Parallel Processing of Adaptive Meshes with Load Balancing

Sajal K. Das and Daniel J. Harvey

Department of Computer Sciences
University of North Texas
P.O. Box 311366
Denton, TX 76203-1366
E-mail:{das,harvey}@cs.unt.edu

Rupak Biswas

MRJ Technology Solutions
NASA Ames Research Center
Mail Stop T27A-1
Moffett Field, CA 94035-1000
E-mail: rbiswas@nas.nasa.gov

Abstract

Many scientific applications involve grids that lack a uniform underlying structure. These applications are often also dynamic in nature in the sense that the grid structure significantly changes between successive phases of execution. In parallel computing environments, mesh adaptation of unstructured grids through selective refinement/coarsening has proven to be an effective approach. However, achieving load balance while minimizing interprocessor communication and redistribution costs is a difficult problem. Traditional dynamic load balancers are mostly inadequate because they lack a global view of system loads across processors. In this paper, we present a novel, general-purpose load balancer that utilizes symmetric broadcast networks (SBN) as the underlying communication topology. The experimental results on the IBM SP2 demonstrate that performance of the SBN-based load balancer is comparable to results achieved under PLUM, a global load balancing environment created to handle adaptive unstructured applications.

1 Introduction

Mesh partitioning is a common approach to processing many scientific applications in parallel. These applications are generally modeled discretely using a mesh (or grid) of vertices and edges. For maximum (parallel) efficiency, the computational workloads on the processors have to be balanced and the number of edges that are cut (and hence the overall interprocessor communication cost at runtime) needs to be minimized [11, 13]. For this purpose, each vertex is usually assigned a weight that indicates the amount of computation required to process it. Similarly, each edge in the mesh has an associated weight indicating the amount of interaction between adjacent vertices. To achieve load balance dynamically, portions of the mesh have to be migrated among the processors during the course of a computation. Thus, in a multiprocessing environment, the vertex weight contains an additional component that models the cost of redistributing the vertex from one processor to another. These weights are used to minimize the data redistribution cost during the remapping phase.

In adaptive meshes, the grid topology changes during the course of a computation. Traditionally, this class of problems is processed by load balancing the mesh after each adaptation. A number of partitioners designed for this purpose has been proposed in the literature [9, 10, 11, 14, 18, 20, 23]. A majority of the successful partitioners are based on a multilevel approach that has proven to be extremely effective in producing good partitions at reasonable execution cost. In this approach, the grid graph is first contracted to a small number of vertices and edges, and the coarsened graph is then partitioned and successively refined using a Kernighan-Lin replacement algorithm [15]. For an excellent survey on other partitioning methods, refer to [1].

Although several dynamic load balancers have been proposed for multiprocessor platforms [3, 4, 6, 12, 16, 21, 22], most of them are inadequate for adaptive mesh applications because they lack a global view of system loads across processors. Also, job migration in these approaches does not take into account the structure of the adaptive grid. In this paper, we overcome these deficiencies by modifying the load balancer proposed earlier by us [6]. Our load balancer makes use of a symmetric broadcast network (SBN) which is a robust and topology-independent communication pattern among processors [7]. The proposed SBN-based load balancer can be classified as:

- **Adaptive:** Processing automatically adjusts to the number of jobs that are queued.
- **Decentralized:** Responsibility for load balancing is shared by all the nodes of the system. Any

node can initiate load balancing activity.

- **Stable:** Excessive load-balancing traffic does not burden the network, especially under extremely light or heavy system loads.
- **Effective:** System performance does not degrade because of load balancing activities.

In an earlier work [6], we have shown that our approach achieves superior load balance when compared to other popular load balancing techniques such as Random, Gradient, Receiver Initiated, Sender Initiated, and Adaptive Contracting.

Recently, experiments that measure the effectiveness of load balancing adaptive meshes have been conducted using an automatic portable environment, called PLUM [17], that was developed at NASA Ames Research Center. PLUM uses a novel balancing strategy consisting of two separate phases: repartitioning and remapping. After each mesh adaptation step, the computational grid is globally repartitioned if the workload distribution is unacceptable. The new partitions are then reassigned among the processors in a way that minimizes the cost of data movement. Only if the remapping cost is compensated by the computational gain that would be achieved with balanced partitions, is the necessary data appropriately redistributed. Otherwise, the new partitioning is discarded. Notice that data is not physically migrated unless the cost estimates indicate that doing so is beneficial.

