УДК 004.722

YURII KULAKOV
ALEXEY GOROBETS

ALGORITHM FOR CREATING BINOMIAL GRAPH TOPLOGY IN A FULLY CONNECTED SYSTEM

In this paper we described an algorithm for the constructing binomial graph topology in a fully connected system. Parallel execution of the algorithm parts form a ring topology, and then fill in lists of links to nodes, in accordance with rules that describe binomial graph topology.

Creating binomial graph topology

For systems, which consist of a set of parallel execution processes, an important parameter is the delay in information sharing. Exchange of messages between processes is done by the executive environment. The configuration of the environment should optimally distribute the load on system nodes. As topology, which will reduce the load on the transfer, was proposed binomial topology graph [2]. Initially, the algorithm for creating binomial graph topology [7] was created for process tree topology. The choice of this topology is not accidental - this topology is actually very easy to create - the processes to solve complex problems, or originally designed as a multi-threaded create new threads that run on other tasks. In the process of build a hierarchy of processes, this is a tree.[1]

This decision is also due to the fact that at the initiation of a parallel runtime environment for which to build this topology, the boot loader initially creates a process tree, which contains the necessary information for the algorithm. Each process in the tree has an ID of its parent process (as an index or some unique value), the ID of the child process.

In ring topology, each node knows the identity of the node that follows it, and the ID of the previous node. This topology distributes the workload associated with the transmission of messages between nodes. Also, each element of the ring (in addition to information about the previous and next nodes) formed two arrays that contain the identifiers of nodes that are at power of 2 distances from the node in the clockwise and counter-clockwise.

Creating binomial graph topology from tree topology

In order to create the topology described above created algorithm, which can be divided into two algorithms: an algorithm for creating a ring of processes from the process tree, and the algorithm to create a binomial graph of the ring topology [3, 6].

Algorithm creating ring processes topology from the process tree is presented in [7].

First basic idea of that algorithm is to create a chain of parents and their children first, and connect those chains in the ring.

Chain creation is using rules 1 and 2 of Algorithm 1. Rule 1 can be used by any process that has at least one process-heir. When you run it, the first process-successor is recorded as the next process in the ring. When sending a message with its identifier, the process sets itself as the previous element of the ring for his first child process with Rule 2.

Should be noted that the result is a chain with a "leaf" of the tree on one end, and each leaf of the tree will be the end point of any chain.

Combine chains in the ring is reduced to searching for each "leaf" of the first free process to set him as the next in the ring [5]. Rule 3 can be used to the leaf of the tree, and as a result it will post Info to the parent process. Rule 4 describes how to react to the flow of the message Info.

During the second algorithm, each node regularly sends the identifiers of its neighbors to each other (Rule 1). When the process is the process ID at 2^i distance, it stores it in the appropriate array, and sends it to a distance of 2^i in both directions.

Creating binomial graph topology in randomly configuration system

In order to implement the algorithm in a system that does not create a process tree program need modification relations environment to create the tree. Given the fact that the configuration of the tree is not being critical of the algorithms described above, the algorithm for constructing the tree

topology is reduced to the choice of the root node, and steps of forming the branches of a tree.

To determine the root node will assume that the algorithm of tree creation is performed in parallel for each branch, respectively choose the root node with the highest degree will give positive impact on the configuration tree.

Tree creation algorithm will be next:

1) Find the node with the highest degree and define it as root node.

2) Define of all connected with the current node nodes as child nodes in the tree.

3) All child nodes set this node as the parent node in the tree. All child nodes are added to the list.

4) While the list has at least one node:

- The first node in the list is as current;

- All the nodes that are directly connected with the current, not marked as processed and are not in the list is defined as a child of the current node;

- All the child nodes sets current node as the parent node in the tree. All child nodes are added to the list;

- Current node marked as processed and removed from the list.

Main advantage of this algorithm is the lack of an integrated network transformation. The tree is formed on the basis of already existing connections.

Algorithm for creating binomial graph topology in a fully connected system

Algorithms described above are redundant for strongly connected systems. It clearly manifested in fully connected systems, as required for the binomial graph connections are already present, and the creation of intermediate topologies, reconfiguring the system would be clearly excessive. For such systems we need an algorithm that will select the necessary links from existing, and create relevant lists of clockwise and counter clockwise neighbors for each node. Also, the algorithm should run in parallel on each node, which will accelerate the construction of binomial graph topology.

Generally, for each of the N nodes to create two lists - CW and CCW, which contain nodes at a distance $\{(i+1) \bmod n, (i+2) \bmod n, \ldots, (i+2^k) \bmod n \mid 2^k <= n\}$ and $\{(n-i+1) \bmod n, (n-i+2) \bmod n, \ldots, (n-i+2^k) \bmod n \mid 2^k <= n\}$ [4].

Connections between nodes at distance 1 form a ring topology, the concrete realization of which is not important for the formation of the rest of the list.

Each individual node can form a link at a distance of 2^i for any two nodes at a distance of 2^{i-1} from it. Nodes receive a message that indicates the node identifier and the distance to it, and set the appropriate values in their CW and CCW lists.

