Self-Discovery on Hierarchical Three Dimensional Torus

Jose Castanos* (castanos@us.ibm.com) Rahul Garg† (grahul@in.ibm.com) José Moreira* (jmoreira@us.ibm.com)
Vinayaka Pandit† (pvinayak@in.ibm.com) Meeta Sharma† (meetasha@in.ibm.com)

ABSTRACT
The advances in technology have made it possible to build massively parallel supercomputers connected using interconnection networks. IBM's Blue-Gene/L (BG/L) machine with 65536 (2^{16}) processors operating at 180 TFLOPS is a good example. BG/L is planned to have a three dimensional hierarchically organized torus topology. The process of discovering size of a machine, assigning consistent, and unique spatial coordinates to the processors for identification, is called *self-discovery*. An automated, and distributed fault-tolerant algorithm to achieve self-discovery is very important for the operation of such a system. In this paper, we present an autonomic, distributed, and fault-tolerant algorithm that hierarchically achieves self-discovery on a BG/L machine of previously unknown size, unknown extent along its dimensions. Our algorithm works even in the presence of node and link failures.
Keywords: Autonomic computing, Fault tolerance, Self-discovery, Massively parallel systems.

1. INTRODUCTION
Blue-Gene/L system is based on a large scale cellular architecture. The basic building blocks in such an architecture is a *cell*, which is replicated in a pattern to form the whole system. Each cell is isomorphic to other cells and acts locally. Such architectures can be connected in several topologies, like *torus, mesh, tree etc.* Large systems are hierarchically constructed using smaller sub-systems of the same topology. The advantage of such an architecture is *unlimited scalability*.

The Blue-Gene/L system consists of maximum 65536 (2^{16}) nodes, connected through a three dimensional torus network. The system is hierarchically constructed using smaller toruses (cubes) as shown in Figure 1. The smallest entity in the system is a *chip*, which is also called a *compute node*. Each chip consists of two PowerPC 440 cores and 256MB of DRAM. Each node has six bi-directional links to the torus. Two nodes combine to form a (2x1x1) torus called compute card. Sixteen compute cards combine into a node board which is a 32 node torus in a logical topology of 4x4x2. Sixteen node boards combine to form a 8x8x8 torus called the midplane. A cabinet consists of two such midplanes. The system is a 32x32x64 torus consisting of sixty four cabinets.

Due to its massive size, several key parameters such as size of the system, its extent along each dimension, and spatial coordinates (addresses) of nodes cannot be pre-programmed into the system. The process of discovering these parameters automatically during initialization of the system is called *self-discovery*. It forms an important component of such systems. Such large systems are also very prone faults due to node and link failures. Therefore, an autonomic, distributed and fault tolerant algorithm for self discovery is essential.

We begin by formally defining the problem in Section 2. In Section 3 we describe the algorithm to resolve disorientation. We describe the algorithm to assign unique and consistent spatial coordinates to the nodes and to discover the extent of the torus in Section 4. We conclude in Section 5.

2. PROBLEM FORMULATION
The BG/L system can be modeled as a large three dimensional torus, hierarchically constructed using smaller three dimensional cubes (we also refer to these cubes as toruses, as the interconnections out of a cube can be looped back to form a torus). The problem of self discovery is to discover the size, extent of the system along each axes and assign consistent spatial coordinates to the nodes. This problem can be solved hierarchically: given that the smaller toruses have discovered their sizes, extent and spatial coordinates, how to discover the size, extent, and spatial coordinates of the bigger torus?

In the BG/L machine a cabinet consists of two midplanes. Each midplane is a 8x8x8 torus consisting of sixteen node boards. These boards are connected to each other via a back plane. The interconnections of different compute cards in a node board and of different node boards in a midplane are fixed and done in hardware that is tested extensively. Therefore it is unlikely that these interconnections are incorrect. However, the interconnections across midplanes is

*IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA.
†IBM India Research Lab, Hauz Khas, Delhi 110016, INDIA.