The SBN-based load balancer differs from PLUM in several ways as discussed below:

- Processing is temporarily halted under PLUM while the load is being balanced. During the suspension, a new k-way partitioning is generated and data is redistributed among the nodes of the network. The SBN approach, on the other hand, allows processing to continue while the load is dynamically balanced. This feature also allows for the possibility of utilizing latency tolerance techniques to hide the communication and redistribution costs during processing.
- Under PLUM, suspension of processing and subsequent repartitioning does not guarantee an improvement in the quality of load balance. If it is determined that the estimated remapping cost exceeds the expected computational gain that is to be achieved by a load balancing operation, processing continues using the original mesh assignment. This could result in unnecessary idle time. In contrast, the SBN approach will always result in improved load balance among processors.
- PLUM redistributes all necessary data to the appropriate nodes immediately before processing continues. SBN, however, distributes in a "lazy" manner. Data is migrated to a processor only when it is ready to process the data, thus reducing redistribution and communication overhead.

We have performed extensive experiments on an IBM SP2 to compare the performance of our SBN load balancer to the results obtained under PLUM in [2]. The results demonstrate that the proposed approach achieves excellent load balance and also significantly reduces the redistribution cost compared to those obtained by using the `PMeTiS` or `DMeTiS` partitioners under PLUM. However, the edge cut percentages are higher than those for `PMeTiS`, indicating that the SBN strategy reduces the redistribution cost at the expense of a higher communication cost. For example, with a network of 32 processors, the redistribution cost using the SBN strategy is approximately half of the cost incurred under PLUM. However, total communication is almost doubled.

This paper is organized as follows. Section 2 reviews the definition of SBN and a basic load balancing algorithm. Section 3 describes the modifications made to incorporate a global view needed for adaptive mesh applications. Section 4 presents an overview of PLUM and the partitioners operating under that environment that are used for comparisons. Section 5 presents experimental results and a comparative performance analysis. Section 6 concludes the paper.

2 Symmetric Broadcast Networks

A *symmetric broadcast network* (SBN), proposed by Prasad and first presented in [7], defines a communication pattern (logical or physical) among the P processors in a multicomputer system. This communication pattern can be efficiently embedded into different parallel architectures in a topology-independent manner [5, 8]. Let us first give a brief overview of SBN and outline how to use it for load balancing.

Definition 1 *An SBN of dimension $d \geq 0$, denoted as SBN(d), is a $(d+1)$-stage interconnection network with $P = 2^d$ processors in each stage. It is constructed recursively as follows. A single node forms the basis network SBN(0). For $d > 0$, an SBN(d) is obtained from a pair of SBN($d-1$)s by adding a communication stage in the front with the following additional interprocessor connections:*

- *Node i in stage 0, is made adjacent to node $j = (i + P/2) \bmod P$ of stage 1; and*

- *Node j in stage 1 is made adjacent to the node in stage 2 which was the stage 0 successor of node i in SBN*$(d-1)$.

Figure 1 depicts an SBN(2), recursively constructed from two SBN(1)s.

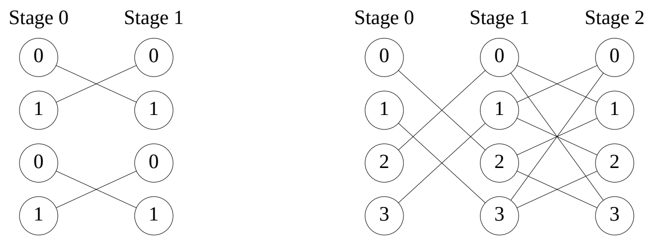

Figure 1: Construction of SBN(2) from a pair of SBN(1)s. The new connections are shown by solid lines and the original connections by dashed lines

By definition, each node in every stage s, for $1 \leq s \leq d-1$, of SBN(d) has exactly two successors; whereas a node in stage 0 has only one successor and a node in stage d has no successor. The SBN(d) defines unique communication patterns (or broadcast trees) among the nodes in the network. Precisely, for any source node or root x at stage 0, where $0 \leq x < P$, there exists a unique broadcast tree T_x of height $d = \log P$ such that each of the 2^d nodes appears exactly once.