Algorithm for creating binomial graph topology in a fully connected system is consists of two parts. The first part of the algorithm will randomly form a ring topology, choosing neighboring nodes at a distance of 1. The second part of the algorithm will generate the appropriate lists CW and CCW for each of the nodes.

Input data for the algorithm is a list of identifiers of the available nodes of the current node. Algorithm is executed in parallel on each node in the topology.

Algorithm for creating binomial graph topology in a fully connected system:

1) Send a random node request the presence of "neighbor CW." If receipt of the answer is its Id set up a node to which the request was send, as the current "neighbor CW" and add it to the list of ID CW.

2) Pending for request the presence of "neighbor CW". Upon receipt of such a request, if the current "neighbor CCW" is not defined, set the node that sent the request, as the current "neighbor CCW", add the Id to the list CCW and send identifier of current "neighbor CCW" in response.

3) Send Id of the current "neighbor CW" in the node current "neighbor CCW".

Send Id of the current "neighbor CCW" in the node current "neighbor CW".

4) Pending for receiving node id from current "neighbor CW". Upon receipt of such, if received Id not in CW list, and not equals to current "neighbor CCW", then set the received Id as the current "neighbor CW" and add it to the CW list. Otherwise, set end flag true.

Pending for receiving node id from current "neighbor CCW". Upon receipt of such, if received Id not in CCW list, and not equals to current "neighbor CW", then set the received Id as the current "neighbor CCW" and add it to the CCW list. Otherwise, set end flag true.

5) If end flag false, go to step 3.

Steps 1 and 2 are the first part of the algorithm. They are executed in parallel on all the nodes, and randomly form the ring of nodes. Steps 3 and 4 are performed in a loop, forming lists of CW and CCW components, as long as it reaches the maximum value of the distance ($\{(i+1) \bmod n, (i+2) \bmod n, \ldots, (i+2^k) \bmod n \mid 2^k <= n\}$ and $\{(n-i+1) \bmod n, (n-i+2) \bmod n, \ldots, (n-i+2^k) \bmod n \mid 2^k <= n\}$). This

condition will be satisfied, when each node will reach the most remote node according to the structure of topology.

This algorithm will generate binomial graph topology in a fully connected system more optimal way. This is achieved by eliminating the intermediate transformations topology, the parallel execution of parts of the algorithm on each node.

Algorithm work example

To create binomial graph topology in fully connected system of 8 nodes, algorithm described above will perform the following steps. Each node will have a unique identifier and a list of links to all the nodes in the system. Figure 1 shows the initial state of the system.

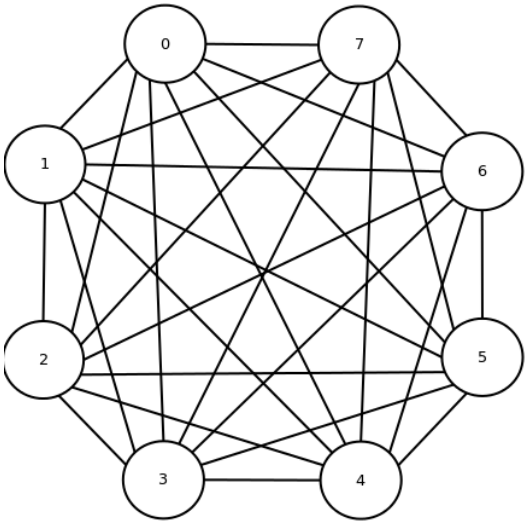

Fig. 1. The initial state of the system

On the first stage, ring of process formatted randomly. The result of the first stage is shown in Figure 2.

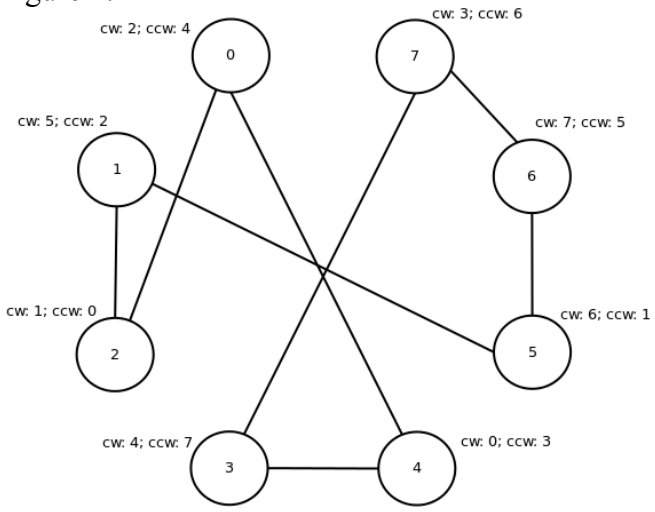

Fig. 2. The result of the first stage of the algorithm

As you can see, each of the nodes began to form lists of neighboring nodes clockwise (CW) and counterclockwise (CCW).

