed bit counting. Figure 5 illustrates an example of the 8-bit lookup table. The LTB8 holds the number of bit-1 of every value (0-255) for an 8-bit number. Counting the bit of a 32-bit integer can be performed by masking out four sets of eight bits in the given integer and indexing them into the LTB8 array. Then, the final result is the sum of results of every set of eight bits. Methods based on the lookup table also have complexity O(1). They might be faster than the parallel counting techniques, however they require extra memory to store the table.

```
1  static int LTB8 [256] =
2  {
3    0,1,1,2,1,2,2,3,...
4    ...,5,6,6,7,6,7,7,8
5  };
6
7  int lookup8 (unsigned int n)
8  {
9    return LTB8[n        & 0xffu] +
10          LTB8[(n>>8)   & 0xffu] +
11          LTB8[(n>>16)  & 0xffu] +
12          LTB8[(n>>24)  & 0xffu] ;
13 }
```

Figure 5. 8-Bit Lookup Table for 32-bit Architecture

Figure 6(a) and Figure 6(b) present the $\bar{d}$ and D overhead of the bit counting method. They emphasize that the bit counting method is sub-optimal.



(a) (b)

Figure 6. Performance of the bit counting method. (a) $\bar{d}$ overhead (%). (b) D overhead (hops).

The average and maximum values of the overhead of $\bar{d}$ and D for the bit counting method in a configuration between 16 and 4096 nodes compared with the original algorithms, basic and variant sub-optimal are shown in Table 1 and Table 2. Estimating the number of hops by a simple bit counting method does not seem to be a good idea. However, it is worth mentioning in this section, since it will be used in the subsequent section.

3.2 Consecutive Bit Elimination Method

Estimating the number of hops using the bit counting method might be too pessimistic, e.g., if a distance between a neighbor and a destination ($|destID - neighborID|$) is 7 (binary is 0111), the bit counting method will estimate the number of hops is 3, jumping

Table 1. $\bar{d}$ Overhead comparison

Algorithms	Values (%)	
	Average	Maximum
Basic	5.555140	11.3849
Variant	4.692380	10.5893
Bit Counting	25.1307	36.2324
Con. Bit Elimination	1.28199	4.43152

Table 2. D Overhead comparison

Algorithms	Values (Hops)	
	Average	Maximum
Basic	0.454055	2
Variant	0.449155	2
Bit Counting	2.23254	5
Con. Bit Elimination	0.351139	2

to nodes with distance 4 (0100), 2 (0010) and 1 (0001), respectively. However, by using the counter-clockwise links we can use only 2 hops by jumping to distance 8 (1000) on one direction and 1 (0001) on the other direction (7 = 8-1). Hence we may estimate more precisely the number of hops between two nodes by eliminating the sets of consecutive bits before counting the number of bits in both directions. Consecutive bits elimination can be performed by adding the value of the least significant consecutive bit to the distance between source and destination in both clockwise (CW) and counter-clockwise (CCW) directions. This procedure is repeatedly performed until there is no consecutive bit left or the result after adding the values is more than j_m, where j_m is a maximum power of 2 that is less than or equal to n.

For example, a distance in clockwise direction is 110 (the binary is 1101110), i.e. CW = 110 and CCW = 0. Consider the distance binary 1101110, the first group of consecutive bits from the right is 1110; therefore, the least significant, consecutive bit is 2 (binary is 10). The first step is performed by adding 2 (10) to both CW and CCW directions. Hence, the distance 110 can be routed by jumping with distance 112 (binary is 1110000) in CW and 2 (binary is 10) in CCW. Notice that the binary of 112 (1110000) still has a consecutive bit, thus the next value to add in both directions is 16 (binary is 10000). After adding 16, distance 110 can be jumped with distance 128 (binary is 10000000) in CW and 18 (binary is 10010) in CCW.

In conclusion, the routing for a distance of 110 in the clockwise direction can be performed within three hops, i.e., one jump in a clockwise direction with the distance 128 and two jumps in a counter-clockwise direction with the distance 2 respectively 16. Again, it does not matter the order in which these jumps are undertaken. Thus, the load balancing and quality of service can be implemented as mentioned in section 3.1.

The complexity of this algorithm is O($log_2(n)$).

An implementation of this algorithm can be done by simply scanning and transforming a given $log_2(n)$ bit distance from right to left using a state diagram as shown in Figure 7, where d_i is an original distance and q_i is a distance in the opposite direction. The label on each transition between states is written in the form of input/output (i.e., Mealy machine).