Lemma 1 *Let n_x^s be a node at stage s in the broadcast tree T_x having the root node x (at stage 0), where $0 \leq x < P$. Then $n_x^s = n_0^s \oplus x$, where $\oplus$ is the exclusive-OR operator, thus leading to $T_x = T_0 \oplus x$.*

Thus, all SBN communication patterns can be derived from the template tree with node 0 as the root. As an example, consider two communication patterns T_0 and T_5 in SBN(3) as shown in Figs. 2(a) and 2(b) respectively. By our convention, n_0^s denotes a node at stage s in Fig. 2(a) while n_5^s is the corresponding node in Fig. 2(b). Furthermore, $n_5^s = n_0^s \oplus 5$.

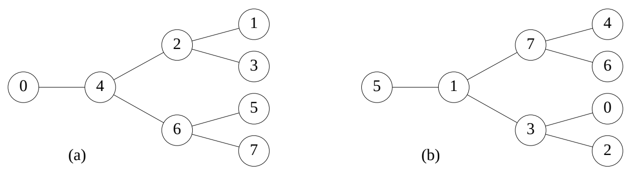

Figure 2: Examples of SBN communication patterns in SBN(3)

The predecessor and successors of each node can be uniquely defined by specifying the root x and the communication stage s so that messages from x can be appropriately routed to the other nodes.

Lemma 2 *The predecessor and successors of node n_0^s can be computed as:*

$$Predecessor = (n_0^s - 2^{d-s}) \vee 2^{d-s+1}.$$

$$Successor\_1 = n_0^s + 2^{d-s-1}, \qquad for\ 0 \leq s < d.$$

$$Successor\_2 = n_0^s - 2^{d-s-1}, \qquad for\ 1 \leq s < d.$$

2.1 SBN-Based Load Balancing

This section outlines the basic SBN-based load balancer, proposed by us in [6], which processes two types of messages. The first type is a *load balancing* message that is broadcast when a node n determines that the number, `QLen`(n), of locally queued jobs falls below the minimum threshold, `MinTh`. A load balancing message will also be broadcast if `QLen`(n) exceeds the maximum threshold, `MaxTh`, or if distributing the excess jobs will result in other processors exceeding `MaxTh`. As the load balancing message passes from one node to another, values for the global number of jobs queued (`GLen`) and the average number of jobs queued per node or the system load level (`SysLL`) are computed.

The second type of messages is called *job distribution* messages which are used to distribute jobs when `QLen`$(n) >$ `MaxTh`. Distribution messages are also used to complete the load balancing process. After the `SysLL` value is calculated, a distribution message is broadcast through the network when jobs are routed to the lightly loaded processors and the threshold values (`MinTh` and `MaxTh`) are updated. As a result, the workload at all nodes is balanced.

3 Modified SBN Load Balancer

In order to provide a global view of the system and make it effective for adaptive mesh applications, we have made a series of modifications to the basic SBN load balancer [6], just outlined. These modifications are described in the following subsections.

3.1 Weighted Queue Length

The local queue length, `QLen`(p), as used by the basic SBN approach is assumed to be an estimate of the amount of processing to be performed by p. However, this parameter is inadequate for situations where the mesh is adapted because it does not accurately reflect the total processing load at p. To achieve a better balance, we must take into account the system variables like computation, communication, and redistribution costs, that affect the processing of a local queue. Therefore, we define a new metric called weighted queue length, $\texttt{QWgt}(p)$, to estimate the total time required to completely service the vertices in the local queue of p.

Let Wgt^v be the computational cost to process a vertex v, $Comm_p^v$ be the communication cost to interact with the vertices adjacent to v but whose data sets are not local to p, and $Remap_p^v$ be the redistribution cost to copy the data set for v to p from another processor. These factors vary widely from one vertex to another in a given mesh. $\texttt{QWgt}(p)$ is defined as:

$$\texttt{QWgt}(p) = \sum_{v=1}^{\texttt{QLen}(p)} (Wgt^v + Comm_p^v + Remap_p^v).$$

Clearly, if the data set for v is already assigned to p, no redistribution cost is incurred, i.e., $Remap_p^v = 0$. Similarly, if the data sets of all the vertices adjacent to v are also already assigned to p, the communication cost, $Comm_p^v$, is 0.

The threshold values, MinTh and MaxTh, must now be based on $\texttt{QWgt}(p)$ instead of $\texttt{QLen}(p)$ to accommodate the redefinition of processing load. In our experiments, MinTh, was set to reflect a 0.1 second processing load.

3.2 Weighted System Load

The weighted system load level is computed as:

$$\texttt{WSysLL} = \left\lceil \frac{1}{P} \sum_{i=1}^{P} \texttt{QWgt}(i) \right\rceil,$$

where P is the total number of processors used.