At the next stage of formation of these lists. In the first step be added to the list of nodes that are at distance 2. As described above, this is due to send the current node from the list of neighboring nodes on the left to the node that is currently on the list of neighboring nodes on the right, and vice versa. The result is a first step; the second part of the algorithm is shown in Figure 3.

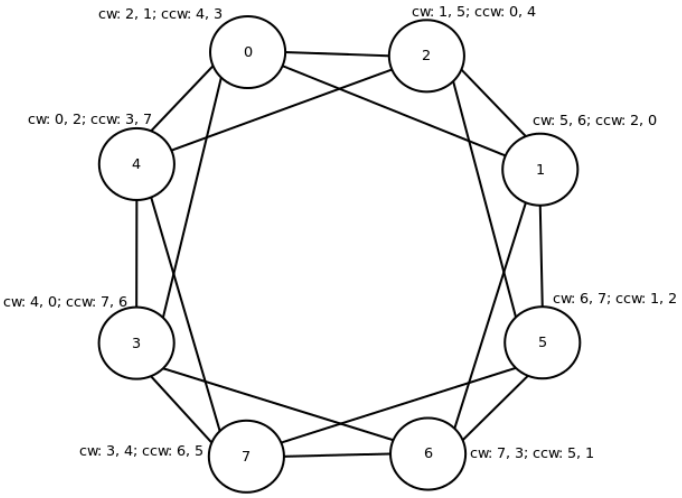

Fig. 3. The result of a first step, of the second part of the algorithm

End condition is not true, so the next step of the second part of the algorithm proceeds. The result of his work is shown in Figure 4.

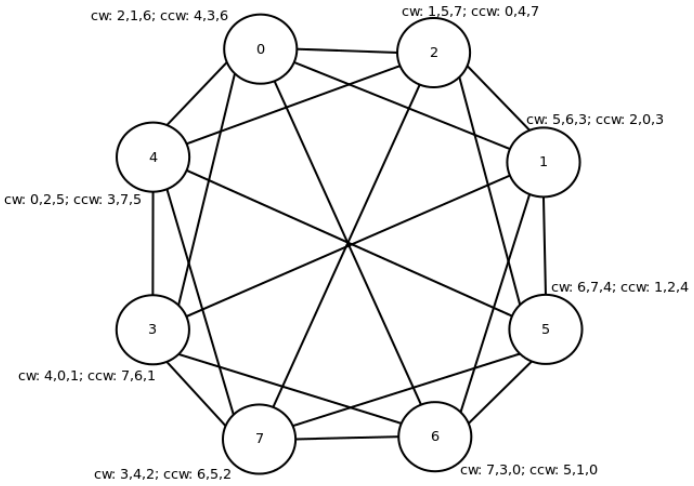

Fig. 4. The result of the second step of the second part of the algorithm

As it shown in Figure 4, the current node from the list of CW is also the current node in the list of

CCW, which is a condition of the completion of the second part of the algorithm.

As a result of the algorithm in each node was formed two lists of nodes, containing nodes at distance 2^k, which actually creates a list of links for the binomial graph topology.

References:

1. Sutter, H. Software and concurrency revolution / H. Sutter, J. Laurus // Queue. – New York: ACM, 2005. – Vol. 3, № 7. – P. 54-62. – ISSN 1542-7730.
2. T. Angskun, G. Bosilca, and J. Dongarra. Binomial graph: A scalable and fault-tolerant logical network topology. - Heidelberg: Parallel and Distributed Processing and Applications, ISPA 2007 - Vol. 4742/2007 of Lecture Notes in Computer Science - P. 471–482. Springer Berlin /.
3. Lemarinier P. Constructing Resiliant Communication Infrastructure for Runtime Environments / Pierre Lemarinier George Bosilca, Camille Coti, Thomas Herault, and Jack Dongarra (University of Tennessee Knoxville, Universite Paris Sud, INRIA) – 2009 P. 4-5.
4. Bacon, D.F. Compiler transformations for high performance computing / D.F. Bacon, S.L. Graham, O.J. Sharp // ACM Computing Surveys. – New York: ACM, 1994. – Vol. 26, № 4. – P. 345-420. – ISSN 0360-0300.
5. Hendren, L. J. Parallelizing programs with recursive data structures / L. J. Hendren, A. Nicolau // IEEE Transactions on Parallel and Distributed Systems. – Piscataway, New Jersey, USA: IEEE Press, 1990. – Vol. 1, № 1. – P. 35-47. – ISSN 1045-9219.
6. Herault T. A model for large scale self-stabilization. / Thomas Herault, Pierre Lemarinier, Olivier Peres,. Laurence Pilard, Joffroy Beauquier // IEEE International on Parallel and Distributed Processing Symposium, march 2007 - P. 1–10, - IPDPS 2007.
7. Bosilca, G. Constructing Resiliant Communication Infrastructure for Runtime Environments / George Bosilca, Camille Coti, Thomas Herault, Pierre Lemarinier, Jack Dongarra // University of Tennessee Knoxville University of Tennessee Knoxville, Universite Paris Sud, INRIA