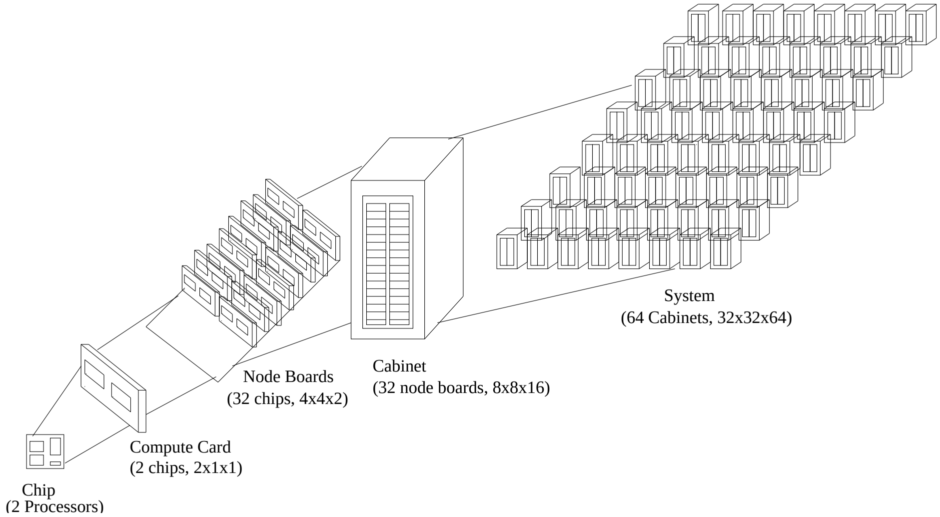

Figure 1:

done manually using external cables. Therefore it is possible that, while building the machine these cables are misconnected. In some cases, these cables might be connected incorrectly. In some other cases, these cable might be connected such that machine still forms a torus, but different midplanes are not oriented correctly.

The basic problem in carrying out the self-discovery of the big torus is the *disorientation* problem. Note that two small toruses may be interconnected (see Figure 2) in a way that their relative orientations mismatch. For example, the positive x-axis of a torus may be connected with the negative y-axis of the adjacent torus. So a request by a node in one torus to pass message along x-axis may be interpreted as passing along y-axis by nodes in the adjacent torus. Note that the smaller torus can be collapsed into a node and the self discovery problem can be solved on the big torus considering each small torus as a single node. However, we have retained the details for the small toruses for two reasons: (i) it is closer to the BG/L machine, and (ii) collapsing the torus loses a spatial relation between nodes on interconnecting faces of two different toruses.

In order to carry out routing of the messages in the torus, the routing sub-system needs to know the spatial coordinates, the size and the extent of the system. Therefore, we do not make use of the routing sub-system to carry out the self discovery. We assume that every node is capable of communicating only with its immediate neighbors by specifying the local direction in which the message should be sent.

Such a large system is expected to suffer node, and link failures that could partition the system into disjoint connected components. In the independent fault model, the probability of such an event is very small. We therefore assume that the system forms a single connected component. We also assume that every node is in-built with a unique identifier. It can be achieved either during processor manufacture or by generation of large random numbers.

Our algorithm works in two phases. It assumes that the big torus is constructed using small toruses each of which know the size, extent and have a consistent spatial coordinate assignment locally. The first phase resolves the *disorientation* between every neighboring small toruses (also called S-torus). The algorithm is distributed, fault tolerant and has no central point(s) of failure. We show that this phase has a very high probability of success for such high fault-rates as 15%. Our second phase assigns unique, and consistent spatial coordinates in the big torus through a leader election algorithm. We note that the same algorithm can be shown to be *self-stabilizing* as well, i.e., it tolerates transient faults such as memory corruption, and corrupted messages. In the absence of transient faults, which is the setting we are in, our algorithm runs in $O(d)$ where d is the diameter of the network connecting the system nodes. Our algorithm has very minimal space requirement at each node, which is $O(1)$.

