Mapping Unstructured Computational Graphs for Adaptive and Nonuniform Computational Environments

Maher Kaddoura, Chao-Wei Ou, and Sanjay Ranka
School of Computer and Information Science
4-116 Center for Science and Technology
Syracuse University
Syracuse, NY 13244-4100
ranka@top.cis.syr.edu
315-443-4457

August 1994 (revised February 1995)

submitted to
IEEE Parallel and Distributed Technology

Abstract

In this paper we study the problem of mapping a large class of irregular and loosely synchronous data-parallel applications in a nonuniform and adaptive computational environment. The computational structure of these applications can be described in terms of a computational graph, where nodes of the graph represent computational tasks and edges describe the communication between tasks.

Parallelization of these applications on nonuniform computational environments requires partitioning the graph among the processors in such fashion that the computation load on each node is proportional to its computational power, while communication is minimized. We discuss the applicability of current methods for graph partitioning for such environments. For an adaptive computational environment, the partitioning of the graph needs to be updated as the environment adapts, hence most algorithms described in the literature are computationally prohibitive. We discuss novel strategies that allow for fast remapping.

Keywords: *Graph Partitioning, Nonuniform machines, Heterogeneous computing, Irregular Problems, Network of Workstations*

1 Introduction

Heterogeneity has become commonplace in high-performance computing environments. In the future most computing environments will consist of a cluster of nodes connected by a high-speed interconnection network. Node architectures will include high-performance SIMD and MIMD parallel computers as well as numerous high-performance workstations. By pooling together as many resources as possible, a heterogeneous environment represents the largest machine to which a researcher has access. This pool of resources may change over the lifetime of the computation due to machine failures or differing usage patterns. It should be possible to add or remove computational resources without significantly affecting the other machines and without changing the existing software. In such an environment an individual machine could be either dedicated to a single user's computation or shared among users. The former has the advantage of having static computing capability for each machine, while the latter has the advantage of a higher rate of utilization. The resources available to the user could be classified as:

1. Static: Computational resources are fixed throughout the completion of all tasks.
2. Dynamic: Computational resources vary dynamically throughout the computation because of sharing among users.
3. Adaptive: Computational resources remain fixed for a reasonable interval of time followed by a change.

In this paper we study the mapping requirements for the parallelization of a large class of irregular and loosely synchronous data-parallel applications on a nonuniform and potentially adaptive computational environment. The computational structure for these irregular and loosely synchronous problems can be described in terms of a computational graph, where nodes of the graph represent computational tasks and edges describe the communication between tasks. Parallelization of these applications requires partitioning the graph among the processors in such fashion that the computation load on each node is balanced, while communication is minimized. Optimal partitioning would allow optimal parallelization of the computations with the load balanced over various processors and with minimal communication time. Obtaining suboptimal solutions is possible and often satisfactory.

There are several algorithms available in the literature for partitioning and mapping this computational graph. Important heuristics include simulated annealing [12], neural networks [10], genetic algorithms [12], and spectral bisection [13]. For many applications, the computational graph is such that the vertices correspond to two- and three-dimensional coordinates, and the interaction between computations is limited to physically proximate vertices. For these applications, partitioning can be achieved by exploiting the above property. Essentially proximate points are clustered together to form a partition such that the numbers of vertices attached to every partition are equal. Many such algorithms are available in the literature, including coordinate bisection [17], inertial bisection [14], and index-based mapping schemes [2].

All of the above methods have been studied for mapping graphs onto uniform parallel machines. In this paper we evaluate the performance of these methods for partitioning computational graphs for a cluster of machines having a nonuniform and adaptive computational environment. The algorithms assume that all the processors are connected by an interconnection network in which the cost of unit communication is the same between all the processors (e.g., a bus). A good decomposition should minimize interprocessor communication while making sure that each processor is assigned work proportional to its computational power. While the algorithms are specifically targeted towards a cluster of workstations connected by a network, the issues are similar for parallelizing such applications on a network of machines.

