A New Load Balancing Method Based on Dynamic Cluster Construction for Solution-Adaptive Finite Element Graphs on Distributed Memory Multicomputers

Maheen Islam
Computer Science and Engineering Department
East West University, Dhaka, Bangladesh.
Upama Kabir
Computer Science and Engineering Department, University of Dhaka
Mossadek Hossain Kamal
Computer Science and Engineering Department, University of Dhaka

Abstract

To solve the load imbalance problem of a solution-adaptive finite element application program on a distributed memory multicomputer, the load of a refined finite element graph can be redistributed, based on the current load of each processor. For this purpose a load-balancing algorithm can be applied to balance the computational load of each processor. In this paper, a distributed method for load balancing is proposed, which is based on the global load balancing information and current load distribution of the system. A simulation model has been developed to compare the performance of the proposed method with previously stated methods like Maximum Cost Spanning Tree Load-Balancing (MCSTLB) Method, Binary Tree Load Balancing (BTLB) Method and Condensed Binary Tree Load Balancing (CBTLB) Method. Two criteria, the execution time and the number of process migration required by different load balancing methods have been used for performance evaluation. The experimental result shows that the execution time and the number of process migration required by the proposed method is better than that of existing methods.

The finite element method is widely used for the structural modeling of physical systems. In the finite element model, an object can be viewed as a finite element graph, which is a connected and undirected graph that consists of a number of finite elements. Each finite element is composed of a number of nodes. Due to the properties of computation-intensiveness and computation-locality, to implement the finite element method on distributed memory multicomputers (Angus, Fox, Kim & Walker, 1990; Fox, Johnson, Lyzenga, Salman & Walker, 1988; Simon, 1991, p. 135; Williams, 1990; Williams, 1991, p. 457) appears as an attractive proposition.

In the context of parallelizing a finite element application program that uses iterative techniques to solve a system of equations (Aykanat, Doraivelu, Martin & Ozg†ner 1987, p. 662), a parallel program may be viewed as a collection of tasks represented by nodes of a finite element graph. Each node represents a particular amount of computation and can be executed independently. To efficiently execute a finite element application program on a distributed memory multicomputer, we need to map nodes of the corresponding finite element graph to processors of a distributed memory multicomputer in such a way that each processor has approximately the same amount of computational load and so that the communication among processors is minimized. Since this mapping problem is known to be NP-complete (Garey & Johnson, 1979), many heuristic methods have been proposed to find satisfactory suboptimal solutions (Barnard & Simon, 1994, p.101; Barnard & Simon, 1995, p. 627; Ercal, Ramanujam & Sadayappan, 1990, p.35; Fiduccia & Mattheyes, 1982, p.175; Gilbert & Zmijewski, 1987, p. 427; Gilbert, Miller & Teng, 1995, p. 418; Hendrickson & Leland, 1995, p.469; Hendrickson & Leland, 1995; Karypis & Kumar, 1995; Karypis & Kumar, 1995; Kernigham & Lin, 1970, p. 292; Simon, 1991, p.135; Williams, 1991, p. 457).

For a solution-adaptive finite element application program, the number of nodes increases discretely due to the refinement of some finite elements during the execution. This may result in load imbalance of processors. A node remapping or a load-balancing algorithm has to be performed many times in order to balance the computational load of processors while keeping the communication cost among processors as low as possible. For the load balancing approach, some load-balancing algorithms can be used to perform the load balancing process according to the current load of processors. Load-balancing algorithms are performed at run-time; their execution must be fast and efficient.

In this paper, a cluster based load-balancing method has been proposed to efficiently deal with the load imbalance problems of solution-adaptive finite element application programs on distributed memory multicomputers. When nodes of a solution-adaptive finite element graph were evenly distributed to processors by some mapping algorithms, according to the communication property of the finite element graph, we can get a processor graph from the partition. For example, Figure 1 shows a partition of a 21-node finite element graph on seven processors. The corresponding processor graph of Figure 1 is shown in Figure 2. In a processor graph, nodes represent the processors and edges represent the communication needed among processors. The weights associated with nodes and edges denote the computation and the communication costs (Chung & Liao, 1999, p.360). As all the nodes are homogeneous, they have the same finite computation cost, n, associated with them. The neighbor processors communicate with each other through message passing. The weighted edge linking the neighbor processors shows the normalized cost related with message passing.