3. RESOLVING DISORIENTATION

Consider two S-toruses which are connected to each other through a face (i.e. an 8x8 plane). We show that two pairs of adjacent nodes in these two toruses are sufficient to resolve the *disorientation* between these toruses. We use it to develop a fault-tolerant technique to orient all the toruses.

Figure 2 shows interconnection of two S-toruses with *disorientation*. The yz-plane on the right of an S-torus is connected with the xz-plane on the left of another S-torus. Note that though a node does not know its absolute orientation of its (x, y, z) axis, it knows their relative orientations. So, a node (say $(1, 1, 1)$) receiving a message traveling in a direction (say from node $(0, 1, 1)$) can forward it along the same direction (to the node $(2, 1, 1)$) without knowing the absolute orientation of its axes.

If two adjacent nodes say a on $S1$, and a' on $S2$ are both functioning, then by exchanging one message each they can resolve the relative orientation along the link which connects them. For example, the node a on $S1$ can send a message containing its coordinates (say (x, y, z)), and the coordinates of a' relative to $S1$ (i.e. $(x + 1, y, z)$). The node a' receives

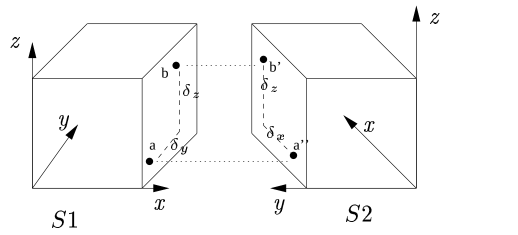

Figure 2: Interconnecting of two small S-toruses

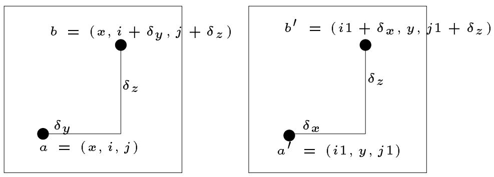

Figure 3: Resolving orientations

this message along the negative Y axis of $S2$. Hence, a' can conclude that the positive X axis of $S1$ is aligned with the negative Y axis of $S2$. Let us call such a message with local coordinate, and relative coordinate of its neighbor as increment messages.

Define the two pairs (a, a') and (b, b') as *satisfying quadruples* if and only if: (i) nodes a, b are in $S1$, nodes a', b' are in $S2$, (ii) node a is connected to a', b is connected to b', (iii) $\delta_y(a, b) = y(a) - y(b) \neq 0, \delta_z(a, b) = z(a) - z(b) \neq 0$, and (iv) $|\delta_y(a, b)| \neq |\delta_z(a, b)|$. Note that the definition is with respect to nodes on a yz-plane, and easily generalizes to any face.

If all the four nodes in the satisfying pair $(a, a'), (b, b')$ are functioning (and the links connecting them are functioning), then all four nodes a, a', b, b' can resolve all their orientations with respect to $S1$ and $S2$. For this, the nodes (a, a') and (b, b') orient themselves along the axis connecting them as described earlier. Now, all the nodes exchange their local coordinates. Each of these nodes can see than $\delta_y(a, b) = \delta_x(a', b')$. Therefore, the positive Y axis of $S1$ is oriented with the positive X axis of $S2$ (see Figure 3). Similarly, since $\delta_z(a, b) = \delta_z(a', b')$ the positive Z axes of $S1$ and $S2$ are aligned. This procedure to discover the relative orientation of two adjacent toruses is called **OrientAdjacent**.

Now, we use **OrientAdjacent** to resolve orientations in a fault-tolerant manner. Every node on the face of a torus exchanges its local coordinates with its adjacent node in the adjacent torus. Every node a on the face of a torus with the adjacent node a' on the other torus, broadcasts the pair $(x(a), y(a), z(a)), (x(a'), y(a'), z(a'))$ to every other on the same face. Any other node b on the same face that can form a satisfying pair $(a, a'), (b, b')$ using the broadcast messages that reach b uses **OrientAdjacent** to resolve the relative orientation between the two toruses. Any node that has resolved the orientation, broadcasts the orientation map to rest of the nodes in its local torus. Note that the algorithm takes at most $2d+2$ steps to to resolve the orientation, where d is the diameter of the local torus. So, if a node does not receive the orientation map in $2d+2$ steps it can conclude that the torus cannot be aligned using the **FaultTolerantOrient** algorithm. When this happens, the node broadcasts the failure signal.

