A Fast Load Balancing for Multicomputer Systems

Hwayung Rim
School of Elec. & Computer Eng.
Kyungho University
Kyunggido, Korea
ackyung@khu.ac.kr

Ju-wook Jang
Dept. of Electronic Eng.
Sogang University
Seoul, Korea
jjang@ccs.sogang.ac.kr

Sungchun Kim
Dept. of Computer Science
Sogang University
Seoul Korea
ksc@arqlab1.sogang.ac.kr

Abstract

Dynamic load balancing methods for multicomputers of hypercubes, mesh, and tree are reconsidered for possibility of hiding communication overheads involved in transfer of load units. Previous methods perform transfer in a sequential way, leaving many links idle in some phases. The proposed method is either coalescing some phases of balancing to overlap the transfer of load units on different links or dividing each phase into steps to pipeline the transfer of load unit by unit for maximum utilization of links. The our methods are shown via simulation to result in 15% to 65% reduction to known methods, depending on initial load distribution.

Keywords: dynamic load balancing, communication overheads, multicomputers

1 Introduction

The balancing of loads has been an important matter in parallel and distributed processing which greatly affects the speedup in processing time. The underlying network topology or architecture, communication overheads involving transfer of data between processors, the size of load, the size of smallest unit of load and etc. should be carefully considered to obtain an efficient solution to balancing problem.

In this paper, we define a new measure for communication overheads involved in balancing. We are concerned with only the link in which the maximum number of units in a phase is transferred. If $T_{max}$ represents the maximum transmission time in phase $i$, $\sum_i T_{max}$ is our measure of total communication overheads. Our methods aim to reduce $\sum_i T_{max}$ by hiding part of communication overheads. Transfers on different links in different balancing phases are combined (or performed together) into a new single phase as far as some conditions are not violated. One of the conditions is that each processor involved has enough units than the number of units to be transferred from itself before joining the new phase. After the phase, load distribution will change and a new subset of processors is formed to start the next new phase. This will continue until all the processors have exactly the number of units determined in advance. Simulation results show as much as 30% saving in communication overheads depending on initial distribution of load, number of processors and etc.

We call this as method1 since transfers on different links in different (and/or same) links are overlapped if certain conditions are met. There are some cases where the above method1 technique is not quite effective. For some severely uneven load distribution, many processors can not start sending until they have received enough. What we propose is a cascaded transfer which allows each processor to initiate (or continue) a transfer as far as it has at least 1 unit. It stops sending out only when it has zero unit and resumes as soon as it has received 1 unit from a neighboring processor. We call this as method2.

We developed techniques to embed the method1 and the method2 techniques into dynamic balancing on multicomputers of hypercube, mesh and tree topologies.
Known balancing methods, MWA(mesh walking algorithm) and TWA(tree walking algorithm)[7] are chosen to illustrate the efficiency of the our techniques. Simulation results show as much as 70% reduction in communication overheads, depending on input distribution. One major contribution of our paper is in developing techniques to hide communication overheads involved in dynamic load balancing which, we believe, received little attention.

The paper is organized as follows. Section 2 describes our methods of hiding communication overheads. Section 3 provides simulation results and Section 4 concludes this paper.

2 Communication overheads on multi-computer systems

In [7], task-hop is defined as a measure of the communication overheads for transfer of task units(or loads) in balancing phases. If \( T_k \) denotes the number of units transferred through the link \( k \), \( \sum_k T_k \) represents the task-hop. It is shown via simulation that the CWA outperforms the DEM in terms of average task-hop as reported in [7]. However, if we assume that execution of loads is held during balancing, some links are idle in some phases of balancing. This observation leads us to another measure of communication overheads. In each phase our new measure concerns only the link in which the maximum number of units are transmitted. Transmission time in other links can be hidden.

If \( T_{max_i} \) represents the maximum transmission time in phase \( i \), \( \sum_{i} T_{max_i} \) is our measure of communication overheads. We introduce new techniques for hiding communication overheads involved in balancing. Our techniques aim to reduce \( \sum T_{max_i} \). If we assume execution is being held during balancing, our techniques will speed up the processing by reducing the balancing overheads. Assumed here is that links not involved in balancing phases are idle and all loads are independent.

2.1 Communication overhead of our method1

Our method1 technique allows transmission on links aligned with different dimensions to be overlapped except when a processor involved in overlapping should delay sending out load until it has received enough. The colored circles in Figure 1 (b) are the processors which can send out load because they have initially enough load to send as shown in Figure 1 (a). For example, the processor at the left bottom has 13 units and 5 units to go. So, it can start sending 5 units. The circled numbers of Figure 1 (b) represent the number of units after sending (and/or receiving) is done. At this time, some new processors have received enough to initiate sending as shown in Figure 1 (c). And transfer as shown in Figure 1 (d) can be initiated after that.

\[ T_{max_1} = 12T_{comm} \]
\[ T_{max_2} = 11T_{comm} \]
\[ T_{max_3} = 8T_{comm} \]

Thus, total communication overheads, \( \sum_{i=1}^{3} T_{max_i} \), is \( 31T_{comm} \) for this example.