When a finite element graph is refined during run-time, it will result in load imbalance of processors. To balance the computational load of processors, the Cluster method first builds up clusters of processors. Based on clusters, the global load balancing information is calculated by the tree walking algorithm (TWA) (Wu, 1997, p. 173). According to the global load balancing information and the current load distribution, a load transfer algorithm is performed to balance the computational load of processors and minimize the communication cost among processors.

This cluster based load balancing algorithm is considered to be run at application level which is independent of the lower layer protocols. As this load balancing process is platform independent, it can run on processors where the underlying network topology varies.

To evaluate the performance of the proposed method, it has been implemented along with three other tree- based parallel load balancing methods, the Maximum Cost Spanning Tree Load Balancing (MCSTLB) method (Chung & Liao, 1999, p.360), Binary Tree Load Balancing (BTLB) method (Chung & Liao, 1999, p.360) and Condensed Binary Tree Load Balancing (CBTLB) method (Chung & Liao, 1999, p.360). The experimental results show that the execution time and the number of process migration of an application program under a cluster- based load-balancing method is always shorter than those of the other methods.

The Parallel Load Balancing Methods

The Maximum Cost Spanning Tree Load-Balancing (MCSTLB) Method

The main idea of the MCSTLB method (Chung & Liao, 1999, p.360) is to find a maximum cost spanning tree from the processor graph that is obtained from the initial partitioned finite element graph. The MCSTLB method can be divided into the following four phases:

Phase 1: Obtain a processor graph G from the initial partition.

Phase 2: Use a similar Kruskal's (Kruskal,, 1956, p. 48) algorithm to find a maximum cost spanning tree T = (V, E) from G. There are many ways to determine the shape of T. In this method, the shape of T is constructed as follows:

1. The processor with the largest degree in V is selected as the root of T.

2. For each nonterminal processor v in T, if {u1 , …, um } are the m children of v and |u1 | ‹ |u2 | ‹… ‹ |um |, then u1 will be the leftmost child of v, u2 will be the second leftmost child of v, and so on, where |ui | is the degree of ui and i = 1, …, m. If the depth of T is greater than logM, where M is the number of processors, we will try to adjust the depth of T. The adjusted method is first to find the longest path (from a terminal processor to another terminal processor) of T. After the longest path is determined, the middle processor of the path is selected as the root of the tree and the tree is reconstructed according to the above construction process. If the depth of the reconstructed tree is less than that of T, the reconstructed tree is the desired tree. Otherwise, T is the desired tree. The purpose of the adjustment is to reduce the load balancing steps among processors.

Phase 3: Calculate the global load balancing information and schedule the load transfer sequence of processors by using the TWA (Wu, 1997, p. 173). Assume that there are M processors in a tree and N nodes in a refined finite element graph. We define N/M as the average weight of a processor. In the TWA method, the quota and the load of each processor in a tree are calculated, where the quota is the sum of the average weights of a processor and its children processors and the load is the sum of the weights of a processor and its children processors. The difference of the quota and the load of a processor is the number of nodes that a processor should send to or receive from its parent. If the difference is negative, a processor should send nodes to its parent. Otherwise, a processor should receive nodes from its parent. According to the global load balancing information, a schedule can be determined.

Phase 4: Perform load transfer (send/receive) based on the global load balancing information, the schedule, and T. Assume that processor Pi needs to send m nodes to processor Pj and let N denote the set of nodes in Pi that are adjacent to those of Pj. In order to keep the communication cost as low as possible, in the load transfer, nodes in N are transferred first. If |N| is less than m, then nodes adjacent to those in N are transferred. This process is continued until the number of nodes transferred to Pj is equal to m.