For an adaptive environment, the partitioning of the graph needs to be updated as the computational resources change over time. This could be due to a change in the computational load of one or more machines and/or the addition or deletion of one or more machines. Ideally, a solution of the previous graph-partitioning problem should be utilized to partition the graph for the new environment, and the time required for such repartitioning should be much less than the time required to reapply a partitioning algorithm from scratch. If the graph is not repartitioned, it may lead to imbalance in the time required for computation on each node, which will cause considerable deterioration in the overall performance. We develop simple strategies for such an environment by mapping the vertices of a graph onto a one-dimensional space. This conversion allows us to provide an extremely fast remapping when the computational environment changes. The quality of partitioning achieved by this simple remapping is an acceptable deterioration over mapping from scratch.

Let $a_1, a_2, a_3 \ldots, a_p$ be the relative speed of nodes in a heterogeneous environment such that $a_1 + a_2 + a_3 \ldots a_p = 1$. The problem can then be stated as follows:

> Decompose a graph into p partitions such that the weight of each partition is in the ratio $a_1 : a_2 : a_3 : a_4 \ldots a_p$, and the total number of edges between the p partitions (cross edges) is minimized.

Another important way to evaluate the quality of partitioning is by the number of edges generated in the supergraph representing the connectivity of different partitions produced. Each node of this graph represents a partition. An edge in the supergraph is present whenever there are any cross edges from a node of one partition to a node of another partition. The number of edges in the supergraph represents the number of communication messages generated. For current message-passing software, a startup overhead (setup cost) is required for every message generated. Minimizing this overhead requires the minimization of the number of edges in the super graph. However, there exists message-passing software for broadcast networks such as Ethernet which support multicast [3]. In such cases the software overhead of sending multiple messages can be reduced by multicasting one message together with the combined data for all the destination processors.

The rest of the paper is organized as follows. In Section 2 we describe different graph-partitioning heuristics for a nonuniform but static environment and their performance on representative computational grids derived from real applications. Section 3 presents a new approach for mapping and remapping for an adaptive computational environment.

2 Partitioning for nonuniformity

In this section we present several methods for mapping graphs into nonuniform machines. These methods are of two distinct types:

1. The first type uses coordinate information to partition a graph.

2. The second type uses edge information to partition a graph.

Many of these methods perform mapping by recursively partitioning a graph into two parts. When partitioning a graph among a cluster of machines with nonuniform computational powers, the way in which the computational powers of the processors are grouped will affect the quality of the partitioning. We have evaluated two ways for grouping the weights of the processors, which we shall refer to as "Simple" and "Binpacking." The former partitions the graph at each level according to the first y weights available such that the sum of the weights is approximately half of the total. The latter divides the available weights into two groups such that the sum of the two groups is approximately equal and then partitions the graph according to the two groups. For example, consider the case of four weights with sizes 0.4, 0.4, 0.1, and 0.1. In this case Simple will partition the graph into two parts with one partition having 80% of the total size, while Binpacking will partition the graph into two equal parts.

The following partitioners were used for partitioning the graphs:

1. *Recursive coordinate bisection (RCB):* This method [17] uses only coordinate information and recursively bisects along the longest dimension by finding a hyperplane such that the the sizes of the two partitions are proportional to their weights (see Figures 5 (a) through (e)).

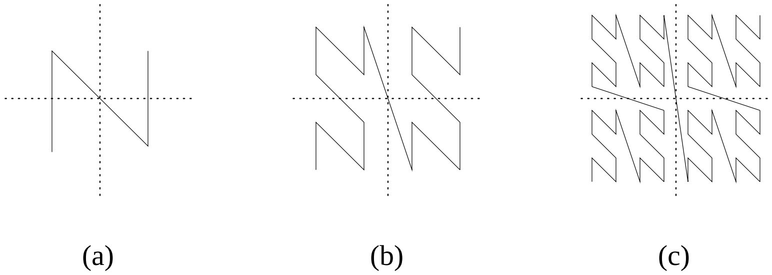

Figure 1: (a) Z-curve for (a) 4, (b) 16, and (c) 64 squares

2. *Index-Based Algorithms:* These methods map the vertices of the graph from multi-dimensional space into a one-dimensional space such that the proximate vertices have proximate indices. One way to achieve such a mapping is by bit-interleaving [2, 6]. Consider a vertex with coordinate (13,4) in two dimensions. The corresponding binary representation is (1101, 0100). The result of interleaving bits is 10110010. Thus (13,4) is mapped to 178 in a one-dimensional