Assuming that the load is perfectly balanced among the processors, WSysLL estimates the time required to process the mesh and reflects the processing, communication, and redistribution costs in the current mesh-to-processor assignment. Hence, a global view of the system is captured that is otherwise not reflected using SysLL in the basic SBN load balancer.

3.3 Prioritized Vertex Selection

The basic SBN balancing approach does not consider the mesh connectivity when selecting vertices to be processed. Therefore, boundary vertices that could otherwise be migrated for more efficient execution, are as likely to be executed locally as internal vertices. The modified SBN load balancer takes advantage of the underlying mesh structure and defers execution of boundary vertices as long as possible.

Thus, the next queued vertex is selected for execution so as to minimize the overall cut size of the adapted mesh. A priority min-queue is maintained for this purpose, where the priority of a vertex v in processor p is given by $(Comm_p^v + Remap_p^v)/Wgt^v$. Therefore, vertices with no communication and redistribution costs are executed first. Those with low communication or redistribution overhead relative to their computational weight are processed next. Conceptually, internal vertices are processed before those on partition boundaries.

3.4 Differential Edge Cut

For balancing the system load among processors, an optimal policy for vertex migration needs to be established. When vertices are being moved, assume that processor p is about to reassign some of its vertices to another processor q. The modified SBN load balancer running on p randomly picks a subset of vertices from those queued locally. For each selected vertex v, the differential edge cut[1], $\Delta\texttt{Cut}$, is calculated as follows:

$$\Delta\texttt{Cut} = Remap_q^v - Remap_p^v + 2 \times (Comm_q^v - Comm_p^v).$$

If $\Delta\texttt{Cut} > 0$, it is normalized as $\Delta\texttt{Cut}/Wgt^v$.

When a vertex is reassigned from one processor to another, the change in the communication cost must be applied to both processors. Hence, the change in the communication cost is doubled in the $\Delta\texttt{Cut}$ formula. The parameters $Remap_p^v$ and $Remap_q^v$ will either be 0 or equal to the redistribution cost of moving the data for v from p to q. For example, assume $p = 3$, $q = 6$, the data for v reside on $p = 1$, and its redistribution cost is 8. In this case, $Remap_p^v = Remap_q^v = 8$. Similarly, if the data for v resides on $p = 3$, then $Remap_p^v = 0$ but $Remap_q^v = 8$.

A positive $\Delta\texttt{Cut}$ indicates that an increase in communication and redistribution costs will result if v is migrated from p to q. Therefore, the formula favors migrating vertices with the smallest increase in communication cost per unit computational weight. In contrast, negative $\Delta\texttt{Cut}$ values indicate a reduction in communication and redistribution costs, hence favoring the migration of vertices with the largest absolute reduction in communication and redistribution costs.

Once $\Delta\texttt{Cut}$ is calculated for all the randomly chosen vertices, the vertex MinV with the smallest value of $\Delta\texttt{Cut}$ is chosen for migration. Next, following a breadth-first search, the SBN balancer selects the vertices adjacent to MinV that are also queued locally for processing at p. The breadth-first search stops either when no adjacent vertices are queued for local processing at p, or if a sufficient number of vertices have been found for migration. If more vertices still need to be migrated, another subset of vertices are randomly chosen and the procedure is repeated. This migration

[1]Here we are deviating from the usual definition of edge cut to account for the dynamic nature of the SBN load balancer.

policy strives to maintain or improve the cut size during the execution of the load balancing algorithm. In contrast, the original SBN algorithms do not consider cut size and hence are likely to experience larger cut sizes as execution proceeds.

3.5 Data Redistribution Policy

The redistribution of data is performed in a "lazy" manner. Namely, the data set for a given vertex v in a processor p is not moved to q until the latter processor is about to execute v. Furthermore, the data sets of all vertices adjacent to v that are assigned to q are migrated as well. This policy greatly reduces both the redistribution and communication costs by avoiding multiple migrations of data sets and having resident all adjacent vertices that are assigned to processor q while v is being processed.

We implement data migration by broadcasting a job migration message when a vertex is about to be processed and its corresponding data set is not resident on the local processor. A locate-message is then broadcast to indicate the new location of the data set.

This policy is expected to maximize the number of adjacent vertices that are local when a grid point is processed. Hence, by considering the underlying mesh structure, the communication overhead is reduced compared to that for the basic SBN algorithm.