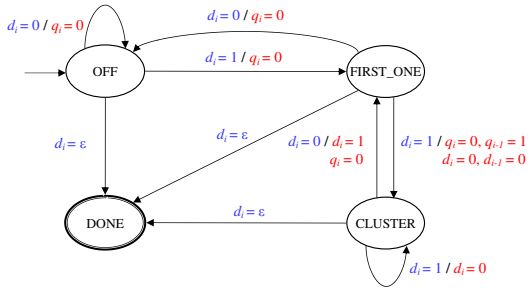


Figure 7. A state diagram to perform bit transformation

Unfortunately, eliminating consecutive bits is difficult to do in constant time using parallel bit transformation because of the dependency between bits. However, this method can be improved by scanning only part of the entire $\log_2(n)$ bits as shown in Figure 8.

```
1  static inline int check_11(int d)
2  {
3      int prev_lsf, lsf;
4      prev_lsf=-1;
5      while(d) {
6          lsf = (d & -d);
7          if ((prev_lsf << 1)==lsf)
8              return prev_lsf;
9          d &= (d-1);
10         prev_lsf = lsf;
11     }
12     return 0;
13 }
```

Figure 8. Consecutive Bit Checking Function

The **check_11** function checks the consecutive bits by skipping all the bit-0s and returns a value of least significant, consecutive bits. It is based on the fact that **(d & -d);** (the statement in line six) gives the value of the least significant bit of **d** and **d &= (d-1);** (the statement in line nine) removes the least significant bits. Thus, the overall performance is dependent on numbers of bits-1.

Figure 9(a) and Figure 9(b) present the $\bar{d}$ and D overhead of the bit counting method. They portray that the consecutive bit elimination method is also sub-optimal. The average and maximum values of $\bar{d}$ and

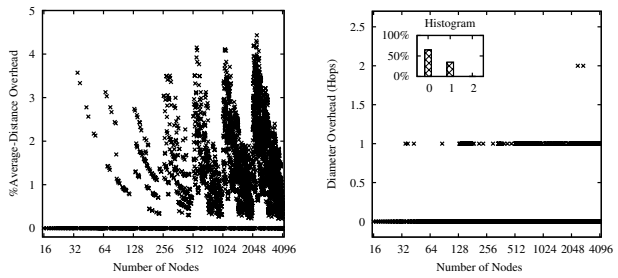


Figure 9. Performance of the consecutive bits elimination method. (a) $\bar{d}$ overhead (%). (b) D overhead (hops).

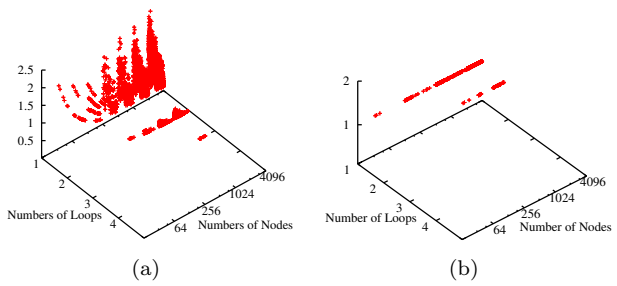


Figure 10. The number of loops (l). (a) $\bar{d}$ overhead (%). (b) D overhead (hops).

D overhead in a configuration between 16 and 4096 nodes are significantly lower than the simple bit counting method as shown in Table 1 and Table 2.

3.3 Equivalence Class Method

Finding the optimal path with distance d from source to destination in BMG size n is actually the same as solving the following problem $\sum_{i=1}^{\delta} x_i j_i \equiv d \mod n$. Hence the equivalence class of distance is applied, i.e., distance d is the same as distance $|d \pm (l \times n)|$, where l is number of loops and $l \in \mathbb{N}$.

The number of loops (l) can be limited by the diameter of BMG, i.e., $|d \pm l \times n| \leq D \times j_k$, where j_k is the maximum power of 2 that is less than or equal to n. Notice that the worst case of a distance has no consecutive bits, e.g., 101, 10101 and 101010. Hence the number of hops required to deliver messages for such distance is $\lceil \frac{\text{Number of bits}}{2} \rceil$. There are $\lceil \log_2(n) \rceil$ bits for a distance. So the diameter of BMG is $O(\lceil \frac{\lceil \log_2(n) \rceil}{2} \rceil)$.

Estimating the number of hops between source and destination based on the equivalence class method is performed by applying the consecutive bit elimination to distance d and $|d \pm (l \times n)|$ in both clockwise and counter-clockwise direction. In practice, a number of loops (l) more than or equal to four gives the optimal result for BMG sized less than 4096 as shown in Figure 10(a) and Figure 10(b). The complexity of this algorithm is $O(l \times \log_2(n))$.