The Binary Tree Load Balancing (BTLB) Method

The BTLB method (Chung & Liao, 1999, p.360) is similar to the MCSTLB method (Chung & Liao, 1999, p.360). The only difference between these two methods is that the MCSTLB method is based on a maximum cost spanning tree to balance the computational load of processors while the BTLB method is based on a binary tree. The BTLB method can be divided into the following four phases:

Phase 1: Obtain a processor graph G from the initial partition.

Phase 2: Use a similar Kruskal's algorithm to find a binary tree T = (V, E) from G, where V and E denote the processors and edges of T, respectively. The method to determine the shape of a binary tree is the same as that of the MCSTLB method.

Phase 3: Calculate the global load balancing information and schedule the load transfer sequence of processors by using the TWA.

Phase 4: Perform load transfer (send/receive) based on the global load balancing information, the schedule, and T. The load transfer method is the same as that of the MCSTLB method.

The Condensed Binary Tree Load Balancing (CBTLB) Method

The main idea of the CBTLB method (Chung & Liao, 1999, p.360) is to group processors of the processor graph into metaprocessors. Each metaprocessor is a hypercube. The CBTLB method can be divided into the following five phases:

Phase 1: Obtain a processor graph G from the initial partition.

Phase 2: Group processors of G into metaprocessors to obtain a condensed processor graph Gc incrementally. The metaprocessors in Gc are constructed as follows: First, a processor Pi with the smallest degree in G and a processor Pj that is a neighbor processor of Pi and has the smallest degree among those neighbor processors of Pi are grouped into a metaprocessor. Then, the same construction is applied to other ungrouped processors until there are no processors that can be grouped into a hypercube. Repeat the grouping process to each metaprocessor until there are no metaprocessors that can be grouped into a higher order hypercube.

Phase 3: Find a binary tree T = (V, E) from Gc , where V and E denote the metaprocessors and edges of T, respectively. The method of constructing a binary tree is the same as that of the BTLB method.

Phase 4: Based on T, calculate the global load balancing information and schedule the load transfer sequence by using a similar TWA method for metaprocessors. To obtain the global load balancing information, the quota and the load of each processor in a tree are calculated. The quota is defined as the sum of the average weights of processors in a metaprocessor Ci and processors in children processors of Ci. The load is defined as the sum of the weights of processors in a metaprocessor Ci and processors in children metaprocessors of Ci. The difference of the quota and the load of a metaprocessor is the number of nodes that a metaprocessor should send to or receive from its parent metaprocessor. After calculating the global load balancing information, the schedule is determined as follows. Assume that m is the number of nodes that a metaprocessor Ci needs to send to another metaprocessor Cj . We have the following two cases:

1. Case 1: If the weight of Ci is less than m, the schedule of these two metaprocessors is postponed until the weight of Ci is greater than or equal to m.
2. Case 2: If the weight of Ci is greater than or equal to m, a schedule can be made between processors of Ci and Cj . Assume that ADJ denotes the set of processors in Ci that are adjacent to those in Cj . If the sum of the weights of processors in ADJ is less than m, a schedule is made to transfer nodes of processors in Ci to processors in ADJ such that the weights of processors in ADJ is greater than or equal to m. If the sum of the weights of processors in ADJ is greater than or equal to m, a schedule is made to send m nodes from processors in ADJ to those in C.

Phase 5: Perform load transfer (send/receive) among metaprocessors based on the global load balancing information, the schedule, and T. The load transfer method is similar to that of the BTLB method. After performing the load transfer process among metaprocessors, a dimension exchange method (DEM) is performed to balance the computational load of processors in metaprocessors.

Cluster-Based Load Balancing Method

The main idea of the cluster-based method is to construct an arrangement of processors, where the processors are combined into clusters. After the construction of processor cluster, the load information for each processor is collected and the load balancing algorithm is performed in such a manner that the processor can balance their load by transferring minimum number of processes and the overall load balancing time is also improved.