space. This approach produces a space-filling curve which traverses the space recursively in a “Z” fashion (Figure 1). We shall refer to the above indexing algorithm as (ZCI). Another way to achieve indexing is based on Hilbert curves [11] (see Figure 2). Hilbert curves, unlike Z-curves, do not have large “jumps.” The algorithm for indexing using the Hilbert curve (HCI) is described in Figure 3. When mapping a computational graph using this approach, the co-ordinate information of the vertex is used to calculate its index (either based on the Z-curve or on the Hilbert curve). This list can then be sorted to rearrange the vertices [2], and can then be partitioned into appropriate sizes of contiguous sublists.

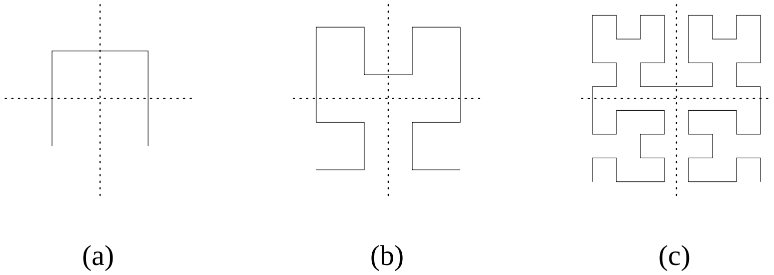

Figure 2: Hilbert curve for (a) 4, (b) 16, and (c) 64 squares

```
Initial conditions:
1. $Rotation\_Table[4] := \{3,0,0,1\}$.
2. $Sense\_Table[4] := \{-1,1,1,-1\}$.
3. $Quad\_Table[4][2][2] := \{ \{\{0,1\},\{3,2\}\},\{\{1,2\},\{0,3\}\},$
     $\{\{2,3\},\{1,0\}\},\{\{3,0\},\{2,1\}\} \}$;
procedure HCI($x, y$)
    $Rotation := 0$;    $Sense := 1$;    $Num := 0$;
    for ($k := side/2; k > 0; k := k/2$)
        $Xbit := x/k$;
        $Ybit := y/k$;
        $x := x - k \times Xbit$;
        $y := y - k \times Ybit$;
        $Quad := Quad\_Table[Rotation][Xbit][Ybit]$;
        if ($Sense == -1$)
            $Num := Num + (k \times k \times (3 - Quad))$;
        else
            $Num := Num + (k \times k \times Quad)$;
        $Rotation := Rotation + Rotation\_Table[quad]$;
        $Rotation := Rotation\ modulo\ 4$;
        $Sense := Sense \times Sense\_Table[quad]$;
end.
```

Figure 3: Hilbert curve based indexing algorithm for two dimensions

3. *Recursive spectral bisection (RSB):* These methods recursively partition the graph based on the second eigenvector calculation of the Laplacian matrix of the given graph [13]. These methods use edge information explicitly and have been empirically shown to perform very

well for uniform computational environments. Recently several extensions have been proposed to improve on the quality and time requirements [15][4].