4 The PLUM Environment

We experimentally compare the performance of our SBN-based load balancer to PLUM [17] – a portable and parallel load balancing framework for adaptive unstructured grids. For the sake of completeness, the features of PLUM are summarized below.

Figure 3 provides an overview of PLUM. After an initial partitioning, a `Solver` executes several iterations of the application. When the grid-to-processor mapping becomes unbalanced due to mesh adaptation, PLUM gains control to determine if the workload among the processors has become unbalanced and to take appropriate action if so required. Mesh repartitioning and processor reassignment are then performed. If the estimated remapping cost exceeds the expected computational gain to be achieved, execution continues without remapping. Otherwise, the grid is remapped among the processors before the computation is resumed.

As the computational mesh is adapted and the processor workloads need to be rebalanced, PLUM can use any general purpose partitioner. In [2], two state-of-the-art partitioners, `PMeTiS` [14] and `DMeTiS` [20], were used. Both partitioners are parallelized and highly optimized for maximum efficiency, and have

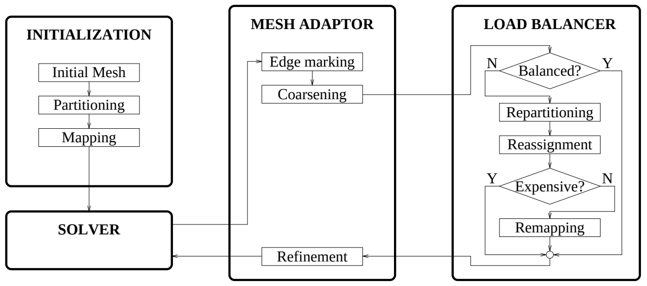

Figure 3: Overview of PLUM, a framework for globally load balancing parallel adaptive computations

proven effective for adaptive grids. `DMeTiS` is a diffusive scheme designed to modify existing partitions, while `PMeTiS` is a global partitioner that makes no assumptions on how the mesh is initially distributed.

Both `DMeTiS` and `PMeTiS` are k-way multi-level algorithms that operate in three phases: (i) in the initial coarsening phase, the original mesh, M_0, is reduced by collapsing adjacent vertices or edges through a series of smaller and smaller meshes to M_k, such that M_k has a sufficiently small number of vertices; (ii) in the partitioning phase, the mesh workload is balanced among the processors and the edge cut size is minimized; and (iii) in the projection phase, the partitioned mesh M_k is gradually restored to its original size M_0.

5 Experimental Study

The SBN-based load balancing algorithm has been implemented using MPI on the wide-node IBM SP2 located at NASA Ames Research Center, and tested with actual workloads obtained from adaptive calculations. The computational mesh used for the experiments simulates an unsteady environment where the adapted region is strongly time-dependent. This goal is achieved by propagating a simulated shock wave through the initial mesh shown in Fig. 4. The test case is generated by refining all elements within a cylindrical volume moving left to right across the domain with constant velocity, while coarsening previously-refined elements in its wake. Performance is measured at nine successive adaptation levels. The weighted sum of vertices increased from 50,000 to 1,833,730 over the nine levels of adaptation. This test case was chosen so that the results could be compared to those compiled in [2] using the PLUM environment.

5.1 Performance Metrics

The following metrics are chosen to evaluate the effectiveness of the SBN load balancer when processing an unsteady adaptive mesh. Recall that v denotes a vertex to be processed and P is the number of processors.

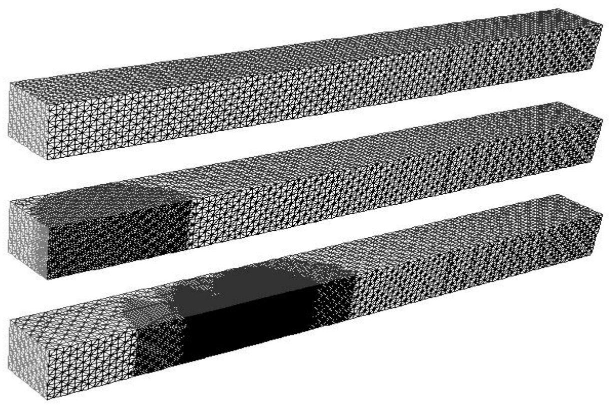
Figure 4: Initial and adapted meshes (after levels 1 and 5) for the simulated unsteady experiment