4 Experimental Results

This section presents the evaluation of the new optimal routing algorithm using an equivalence class method compared with the other routing algorithms. The experiments have been conducted on an Intel 2.13 GHz machine with 1 GB of main memory, running on Linux kernel 2.6.18. All experimental programs were compiled with "gcc -O3".

The average elapsed time of calling the function that calculates the next hop for a given distance d between source and destination in BMG size n has been measured. The elapsed time was calculated from $\frac{Tick_{\text{after}} - Tick_{\text{before}}}{Hz}$, where the CPU tick was read with an assembly code (RDTSC instruction). The tick before and after the functions has been recorded. The Hz is the CPU frequency (i.e., the timer precision is up to 10^{-9} seconds on a GHz machine).

Figure 11 illustrates that the new optimal routing algorithm (Equi-Class) is significantly faster than the original optimal routing algorithm (BFS). It is even faster than the basic sub-optimal algorithm. Although the complexity of the Equi-Class method ($O(4 \times \log_2(n))$) is more than the basic algorithm ($O(2 \times \log_2(n))$), the actual performance of Equi-Class is slightly better. This is the result of the implementation technique that does not transform the entire $\log_2(n)$ bits for a given distance. It always skips all bit-0s as mentioned in section 3.2.

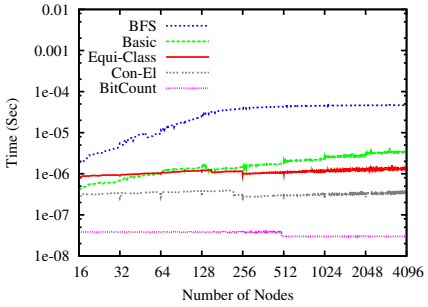


Figure 11. Performance comparison

5 Conclusions and Future Work

This paper presents an optimal (shortest path), two-terminal routing algorithm for BMG networks. This algorithm uses only the destination address to determine the next hop in order to stay on the shortest path. Unlike the original algorithm, it does not require

extra memory for routing tables nor additional information in the packet. The experimental results show that the new optimal routing algorithm is significantly faster than the original optimal algorithm. It is even faster than an original sub-optimal algorithm.

There are several improvements that we plan for the near future. Making the routing algorithm aware of the underlying network topology (in both the LAN and WAN environments) will greatly improve the overall performance for both the unicast and broadcast message transmissions. This is equivalent to adding a function cost on each possible path and integrating this function cost to the computation of the shortest path. Over the longer term, we hope that BMG will become the basic logical topology of the runtime environments within the FT-MPI and Open MPI libraries.

References

[1] Jack J. Dongarra, Hans Meuer, and Erich Strohmaier. TOP500 supercomputer sites. *Supercomputer*, 13(1):89–120, 1997.

[2] Youcef Saad and Martin H. Schultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, 37(7):867–872, 1988.

[3] Subrata Banerjee and Dilip Sarkar. Hypercube connected rings: A scalable and fault-tolerant logical topology for optical networks. 24(11):1060–1079, 2001.

[4] Q.M. Malluhi and M.A. Bayoumi. The hierarchical hypercube: A new interconnection topology for massively parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 05(1):17–30, 1994.

[5] A. El-Amawy and S. Latifi. Properties and performance of folded hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 2(1):31–42, 1991.

[6] J. M. Kumar and L. M. Patnaik. Extended hypercube: A hierarchical interconnection network of hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 3(1):45–57, 1992.

[7] Nian-Feng Tzeng and Sizheng Wei. Enhanced hypercubes. *IEEE Transactions on Computers*, 40(3):284–294, 1991.

[8] Franco P. Preparata and Jean Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Commun. ACM*, 24(5):300–309, 1981.

[9] Ahmed Louri and Costas Neocleous. A spanning bus connected hypercube: A new scalable optical interconnection network for multiprocessors and massively parallel systems. *IEEE/OSA Journal of Lightwave Technology*, 15(7):1241–1252, 1997.

[10] Ahmed Louri and Hongki Sung. An optical multimesh hypercube: A scalable optical interconnection network for massively parallel computing. *Journal of Lightware Technology*, 12(4):704–716, 1994.

[11] Sabine Ohring and Sajal K. Das. Folded petersen cube networks: New competitors for the hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 7(2):151–168, 1996.