2.1 Simulation results

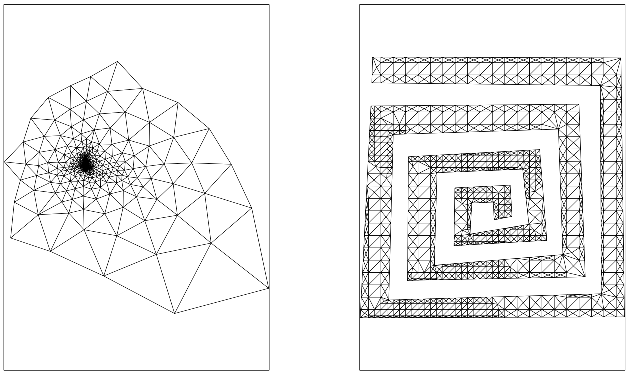

Figure 4: Graph 1 (4720 vertices, 13722 edges), Graph 2 (1200 vertices, 3191 edges)

In this section we present the performance of the algorithms described in the previous section. The quality of partitioning produced by the different methods was measured for the following parameters:

1. *Number of partitions (Par):* We performed our simulation for 4, 5, 10, 15 and 20 partitions. We decided upon a limit of 20 partitions, because we believe that the use of such environments for data-parallel computing will tend to be limited to this number.

2. *Ratio of Maximum Weight/Minimum Weight:* We varied this ratio from 1 to 8 in order to study the effects of nonuniformity between computational units. We limited ourselves to a ratio of 8 because our experimental results (to be presented) suggest that this ratio has limited impact on the relative performance of different partitioning stratagies.

3. *Strategies used for grouping processors:* We considered two different ways for grouping partition weights (Simple and Binpacking).

4. *Computational grids:* We performed experiments on several irregular graphs. We decided to present results for two representative graphs (Figure 4); the performance measures of other graphs have similar behaviour. Graph 1 has 4720 vertices and 13722 edges. Graph 2 has 1200 vertices and 3191 edges.

For each value of the above parameters, 20 samples were generated randomly with different computational powers. Based on our preliminary experiments, we concluded that ZCI and HCI had comparable performances. In most cases HCI performed slightly better than ZCI, hence the results are presented for HCI only. Table 1 shows a comparison between HCI and ZCI for Graph 1 and Graph 2 for the different number of partitions when all the processors have equal computational power. It shows average total edges crossing the partitions (Edges) and average number of edges produced in the supergraph (Setups).

Table 2 presents the performances of RSB, RCB, and HCI for the different computational ratios of 10 partitions. Each entry represents an average of the 20 randomly generated samples. These results are typical of other partitions size and show that the relative values of cross edges and setups do not have much variation or many patterns for the different computational ratios of a given partitioner.

	Graph 1				Graph 2			
Par	HCI		ZCI		HCI		ZCI	
	Edges	Setups	Edges	Setup	Edges	Setups	Edges	Setup
4	620	5	687	5	170	5	164	5
5	604	9	748	9	198	7	235	8
10	868	23	1090	25	307	18	355	22
15	1157	36	1479	40	427	31	471	33
20	1346	50	1566	61	487	40	486	50

Table 1: Performance of HCI and ZCI using Simple for grouping partitions (Ratio=1)

Table 3 presents the performances of RSB, RCB, and HCI for a different number of partitions. Each entry represents an average of the samples for different computational ratios (i.e., it represents the average over 100 samples). Table 4 shows the average time spent by the algorithms for Graph 1 and Graph 2. The following observations can be made:

1. RSB performs much better than the methods that use only coordinate information, but is computationally more expensive. This extends the corresponding empirical evidence in the literature [17] for uniformly sized partitions.

2. For methods that use only coordinate information, the performance of HCI is much better than RCB in most cases. The method based on HCI seems to be able to provide better clustering of points because the indexing is based on using all the dimensions simultaneously.

   The computational cost of HCI is higher because it requires sorting. However, this cost is independent of the number of partitions. The cost of RCB increases with the number of partitions but is cheaper, as at each stage it requires finding only the median.

3. The method of grouping processors does not play a major role in the quality of partitioning. This is not surprising, as none of the methods have a particular bias towards particular partition sizes.