- **Maximum redistribution cost:** The goal is to capture the total cost of packing and unpacking data, separated by a barrier synchronization. Since a processor can either be sending or receiving data, the overhead of these two phases is modeled as a sum of two costs in this metric:

$$\texttt{MaxSR} = \max_{p \in P}\Big\{ \sum_{v \text{ sent from } p} Remap_p^v \Big\} + \max_{p \in P}\Big\{ \sum_{v \text{ recv by } p} Remap_p^v \Big\}.$$

  Since MaxSR pertains to the processor that incurs the maximum redistribution cost, a reduction in the total data redistribution overhead can be guaranteed by minimizing MaxSR.
- **Load imbalance factor:** It is formulated as:

$$\texttt{LoadImb} = \max_{p \in P} \texttt{QWgt}(p) \;/\; \texttt{WSysLL}.$$

  This factor should be as close to unity as possible.
- **Cut percentage:** The runtime interaction between adjacent vertices residing on different processors is represented by this metric as:

$$\texttt{Cut\%} = 100 \sum_{p \in P} \sum_{v \text{ assigned to } p} Comm_p^v \Big/ \sum_{e \text{ in mesh}} Edge^e,$$

  where $Edge^e$ is the weight of edge e in the adaptive mesh. The Cut% value should be as small as possible.

  Pre-Exec Cut% in Tables 1, 2, and 3 initially projects the mesh edge cut before processing an adaptation level but after the previous adaptation level has been processed.

  Post-Exec Cut% is the actual cut realized after processing a given adaptation level.

Table 1 presents the results of processing the adaptive mesh with the SBN load balancer. Tables 2 and 3 respectively chart the results obtained using the PMeTiS and DMeTiS partitioners within the PLUM environment. Note that Tables 2 and 3 do not show results corresponding to all values of P = 2, 4, 8, 16, and 32. We have included only those data sets that were available to us.

5.2 Summary of Results

The SBN-based approach achieves excellent load balance. For example, LoadImb = 1.02 for $P = 32$ (see Table 1). When $P \leq 8$, an ideal load imbalance factor of 1.00 is achieved for most of the adaptation levels. In contrast, this factor using PMeTiS and DMeTiS under the PLUM environment is 1.04 and 1.59 respectively (cf. Tables 2 and 3).

The MaxSR metric indicates the amount of redistribution cost incurred while processing the adaptive mesh. The SBN "lazy" approach to migration of vertex data sets produces significantly lower values than those achieved by PMeTiS or DMeTiS under PLUM. For example, for $P = 32$, Table 1 shows MaxSR $= 28,031$, which is significantly less than the corresponding value in Table 2 (Maxsr $= 63,270$) and in Table 3 (MaxSR $= 62,542$).

Table 1 shows an SBN cut percentage that is almost triple compared to those reported by PMeTiS (30.58% compared to 10.94% for $P = 32$). This difference in the cut percentage is significantly lower when compared to the results obtained with DMeTiS (30.58% compared to 20.30%).

In conclusion, these experimental results demonstrate that the proposed SBN-based dynamic load balancer is effective in processing adaptive mesh applications, thus providing a global view across processors. In many mesh applications in which the cost of data redistribution dominates the cost of communication and processing, the SBN balancer would be preferred.

6 Future Work

Currently, we are experimenting on the SGI Origin 2000 (a distributed shared-memory architecture) to test the consistency of our results and implement additional performance refinements to reduce communication cost. It would also be interesting to apply the SBN balancer to adaptive mesh applications using a heterogeneous network of workstations in which P is not necessarily a power of 2. Since low-cost processing power is readily available, it would be desirable to explore the effect of adding latency-tolerant techniques to the load balancing method. This research also includes techniques to adapt the processing to situations where some of processors in a network environment