As we assume that our machine is connected, we can accomplish broadcast among nodes in an S-torus by techniques such as reverse path forwarding, time linear with size of S-torus. The nodes on a face resolve orientations using **FaultTolerantOrient**, and then the resolved orientations are broadcast to other nodes on the same face. To eliminate any confusion about messages from different faces, messages can be tagged with face identification.

3.1 A lower bound on the probability of success

We will place a lower bound on the probability that **FaultTolerantOrient** succeeds. We assume that a node on a face has failed if (i) it has failed, or (ii) its link connecting to its corresponding node has failed. Let us assume that every node fails with a probability q, and a link is functioning with a probability r. Our algorithm fails to resolve orientations only when it fails to find satisfying quadruples. We calculate upper bound on the probability that no such satisfying quadruples is found. The probability that a node is functional is given by $p = 1 - q$. We consider the general case when two cubes of size N^3 are interconnected. So the faces are of six $N \times N$. Consider a node a on face $S1$

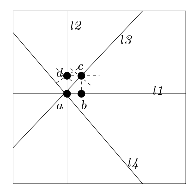

Figure 4: Illustration of lower bound calculation

as shown in Figure 4. All the nodes on the straight lines $l1$, $l2$, $l3$, and $l4$ have either 0 displacement along one of the dimensions or have equal displacements on both. We will call these lines as *unallowed lines* for node a. All other nodes ($N^2 - 4N + 3$ of them) satisfy δ_1, and δ_2 property. So, the probability that we do not find a satisfying quadruple involving a when both a, and a' are alive is given by

$$rp^2(1 - (1 - rp^2)^{N^2 - 4N + 3}) \quad (1)$$

Suppose one of a or a' was down, then let us look at the probability of finding a satisfying quadruple with the next node on the line $l1$. This is given by

$$(1 - rp^2)rp^2(1 - (1 - rp^2)^{N^2 - 4N + 3}) \quad (2)$$

Similarly, we can calculate for the $(i + 1)$st node on $l1$ when no node upto i is involved in a *satisfiable quadruple* is

$$(1 - rp^2)^i rp^2(1 - (1 - rp^2)^{N^2 - 4N + 3}) \quad (3)$$

Suppose that we are not able to find a *satisfiable quadruple* involving any node on $l1$, then we try to find a node on line $l3$ with node a as the first node at the beginning. Note that for the nodes on the line $l3$ some of the nodes on $l1$ are among the $N^2 - 4N + 3$ nodes satisfying the δs property.

But there are at most $N-2$ of them as two of the nodes on $l1$ fall on its unallowed lines. So the probability is

$$(1-rp^2)^{i+N-1}rp^2(1-(1-rp^2)^{N^2-5N+5}) \quad (4)$$

We try to find a satisfying quadruple along $l2$ if we fail to find any such quadruple along $l1$, and $l3$. For the $(i+1)$st node on $l2$, among N^2-4N+3 nodes that can be considered, $2N-4$ belong to $l1$, and $l3$ because 4 nodes on $l1$, and $l3$ are along its unallowed lines. Hence the probability is

$$(1-rp^2)^{i+2N-2}rp^2(1-(1-rp^2)^{N^2-6N+7}) \quad (5)$$

From 3, 4, and 5 we have a lower bound on the probability of success given by

$$\begin{aligned} LB &= \sum_{i=1}^{i=N} rp^2(1-rp^2)^{i-1}(1-(1-rp^2)^{N^2-4N+3}) + \\ &\quad \sum_{i=N+1}^{i=2N-1} rp^2(1-rp^2)^{i-1}(1-(1-rp^2)^{N^2-5N+5}) + \\ &\quad \sum_{i=2N}^{i=3N-2} rp^2(1-rp^2)^{i-1}(1-(1-rp^2)^{N^2-6N+7}) \\ LB &>= \sum_{i=1}^{i=3N-2} rp^2(1-rp^2)^{i-1}(1-(1-rp^2)^{N^2-6N+7}) \\ LB &>= (1-(1-rp^2)^{N^2-6N+7})(1-(1-rp^2)^{3N-2}) \quad (6) \end{aligned}$$