Compared with the above MWA method[7], our method1 results in around 30% saving in communication overheads. However, the saving will differ depending on initial load distribution but will not increase the communication overheads under any circumstances.

Method1 is also applied to the TWA example shown in Figure 2. While it takes \( 13T_{comm} \) to balance the tree using TWA, method1 shown in Figure 2 reduces the communication overheads to \( 11T_{comm} \).

Figure 3 shows the effect of method1 on the hypercube. The effective transmission time, \( \sum_i T_{max_i} \), is reduced to \( 3T_{comm} \). For this specific example, method1 saves more than 50% of communication overheads.
2.2 Communication overhead of our method

There are some cases where the method1 technique is not quite effective. Consider balancing in Figure 10 where one processor has 15 units while all other processors have only one unit of load. Figure 4 (a) shows initial load distribution and the number of units to be sent out from each processor. Here processor 001 is supposed to send 3 units to processor 011, but it cannot initiate sending since it has only one unit. Processor 001 should wait for 7 units from processor 000, which prevents overlapping.

We introduce method2 method to circumvent this problem. A processor does not have to wait for whole number of units. It sends out a unit of load immediately as soon as it receives one. Continuing in our way, balancing is complete in 7 steps taking \( T_{\text{comm}} \). With method2, more links will be busy at a time. To increase utilization of links even further, we employ an interleaving scheme when a processor has multiple receivers.

Method2 in mesh may be ineffective for some initial load distributions. Consider Figure 1 again. Let \( P_i, 1 \leq i \leq 16 \), denote the \( i \)-th processor in a row major ordering of the mesh. \( P_7 \) is located at the intersection of the second row and the third column. \( P_7 \) has 7 units of load in this example. Initially, the \( P_7 \) can not initiate sending out as shown in Figure 1 (b). This is because, before phase 1, it has only 7 units which are less than 8 units to go. Thus \( P_7 \) is not allowed to send out but allowed only to receive 10 units from \( P_{11} \). We propose a cascaded transfer which allows each processor to initiate(or continue) as far as it has at least 1 unit. It stops sending out only when it has zero unit and resumes as soon as it has 1 unit from a neighboring processor. Figure 5 (b) shows step 1 where at most 1 unit is transferred in each link. Note that 15 links are used for transfer during step 1 (We use 'step' instead of 'phase' for each cascaded transfer to imply that a step is for 1-unit transfer while multiple units are transferred in a phase ). Compare this with Figure 1 (b) where 10 links are used for transfer.

Considering links as valuable resource for communication, the more we utilize them at a given time slot, the more we benefit from the links. Method2 is proposed here as a way to utilize the links to the maximum. Since \( T_{\text{max}} = T_{\text{comm}} \) for all \( i \), the total communication overhead is \( \sum_{i=1}^{12} T_{\text{max}} \), is \( 12T_{\text{comm}} \) which is far smaller than \( 3T_{\text{comm}} \) for method1 or \( 4T_{\text{comm}} \) for MWA [7]. Proposed method2 technique is applied also to the tree in Figure 2. Continuing in this way, balancing is complete in 6 steps taking \( 6T_{\text{comm}} \).

3 The simulation

SLAM II tool[1] is used to estimate for our simulation. To quantify the effect of unevenness in initial load distribution, we define imbalance as the percentage of processors which have \( k \) times more load than other processors where \( k \) is any positive integer. Thus the smaller the imbalance is the severer the unevenness is. The values of imbalance chosen for experiment are 20%, 40%, 60%, and 80%, among which 20% and 80% represent most and least uneven initial distribution, respectively.

In this specific simulation, we repeated simulations with \( k \) having values of 30, 50, 70, 100, 400, 700, 1000 and averaged the results.
Figure 5. A running example of MWA using the method2

Figure 6 (b) shows the effect of method1 and method2 method on DEM. Again, the method2 has proven powerful for severely uneven distribution. Figure 7 shows simulation results for MWA, proposed MWA of method1 and proposed MWA of method2. MWA of method1 shows around 50% to 60% saving and MWA of method2 gives further savings of around 10% to 20%. Figure 19 compares communication overheads of proposed TWA of method1 and TWA of method1 against TWA[7]. TWA of method2 is shown to improve the TWA by as much as 65%.

4 Conclusion

Dynamic balancing methods for multicomputers are reconsidered for possibility of hiding communication overheads involved in transfer of load units during balancing phases. Previous methods, DEM (dimension exchange method), MWA (mesh walking algorithm), and TWA (tree walking algorithm) perform transfer in a sequential way, leaving many links idle in some phases. Our method1 techniques collect information about transfers in all the links in all phases and determine which can be overlapped with which. We reconstruct phases such that all the transfer actions in each phase can be overlapped. We also introduce method2 in which each phase is split into steps and one unit is transferred to its neighbor processor in a step. Each processor doesn’t have to wait for all the units to be
sent to its neighbor before starting sending out. It can start as soon as it has a unit. It stops with zero unit and resumes after receiving one. The utilization of links is larger than when method1 alone is employed. Combined together, our method2 technique is shown via simulation to contribute 15% to 70% reduction(hiding) to known techniques[3, 7], depending on initial load distribution.

References