Phase 1: Cluster construction.

- Step 1: Divide N number of processors into N/3 number of clusters. In a cluster there might be one or two or three processors. In each case the cluster might be constructed as following:

1. Case 1: If a cluster has three nodes, then one of them is called the parent node, and the other two are called the left and right child, respectively.
2. Case 2: If a cluster has two nodes, then one of them is called the parent node, and the other is called the left child.
3. Case 3: If a cluster has only one node, then it is called the parent node.

In each cluster the children nodes send their state information to the parent node when they try to balance the load.

If there is only one cluster, then go to Phase 3.

- Step 2: Rearrange three local clusters to form a large cluster. In this large cluster, one node acts as parent and other two as left and right child respectively.

This process of constructing a large cluster is continued until there is only one large cluster.

Phase 2: Load Estimation.

Each processor in the system has varying number of processes and each process has varying amount of load. To find the average weight or Quota of a processor we have to first calculate the sum of loads of all processors and then we must divide the total sum by the number of processors of the system. Thus we obtain the quota for each processor and from the quota we calculate the high threshold and low threshold value for each processor, where

High threshold= quota + x (where x = 5% of quota)

Low threshold= quota – x (where x = 5% of quota)

Now a processor's state is defined as follows:

1. Case 1: The processor is in a normal state if its load is greater than the low threshold and less than the high threshold.
2. Case 2: The processor is in underloaded state if its load is below the low threshold
3. Case 3: The processor is in overloaded state if its load is above the high threshold

Phase 3: Load distribution

- Step 1: In this level, for each cluster, the cluster load and the cluster quota are calculated. The cluster load is defined as the sum of loads of each processor in a cluster, which is not in normal state, and the cluster quota is defined as the sum of the quota for each processor in the cluster, which is not in normal state. From the cluster quota, the high threshold and low threshold is also calculated for the cluster. Now depending on the cluster load and threshold values of the cluster, the following two cases may occur:

1. Case 1: If the cluster load is greater than the low threshold and less than the high threshold, then it is possible to balance the load of the cluster internally. For each member node of the cluster, the difference of quota and load is the number of processes that a node should send or receive from other nodes. If the difference is negative, a node should transfer the load, otherwise it should receive loads.
2. Case 2: If the cluster load is greater than the high threshold value or less than the low threshold value, then load balancing is not possible within the cluster. In this case the parent will contain the cluster load information.

If the load of all clusters in this level is balanced, then the load distribution process will be terminated. Otherwise step 1 should be repeated until a higher level large cluster exists.

- Step 2: When the largest cluster has been reached, the cluster load should be distributed among the members of the cluster, which is not in the normal state. For each member node of the cluster, the difference of quota and load is the number of processes that a node should send or receive from other nodes. If the difference is negative, a node should transfer load; otherwise it should receive loads. Then, each cluster of the next lower level distributes the load among the processors of that cluster in the same way.

This process of load distribution is repeated until any lower level cluster exists.

Experimental results

This section compares the performance of the load-balancing methods by implementing the algorithm with some simulation programs. The criteria used to evaluate the performance are execution time and the number of processes to be migrated to balance the system load.

Comparison of execution time of different load balancing methods

The execution time of different load balancing methods, with 7, 15, 25, 30, and 40 processors are shown in Table-1. From Table-1, it is evident that among MCSTLB, BTLB and CBTLB method, the execution time of CBTLB method is better than the other two. This is because the CBTLB method can reduce the size of a tree with a large ratio so that the overheads to do the load transfer among the metaprocessors are less than those of the MCSTLB and BTLB method. Thus it can reduce the load transfer time efficiently. We also observe that the execution time for the Cluster method is less than that of the CBTLB method. This is because the CBTLB method does not try to balance the load within a metaprocessor after forming the cluster. As a result a metaprocessor, which can be balanced locally, is grouped into a higher level hypercube. This makes fruitless process transfer possible and thus it will take more time to balance the load. Though in the Cluster methods, grouping is performed in each refinement, it takes less time to balance the system load.