For a machine with k inter-face connections, the lower bound on overall fault tolerance is given by

$$FTLB >= LB^k \quad (7)$$

A BG/L machine is a 32x32x64 torus composed of 128 midplanes of size 8x8x8. This can be viewed as a 2x2x4 torus, hierarchically constructed from sixteen smaller tori, where each of these smaller torus is a 2x2x2 torus constructed from eight 8x8x8 tori (midplane). Applying Eq. 6 and 7 hierarchically for the BG/L machine we get

$$LB \geq ((1-x^{23})(1-x^{23}))^{12}((1-x^{167})(1-x^{46}))^{20}, \quad (8)$$

where

$$x = 1 - rp^2. \quad (9)$$

So from 8, and 9 even such high fault rate as $p = 0.85$, and $q = 0.15$, and $r = 0.85$ yields a lower bound on the probability of correct operation of the algorithm as $1-10^{-8}$.

4. ASSIGNING SPATIAL COORDINATES

To assign spatial coordinates, a global leader is elected which becomes the origin (coordinate $(0,0,0)$). Now, the system is hierarchically partitioned into 32x32x64 torus (the system), 16x16x16 tori (small torus) and 8x8x8 tori (midplane). A local leader is elected in each of the 16x16x16 torus and each midplane aligns its orientation with respect to its local leader using the **FaultTolerantOrient** algorithm discussed in Section 3. Now, the midplanes re-align their orientation with respect to the global leader using the **FaultTolerantOrient** algorithm on tori of size 16x16x16. After aligning their orientation with the global leader, the coordinate of nodes are set to their displacement form the leader. This is done by locally exchanging and updating coordinate information among neighbors. The algorithm is distributed, fault tolerant, and self-stabilizing. Figure 5 describes the details of this algorithm.

Every node executes local steps, and use logical direction variables to propagate relative orientation information. Values of $\pm1, \pm2, \pm3$ denote the $\pm x$ axis, $\pm y$ axis and $\pm z$ axis. For example logicalDir[1] = -2 means that the logical positive x-axis it towards physical negative y-axis. The local steps try to *follow* a node who with higher leader id. When a node discovers that its neighbor has a higher leader id, it orients itself according to its neighbor and assigns coordinates that are consistent with that of its neighbor. We assume that all the variables used by the program are free from corruption. This algorithm can be extended to deal with memory faults by employing techniques in [4], [1]. The variables of node i are accessed as if i is a structure.

The algorithm terminates when no node receives a message in K steps. If K is at least $d+1$ where d is the diameter of the network, then the algorithm assigns the coordinates, and logical directions correctly. K can be set to a sufficiently large estimate of the network diameter (in presence of faults). It should be noted that, the coordinates assigned can have negative components as well. If a node i has a negative coordinate, then it can set it to corresponding positive coordinate by just adding the extent of the network in that dimension.

LEMMA 1. *The leader node will always be the origin.*

Proof. As our, variables are not prone to corruption, the leader node with the highest id will never follow any other node. □.

LEMMA 2. *For every node i at a distance d from the leader will discover its coordinates with respect to the leader in d steps.*

Proof. We prove by induction. The base case is $d = 0$. Assume that it is true for all nodes at a distance $d-1$, and less. Any node at distance d is adjacent to a node *nbr* at distance $d-1$ from the leader. At time step d, *nbr* has a *lid* estimate which is greater than that of i. So i decides to *follow nbr*. □.