Ratio	Graph 1											
	RSB				RCB				HCI			
	Binpacking		Simple		Binpacking		Simple		Binpacking		Simple	
	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups
1	514	19	514	19	1143	38	1143	38	868	23	868	23
2	560	18	564	18	1143	38	1143	38	994	22	1025	22
3	552	18	521	19	1143	38	1143	38	1003	22	1005	23
4	551	18	518	19	1143	38	1143	38	1001	22	947	23
8	543	18	534	19	1174	40	1143	38	977	22	964	22

Ratio	Graph 2											
	RSB				RCB				HCI			
	Binpacking		Simple		Binpacking		Simple		Binpacking		Simple	
	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups
1	86	9	87	9	429	28	429	28	307	18	307	18
2	93	9	88	9	429	28	429	28	317	18	321	18
3	89	9	84	9	429	28	429	28	316	18	322	18
4	90	9	87	9	429	28	429	28	315	19	325	18
8	92	9	88	9	445	30	429	28	312	19	321	18

Table 2: Performance of RSB, RCB, and HCI (Partitions = 10)

Par	Graph 1											
	RSB				RCB				HCI			
	Binpacking		Simple		Binpacking		Simple		Binpacking		Simple	
	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups
4	269	5	269	5	423	6	413	5	590	5	569	5
5	313	8	343	7	736	16	736	16	664	8	661	8
10	544	18	530	19	1149	38	1143	38	969	22	962	23
15	724	29	726	30	1662	73	1544	64	1227	39	1200	37
20	880	40	861	40	1960	105	1876	96	1396	52	1395	51
Par	Graph 2											
	RSB				RCB				HCI			
	Binpacking		Simple		Binpacking		Simple		Binpacking		Simple	
	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups
4	30	3	30	3	145	5	143	5	173	5	165	5
5	43	4	40	4	231	12	231	12	188	7	196	7
10	90	9	87	9	432	28	429	28	313	18	319	18
15	140	14	137	14	678	52	615	54	424	29	405	30
20	186	19	187	19	865	80	823	72	480	40	484	40

Table 3: Performance of RSB, RCB, and HCI for different numbers of partitions

Par	Graph 1			Graph 2		
	RSB	RCB	HCI	RSB	RCB	HCI
4	22.73	0.08	0.94	5.66	0.04	0.06
10	33.74	0.10	0.94	7.33	0.08	0.24
20	37.93	0.13	0.94	7.79	0.10	0.25

Table 4: Execution time of RSB, RCB, and HCI for Graph 1 and Graph 2 (in seconds)

The above conclusions are true independent of the number of partitions or ratio of maximum/minimum weight of the partitions.

3 Partitioning for adaptivity

For an adiabatic environment there is a need to remap the graph according to the changed computational power of the machines as available computational resources change. The graph could be remapped from scratch by using the best algorithm described above (RSB). However, the high computational cost may make it prohibitive if the computational graph adapts frequently. Index-based algorithms are an attractive partitioning method for adiabatic environments. They map the vertices of a graph into a one-dimensional space. After the initial mapping it is inexpensive to partition the one-dimensional list among the machines according to their computational powers, since partitioning is equivalent to assigning contiguous blocks of vertices to each partition. The size of each block is proportional to the weight of the partition. When the available computational resources change, the graph can be remapped by repartitioning the one-dimensional list.

RSB and RCB can be extended to map the vertices of a graph to one-dimensional space. We shall refer to them as RSBI and RCBI, respectively. Let the number of vertices in the graph be given by N. To obtain the appropriate indices for all the vertices, both algorithms bisect the graph into two equal subgraphs. For the first stage, the index set assigned to the first partition is from 1 to $N/2$, while for the second partition it is $N/2+1$ to N (assuming N is even). The number of recursive steps is equal to $\lfloor \log N \rfloor$. Figure 5 describes the application of recursive coordinate bisection for mapping to a one-dimensional space (from 1 to N).