Comparison of the number of process migration of different methods

The numbers of processes to be migrated in different load balancing methods, with 5, 7, 10, 15, 20, 25, 30, 35, 40 and 45 processors are shown in Table 2.

Conclusion

Different types of load-balancing algorithm for solution-adaptive finite element application program on distributed memory multicomputers were proposed. These are MCSTLB method, the BTLB method, the CBTLB method, and the Cluster method. In MCSTLB method, BTLB method, and CBTLB method, a logical tree (a maximum cost spanning tree for MCSTLB method, a binary tree for BTLB method, and a condensed binary tree for CBTLB method) is constructed from a processor graph. Based on the tree structure and the current load of the system, the existing method has been trying to balance the system load. But in those methods, the static nature of the logical tree makes a huge number of process migrations which consume not only time but also the communication network bandwidth.

In this paper, a new, improved group based method has been proposed to balance the load among the sites of a distributed memory multicomputer system to overcome the problems associated with the previous methods. In this method, the processors have been grouped so that the members of a group can try to balance their load within the group without knowing the states of the other processors belonging to a different group. Otherwise, when balancing the load within the group is not possible, this group tries to balance the load in a large group. Thus, in this method a process is migrated only when it finds its suitable destination. If we consider load balancing without grouping the processors in a cluster, then a huge number of messages have to be transferred among all the processors to balance their loads, as every processor will try to balance its load with every other processor. If we consider clusters consisting of two processors other than considering three, again a huge number of message passing will be required to balance the system load. So the discussion concludes that the proposed method requires fewer process migrations and less execution time than the existing methods.

To evaluate the performance of the existing load balancing methods and the proposed one, the algorithms are implemented with some simulation programs. Two criteria to do so are (i) execution time and (ii) the number of process migration of different algorithms required for an application program is used for performance evaluation. The experiment result shows that the execution time and number of process to be migrated of the proposed method is better than that of the existing methods.

References

Angus, I.G, Fox, G.C., Kim, J.S., & Walker, D.W. (1990). *Solving Problems on Concurrent Processors,* vol. 2. Englewood Cliffs, N.J.: Prentice Hall.

Aykanat, C., Ozguner, F., Martin, S., & Doraivelu, S.M. (1987). Parallelization of a Finite Element Application Program on a Hypercube Multiprocessor. *Hypercube Multiprocessor,* pp. 662-673.

Barnard, S.T., & Simon, H.D. (1994, April). Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice and Experience,* vol. 6, no. 2, pp. 101- 117.

Barnard, S.T., & Simon, H.D. (1995, February). A Parallel Implementation of Multilevel Recursive Spectral Bisection for Application to Adaptive Unstructured Meshes. *Proc. Seventh SIAM Conf. Parallel Processing for Scientific Computing,* pp. 627-632, San Francisco.

Chung, Y.C., & Liao, C.J. (1999). Tree-Based Parallel Load Balancing Methods for Solution-Adaptive Finite Element Graphs on Distributed Memory Multicomputer. *IEEE Transaction on Parallel and Distributed Systems,* vol. 10, No. 4, pp. 360-370.

Ercal, F., Ramanujam, J., & Sadayappan, P. (1990). Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning. *J. Parallel and Distributed Computing,* vol. 10, pp. 35-44.

Fiduccia, C.M., & Mattheyes, R.M. (1982). A Linear-Time Heuristic for Improving Network Partitions. *Proc. 19th IEEE Design Automation Conf.,* pp. 175-181.

Fox, C., Johnson, M., Lyzenga, G., Otto, S., Salman, J., & Walker, D.W. (1988). *Solving Problems on Concurrent Processors,* vol. 1. Englewood Cliffs, N.J.: Prentice Hall.

Garey, M.R., & Johnson, D.S. (1979). *Computers and Intractability, A Guide to Theory of NP-Completeness.* San Francisco: Freeman.