LEMMA 3. *If a node i does not execute a follow move in d_{max} steps, where d_{max} is a reasonable upper bound on the diameter of the torus, then node i has discovered the leader and assigned its coordinates consistently.*

Proof. Follows from Lemma 2 □.

THEOREM 1. *Our algorithm terminates with correct coordinates assigned to all nodes. It runs in $O(d_{max})$ time, and requires constant space at each node.*

```
Variables: Each node maintains the following variables.
    lid:            Leader id.
    coords[3]:      Global (x, y, z) coordinates.
    map(j)[3]:      The relative orientation map w.r.t. the neighboring node j.
    logicalDir[3]:  The logical x, y, z directions.
    dir(i, j):      The physical link on which j is connected to i.

Initialize(i)
    i.lid = id;
    i.coords = {0, 0, 0}
    logicalDir = {+1, +2, +3}
    dir(i, j) = physical link on which j is connected to i
    map[j] = FaultTolerantOrient(j)
end Initialize

Local action for node i with a neighbor j:
do
    if (i.lid < j.lid) // According to j, leader has a higher id
        follow(i, j) // Orient and assign coordinates consistent with node j's view
        Send messages to neighbors
    endif
while (there is no message for K steps)

follow(i, j)
    i.lid = j.lid
    if (i and j are on different toruses) then // Orient axes according to node j's orientation
        for (k = 1 to 3)
            if (j.logicalDir[k] > 0)
                i.logicalDir[k] = i.map(j)[j.logicalDir[k]]
            else
                i.logicalDir[k] = - i.map(j)[- j.logicalDir[k]]
            endif
        endfor
    endif
    // Assign coordinates consistent with node j's view
    Let k be such that | logicalDir[k] | = | dir(i, j) |
    if (i.logicalDir[k] = dir(i, j))
        i.coords[k] = j.coords[k] - 1
    else
        i.coords[k] = j.coords[k] + 1
    endif
end follow
```

Figure 5: Algorithm to assign unique (x, y, z) coordinates

The time, and space complexity of our algorithm is optimal, because just a communication between any two nodes in a network requires $O(d_{max})$ time.

4.1 Discovering Extent

In order to discover extent, a slight modification of the algorithm in Figure 5 is required. Upon receiving a message from the neighbor with the same leader id as that of the node, the node computes its coordinates as in *follow*. If these coordinates are different than that stored with the node, then subtracting the two gives an integer multiple of the extent. This number is then broadcast to all the nodes. The extent of the network along each of the axis is the greatest common divisor (GCD) of these multiples of extents. After discovering the extent, each node can set its coordinates to be non-negative (by adding appropriate extent to their negative coordinates).

5. CONCLUSIONS

In this paper, we have described a distributed and fault-tolerant algorithm to discover the initial configuration of a massively parallel hierarchically organized torus network, and also assign consistent, and unique spatial coordinates to all the nodes of the torus. Such an algorithm is very critical in initializing of large parallel systems.

An interesting generalization of the problem is to maintain a consistent configuration when the variables (memory) used by the programs are prone to corruption. Such an algorithm will be *self-stabilizing*, and we plan to consider it in our future work.

6. REFERENCES

[1] Y. Afek and A. Bremler. Self-stabilizing unidirectional network algorithms by power supply. *Chicago Journal of Theoretical Computer Science*, December 1998.

[2] Y. Afek, S. Kutten, and M. Young. Memory-efficient self-stabilizing protocols for general networks. In *4th International workshop on distributed algorithms*, volume 484 of Lecture Notes in Computer Science, pages 15–28. Springer-Verlag, September 1990.

[3] G. Almasi *et al.* Cellular Supercomputing with System-On-A-Chip. In *International Solid State Circuits Conference*, 2001.

[4] V. Pandit and D. Dhamdhere. Self-stabilizing maxima finding on general graphs. Technical report, IBM Research Report RI02009, 2002.

[5] IBM BG/L Team. An Overview of BlueGene/L Supercomputer. In *Proccedings of ACM Supercomputing Conference*, 2002.