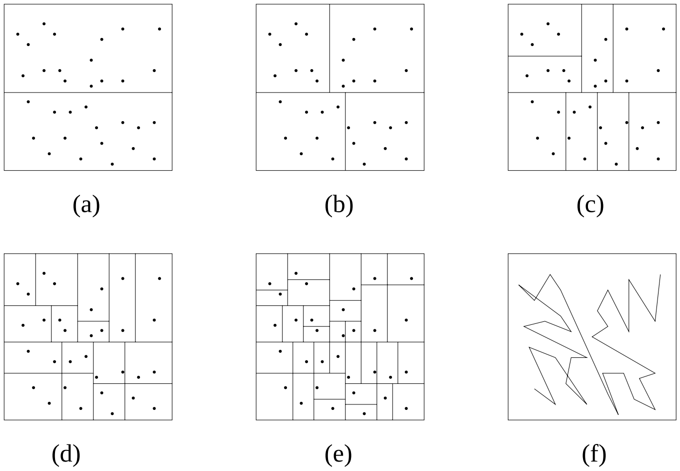

Figure 5: (a) through (e) describe the steps for 32 partitions using recursive coordinate bisection; (f) describes the curve corresponding to RCBI

3.1 Simulation results

Par	Graph 1				Graph 2			
	Binpacking		Simple		Binpacking		Simple	
	Edges	Setups	Edges	Setups	Edges	Setups	Edges	Setups
4	364	5	388	5	61	3	70	3
5	485	7	489	7	91	4	96	4
10	764	22	798	21	177	11	175	11
15	1017	37	1059	37	256	17	257	17
20	1211	51	1244	51	323	25	334	24

Table 5: Performance of RSBI

Par	Graph 1		Graph 2	
	Binpacking	Simple	Binpacking	Simple
4	26	29	45	49
5	36	30	56	55
10	27	34	51	48
15	28	31	46	47
20	27	31	42	44

Table 6: Percentage gain of RSB (from scratch) over RSBI

Graph	Time
Graph 1	53.58
Graph 2	9.68

Table 7: The execution time of RSBI for Graph 1 and Graph 2 (in seconds)

The quality of partitioning produced by the different algorithms was measured for the parameters described in Section 2.1. Our preliminary experiments showed that the quality of RCBI was much lower than that of HCI. This was expected, as even the quality of RCB for a given set of partitions was generally worse than the quality of HCI.

Tables 5 and 6 present the performance of RSBI. For achieving indexing using RSBI, the graph was partitioned into 1024 partitions. The indices within each of the 1024 partitions were assigned randomly.[1] These results show that the quality of partitioning produced by RSBI is better than that of HCI and RCB (from scratch). The quality of partitioning produced by using Binpacking is better than that of Simple grouping, because RSBI maps the vertices of the graph to a one-dimensional space by recursively partitioning the graph into two equal parts at each level. Table 7 shows the average time spent by RSBI for Graph 1 and Graph 2.

Table 6 presents the percentage gain of using RSB from scratch over RSBI. This improvement varies between 26%–56%. This is expected, because the former was specifically optimized for a given set of partitions. RSBI is a reasonable algorithm to use when the environment is adaptive and if the initial cost of mapping is not prohibitive. HCI produces slightly worse partitioning than RSBI, but at a relatively much smaller initial cost.

The above discussion shows that methods using coordinate or edge information can be employed to map a computational graph into a one-dimensional space such that good quality repartitionings can be achieved at a negligible cost. The former methods are useful when the cost of initial mapping is not important, while the latter methods are useful when the cost of initial mapping cannot be ignored.

4 Conclusions

Initial mapping	Computational Environments	
cost	Adaptive	Static
Critical	HCI	HCI
Not critical	RSBI	RSB

Table 8: Comparison of the algorithms based on environments and initial mapping cost

We have presented several algorithms for mapping computational graphs on adaptive and nonuniform computational environments. Table 8 gives the best algorithm, depending on whether the computational environment is adaptive and if the initial cost of partitioning is critical.