Table 1: Mesh adaptation results using SBN balancer

Adaptation	Pre-Exec	Post-Exec		
Level	Cut %	Cut %	MaxSR	LoadImb
P=2				
1	0.09%	4.64%	6,974	1.00
2	3.14%	6.18%	30,538	1.00
3	5.36%	6.08%	57,724	1.00
4	3.93%	3.86%	20,646	1.00
5	2.91%	5.32%	76,893	1.00
6	2.33%	4.62%	103,544	1.00
7	2.23%	5.86%	140,904	1.00
8	2.83%	6.14%	153,735	1.00
9	3.10%	6.89%	129,374	1.00
Avg	3.14%	5.54%	80,037	1.00
P=4				
1	2.26%	8.15%	4,078	1.00
2	7.22%	10.01%	26,187	1.00
3	9.44%	11.69%	64,110	1.00
4	9.16%	9.48%	46,406	1.00
5	6.60%	11.86%	149,042	1.00
6	9.83%	10.89%	94,269	1.00
7	6.58%	8.00%	50,337	1.00
8	2.79%	15.31%	170,408	1.00
9	11.53%	11.48%	85,152	1.00
Avg	7.86%	11.17%	76,665	1.00
P=8				
1	6.66%	10.77%	2,518	1.01
2	13.93%	14.98%	11,109	1.00
3	15.11%	18.16%	46,088	1.00
4	14.65%	15.83%	53,032	1.00
5	11.09%	16.48%	69,583	1.00
6	11.02%	15.91%	85,982	1.00
7	13.75%	18.13%	105,946	1.00
8	12.84%	19.51%	28,974	1.00
9	15.34%	17.35%	80,477	1.00
Avg	13.30%	17.18%	53,745	1.00
P=16				
1	15.36%	20.61%	1,767	1.01
2	24.82%	25.56%	7,259	1.00
3	24.40%	27.45%	36,031	1.01
4	20.60%	22.77%	43,943	1.01
5	16.11%	24.27%	71,736	1.01
6	17.83%	22.28%	66,211	1.01
7	19.75%	25.00%	55,361	1.01
8	17.83%	25.30%	64,796	1.01
9	17.87%	21.59%	74,316	1.01
Avg	19.19%	24.02%	46,825	1.01
P=32				
1	21.59%	26.74%	1,184	1.01
2	30.35%	32.32%	4,387	1.02
3	30.06%	34.04%	8,445	1.02
4	27.28%	31.43%	41,783	1.01
5	21.35%	29.40%	42,843	1.01
6	24.04%	29.42%	42,688	1.01
7	22.35%	30.45%	41,347	1.02
8	20.59%	30.48%	37,006	1.02
9	22.19%	29.43%	32,594	1.02
Avg	23.97%	30.58%	28,031	1.02

Table 2: Mesh adaptation results using `PMeTiS` under the PLUM environment

Adaptation	Pre-Exec	Post-Exec		
Level	Cut %	Cut %	MaxSR	LoadImb
P=16				
1	3.16%	4.38%	10,088	1.02
2	5.34%	7.20%	25,875	1.02
3	7.27%	9.71%	58,887	1.03
4	5.24%	8.62%	134,808	1.03
5	5.77%	8.17%	153,154	1.04
6	4.70%	8.06%	122,151	1.02
7	4.47%	8.45%	159,037	1.02
8	5.31%	7.97%	132,987	1.01
9	4.18%	7.75%	130,824	1.01
Avg	5.05%	7.81%	103,090	1.02
P=32				
1	4.78%	6.45%	5,097	1.01
2	7.56%	10.05%	16,758	1.02
3	10.28%	13.13%	39,565	1.05
4	8.14%	11.60%	73,074	1.06
5	7.59%	11.13%	92,581	1.05
6	6.51%	11.60%	82,751	1.06
7	6.66%	11.43%	88,642	1.03
8	6.88%	11.39%	91,301	1.05
9	6.19%	11.66%	79,662	1.04
Avg	7.18%	10.94%	63,270	1.04

Table 3: Mesh adaptation results using `DMeTiS` under the PLUM environment

Adaptation	Pre-Exec	Post-Exec		
Level	Cut %	Cut %	MAXSR	LoadImb
P=32				
1	4.65%	15.70%	5,047	1.88
2	19.26%	20.50%	17,393	2.12
3	21.14%	25.26%	44,413	2.12
4	17.13%	28.21%	99,232	1.87
5	29.08%	26.46%	97,280	1.68
6	25.31%	24.38%	86,204	1.41
7	20.55%	14.17%	78,312	1.11
8	10.04%	13.08%	72,474	1.05
9	9.41%	14.18%	62,522	1.05
Avg	17.40%	20.30%	62,542	1.59

are not available. Fault tolerance would allow applications to make use of resources that are constantly changing during execution.

Acknowledgements

This work is supported by Texas Advanced Research Program Grant Number TARP-97-003594-013 and by NASA under Contract Number NAS 2-14303 with MRJ Technology Solutions.

References

[1] C. Alpert and A. Kahng, "Recent directions in netlist partitioning," *Integration, the VLSI Journal*, 19(1-2) (1995), pp. 1–81.