Gilbert, J.R., & Zmijewski, E.(1987). A Parallel Graph Partitioning Algorithm for a Message-Passing Multiprocessor. *Int'l J. Parallel Programming,* vol. 16, no. 6, pp. 427-449, 1987.

Gilbert, J.R., Miller, G.L., & Teng, S.H. .(1995). Geometric Mesh Partitioning: Implementation and Experiments. *Proc. Ninth Int'l Parallel Processing Symp.,* Santa Barbara, Calif., pp. 418-427.

Hendrickson, B., & Leland, R. (1995). An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations. *SIAM J. Scientific Computing,* vol. 16, no. 2, pp. 452-469.

Hendrickson, B., & Leland, R. (1995). "An Multilevel Algorithm for Partitioning Graphs," *Proc. Supercomputing* '95, Dec. 1995.

Karypis, G., & Kumar, V. (1995). Multilevel k-way Partitioning Scheme for Irregular Graphs. Technical Report 95-064, Dept. of Computer Science, Univ. of

Minnesota, Minneapolis.

Karypis, G., & Kumar, V. (1995). *MeTiS—Unstructured Graph Partitioning and Spares Matrix Ordering System.* Univ. of Minnesota.

Kernigham, B.W., & Lin, S.(1970, February). An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Systems Technology J.*, vol. 49, no. 2, pp. 292-370.

Kruskal, J.B. (1056). On the Shortest Spanning Subtree of a Graph and the Traveling Salseman Problem. *Proc. AMS,* vol. 7, pp. 48-50.

Simon, H.D. (1991). Partitioning of Unstructured Problems for Parallel Processing. *Computing Systems in Eng.*, vol. 2, nos. 2/3, pp. 135- 148.

Williams, R.D. (1990). DIME: *Distributed Irregular Mesh Environment.* California Inst. of Technology.

Wu, M.Y. (1997, February). On Runtime Parallel Scheduling for Processor Load Balancing, *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 2, pp. 173-186.

Williams, R.D. (1991, October). Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations," *Concurrency: Practice and Experience,* vol. 3, no. 5, pp. 457-481.

Table 1

The execution time in seconds of different load balancing methods with different number of processors.

	No. of processors				
Methods	7	15	25	30	40
MCSTLS	1.500549	1.500549	1.500549	1.500549	1.554396
BTLB	1.500549	1.500549	1.500549	1.500549	1.500549
CBTLB	1.103846	1.10549	1.100000	1.154396	1.100000
Cluster	0.659340	0.692308	0.714286	0.714286	0.714286

Table 2

Number of process migration of different load balancing method for different load samples with different number of processors.

	No. of processors									
Methods	5	7	10	15	20	25	30	35	40	45
MCSTLB	278	440	719	1270	1841	2370	3113	3550	4135	4562
BTLB	301	509	824	1460	2156	2937	2156	4068	4810	4204
CBTLB	432	782	1389	2286	3509	4426	5556	6617	7355	8126
Cluster	112	166	277	312	464	494	577	601	720	1223

Figure 1: A partition of 21-node finite element graph on 7 processors.

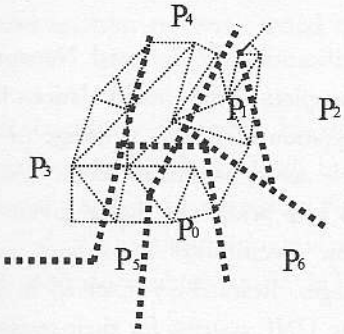
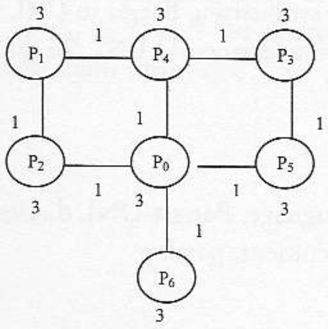


Figure 2: The corresponding processor graph of Figure 1.


3
1
3
1
3
P_1
P_4
P_3
1
1
1
P_2
P_0
P_5
1
1
3
3
3
1
P_6
3