[12] Kumar N. Sivarajan and Rajiv Ramaswami. Lightwave networks based on de bruijn graphs. *IEEE/ACM Trans. Netw.*, 2(1):70–79, 1994.

[13] Elango Ganesan and Dhiraj K. Pradhan. The hyper-debruijn networks: Scalable versatile architecture. *IEEE Transactions on Parallel and Distributed Systems*, 04(9):962–978, 1993.

[14] Chienhua Chen, Dharma P. Agrawal, and J. Richard Burke. dbcube: A new class of hierarchical multiprocessor interconnection networks with area efficient layout. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1332–1344, 1993.

[15] Geetha Panchapakesan and Abhijit Sengupta. On a lightwave network topology using kautz digraphs. *IEEE Transactions on Computers*, 48(10):1131–1138, 1999.

[16] M. J. Karol. Optical interconnection using shufflenet multihop networks in multi-connected ring topologies. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 25–34, New York, USA, 1988. ACM Press.

[17] N. F. Maxemchuck. Regular mesh topologies in local and metropolitan area networks. *AT&T Technical Journal*, 64(7):1659–1685, September 1985.

[18] Stuart Campbell, Mohan Kumar, and Stephan Olariu. The hierarchical cliques interconnection network. *Journal of Parallel and Distributed Computing*, 64(1):16–28, 2004.

[19] James R. Goodman and Carlo H. Sequin. Hypertree: A multiprocessor interconnection topology. *IEEE Transactions on Computers*, 30(12):923–933, 1981.

[20] Thara Angskun, Graham E. Fagg, George Bosilca, Jelena Pješivac-Grbović, and Jack Dongarra. Scalable fault tolerant protocol for parallel runtime environments. In *Recent Advances in PVM and MPI*,

number 4192 in LNCS, pages 141–149. Springer, 2006.

[21] Thara Angskun, Graham E. Fagg, George Bosilca, Jelena Pješivac-Grbović, and Jack J.Dongarra. Self-healing network for scalable fault tolerant runtime environments. In *Proceedings of 6th Austrian-Hungarian workshop on distributed and parallel systems*, Innsbruck, Austria, September 2006. Springer-Verlag.

[22] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

[23] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[24] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu a nd Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality pr operties. In *USENIX Symposium on Internet Technologies and Systems*, pages 113–126, Seattle, WA, USA, 2003. In proceedings of the 4th USENIX Symposium on Internet Technol ogies and Systems (USITS '03).

[25] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IP TPS)*, Cambridge, MA, USA, 2002.

[26] Dahlia Malkhi, Moni Naor, and David RatajczakDavid Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Comput ing*, pages 183–192. ACM, 2002.

[27] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[28] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[29] Thara Angskun, George Bosilca, and Jack J.Dongarra. Binomial graph: A scalable and fault-tolerant logical network topology. In *Proceedings of 5th International Symposium on Parallel and Distributed Processing and Applications (ISPA07)*, ON, Canada, August 2007. Springer-Verlag.

[30] J. C. Bermond, F. Comellas, and D. F. Hsu. Distributed loop computer networks: A survey. *Journal of Parallel and Distributed Computing*, 24(1):2–10, 1995.

[31] Ramón Beivide, Enrique Herrada, José L. Balcázar, and Agustin Arruabarrena. Optimal distance networks of low degree for parallel computers. *IEEE Trans. Comput.*, 40(10):1109–1124, 1991.

[32] Jin-Yi Cai, George Havas, Bernard Mans, Ajay Nerurkar, Jean-Pierre Seifert, and Igor Shparlinski. On routing in circulant graphs. volume 1627 of *Lecture Notes in Computer Science*, pages 360+, 1999.

[33] Tomaz Dobravec, Janez Zerovnik, and Borut Robic. An optimal message routing algorithm for circulant networks. *J. Syst. Archit.*, 52(5):298–306, 2006.

[34] Borut Robic. Optimal routing in 2-jump circulant networks. Technical Report UCAM-CL-TR-397, University of Cambridge, Computer Laboratory, jun 1996.

[35] Ilyong Chung. Construction of a parallel and shortest routing algorithm on recursive circulant networks. In *HPC '00: Proceedings of the The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region-Volume 2*, page 580, Washington, DC, USA, 2000. IEEE Computer Society.

[36] Prasanna Ganesan and Gurmeet Singh Manku. Optimal routing in chord. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 176–185, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[37] Michael Beeler, William Gosper, and Rich Schroeppel. HAKMEM: Memo 239. Technical report, Artificial intelligence Laboratory, Massachusetts Institute of Technology, 1972.