[2] R. Biswas and L. Oliker, "Experiments with repartitioning and load balancing adaptive

meshes," NASA Ames Research Center, Moffett Field (1997), Tech Rep NAS-97-021.

[3] N. Chrisochoides, "Multithreaded model for the dynamic load balancing of parallel adaptive PDE computations," *Applied Numerical Mathematics*, 20 (1996), pp. 321–336.

[4] G. Cybenko, "Dynamic load balancing for distributed-memory multiprocessors," *Journal of Parallel and Distributed Computing*, 7 (1989), pp. 279–301.

[5] S.K. Das and D.J. Harvey, "Performance analysis of an adaptive symmetric broadcast load balancing algorithm on the hypercube," Dept Computer Science, Univ North Texas, Denton (1995), Tech Rep CRPDC-95-1.

[6] S.K. Das, D.J. Harvey, and R. Biswas, "Adaptive load balancing algorithms using symmetric broadcast networks: Performance study on an IBM SP2", *Proc. 26th International Conference on Parallel Processing* (1997), pp. 360–367.

[7] S.K. Das and S.K. Prasad, "Implementing task ready queues in a multiprocessing environment," *Proc. International Conference on Parallel Computing* (1990), pp. 132–140.

[8] S.K. Das, S.K. Prasad, C-Q. Yang, and N.M. Leung, "Symmetric broadcast networks for implementing global task queues and load balancing in a multiprocessor environment," Dept Computer Science, Univ North Texas, Denton (1992), Tech Rep CRPDC-92-1.

[9] J. Garbers, H.J. Promel, and A. Steger, "Finding clusters in VLSI circuits," *Proc. IEEE International Conference on Computer Aided Design* (1990), pp. 520–523.

[10] L. Hagen and A. Kahng, "A new approach to effective circuit clustering," *Proc. IEEE International Conference on Computer Aided Design*, (1992), pp. 422-427.

[11] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," Sandia National Laboratories, Albuquerque (1993), Tech Rep SABD83-1391M.

[12] G. Horton, "A multi-level diffusion method for dynamic load balancing", *Parallel Computing*, 19 (1993), pp. 209–229.

[13] G. Karypis and V. Kumar, "Analysis of multilevel graph partitioning," Dept Computer Science, Univ Minnesota, Minneapolis (1995), Tech Rep 95-037.

[14] G. Karypis and V. Kumar, "Parallel multilevel K-way partitioning scheme for irregular graphs," Dept Computer Science, Univ Minnesota, Minneapolis (1996), Tech Rep 96-036.

[15] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs", *Bell Systems Tech. Journal*, 49 (1970), pp. 291–307.

[16] G.A. Kohring, "Dynamic load balancing for parallelized particle simulations on MIMD computers," *Parallel Computing*, 21(1995), pp. 683–693.

[17] L. Oliker and R. Biswas, "PLUM: Parallel load balancing for adaptive unstructured meshes," NASA Ames Research Center, Moffett Field (1997), Tech Rep NAS-97-020.

[18] R. Ponnusamy, N. Mansour, A. Choudhary, and G.C. Fox, "Graph contraction and physical optimization methods: a quality-cost tradeoff for mapping data on parallel computers," *Proc. 7th Int'l Conf on Supercomputing* (1993).

[19] S. Pulidas, D. Towsley, and J.A. Stankovic, "Embedding gradient estimators in load balancing algorithms," *Proc. Int'l Conf on Distributed Computer Systems* (1988), pp. 488–490.

[20] K. Schloegel, G. Karypis, and V. Kumar, "Multilevel diffusion schemes for repartitioning of adaptive meshes," Dept Computer Science, Univ Minnesota, Minneapolis (1997), Tech Rep 97-013.

[21] R. Van Driessche and D. Roose, "Load balancing computational fluid dynamics calculations on unstructured grids," *Parallel Computing in CFD*, AGARD-R-807 (1995), pp. 2.1–2.26.

[22] A. Vidwans, Y. Kallinderis, and V. Venkatakrishnan, "Parallel dynamic load balancing algorithm for three-dimensional adaptive unstructured grids," *AIAA Journal*, 32 (1994), pp. 495–505.

[23] C. Walshaw, M. Cross, and M.G. Everett, "Parallel dynamic graph-partitioning for unstructured meshes," School of Computing and Mathematical Sciences, Univ of Greenwich, London (1997), Tech Rep 97/1M/20.