The major contribution of the paper is that it shows that index-based algorithms (based on coordinate information or on edge information) provide solutions of reasonable quality at a very low cost (at the time of execution). We believe this strategy would be extremely important for the

[1] This is due to the memory limitations of our current software. Also, further decompositions are not expected to provide any significant (if any) improvements.

parallelization of irregular applications on adaptive and nonuniform environments. Such a mapping has been shown to be extremely important for the parallelization of unstructured mesh using the STANCE (Software Techniques for Adaptive and Nonuniform Computational Environments) [9].

Acknowledgements

We would like to thank Horst Simon for providing the meshes. We would like to thank Elaine Weinman for editing this paper for errors in grammar, punctuation, and syntax. We would like to thank the anonymous referees for their suggestions which greatly improved the quality of original manuscript.

References

[1] A. Beguelin et al., "HeNCE: Graphical Development Tools for Network-Based Concurrent Computing," *Proceedings SHPCC-92*, pp. 129–136, Williamsburg,VA, May, 1992.

[2] C. Ou, S. Ranka, and G. Fox, "Fast Mapping and Remapping Algorithm for Irregular and Adaptive Problems," *Proceedings of the 1993 International Conference on Parallel and Distributed Systems*, pp. 279–283, Taipei, Taiwan, December 1993.

[3] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetwork and extended LANs," *ACM Transactions on Computer Systems*, Vol. 8, pp. 85–110, May 1990.

[4] S. T. Barnard and H. D. Simon, "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Proceedings of the 6th SIAM Conference. Parallel Processing for Scientific Computing*, SIAM, pp. 711–718, 1993.

[5] J. A. Orenstein and T. H. Merett, "A Class of Data Structures for Associative Searching," *Proceedings of Third SIGACT News SIGMOD Symposium on the Principles of Database System*, pp. 181–190, 1984.

[6] J. A. Orenstein, "Spatial Query Processing in an Object-Oriented Database System," *Proceedings of ACM SIGMOD International Conference on the Management of Data*, pp. 326–336, 1986.

[7] J. A. Orenstein, "Redundancy in Spatial Database," *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Portland, OR, May–June, 1989.

[8] A. R. Butz, "Alternative Algorithm for Hilbert's Space-Filling Curve," *IEEE Transaction on Computers*, Vol. C–2, pp. 424–426, April, 1971.

[9] M. Kaddoura, and S. Ranka, "Run-time Support for Parallelization of Data-Parallel Applications on Adaptive and Nonuniform Computational Environments," submitted to *Supercomputing '95*, December 1995.

[10] N. Mansour, "Physical Optimization Algorithms for Mapping Data to Distributed-Memory Multiprocessors Hydrodynamics," technical report, Syracuse University, NY, 1993.

[11] D. Hilbert, "Uber die steitige Abbildung einer linie auf ein Flachenstuck," *Math. Ann*, 38, 1891.

[12] H. Maini, K. Mehrotra, C. Mohan and S. Ranka, "Genetic Algorithms for Graph Partitioning and Incremental Graph Partitioning," *Proceedings of Supercomputing '94*, November 1994.

[13] A. Pothen, H. Simon, and K-P Liou, "Partitioning Sparse Matrices With Eigenvectors of Graphs," *SIAM Journal of Matrix Analysis and Application*, Vol. 11, No. 3, July 1990.

[14] H. D. Simon, "Partitioning of unstructured problems for parallel processing," *Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications*, Pergammon Press, 1991.

[15] B. Hendrickson and R Leland, "An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations," technical report SAND92-1460, Sandia National Laboratories, Albuquerque, NM 87185, 1992.

[16] V. S. Sunderam, "PVM: A framework for parallel distributed computing," *Concurrency: Practice and Experience*, Vol. 2, No. 4, pp. 315–339, December, 1990.

[17] R. D. Williams, "Performance of dynamic load-balancing algorithm for unstructured mesh calculations," *Concurrency: Practice and Experience*, Vol. 3, No. 5, pp. 457–481, October 1991.