A Data-Parallel Adaptive N-body Method

A Data–Parallel Adaptive N–body Method

Yu Charlie Hu
S. Lennart Johnsson
Shang-Hua Teng

TR-14-96

December 1996

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

To appear in the *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, March 1997.

A Data–Parallel Adaptive N–body Method

Yu Charlie Hu* S. Lennart Johnsson† Shang-Hua Teng‡

Abstract

We present a data–parallel formulation of an adaptive version of Anderson's method for N–body particle interactions. Our formulation consists of a storage and computationally efficient array–based representation for the nonuniform hierarchy that models arbitrary particle distributions for the computational procedure. We also present data–parallel implementations (in HPF) of several well known partitioning methods. These partitioning methods balance nodal weights (for computation). We present preliminary experimental results for these partitioning schemes and discuss the complete code for adaptive particle simulations.

1 Introduction

Hierarchical methods for N–body simulations enabled the simulation of particle systems with up to 100 million particles on Massively Parallel Processors (MPP) installed a few years ago, and should allow for the simulation of 1 – 10 billion particle systems in main memory on the MPP systems currently under installation. Large–scale N–body simulations have applications in areas such as celestial mechanics, plasma physics, materials science and molecular design. Hierarchical $O(N)$ N–body methods have been proposed by Greengard and Rokhlin [7], Zhao [24], Anderson [1], Appel (proven to be of $O(N)$ in [6]), and Warren and Salmon [23], and $O(N \log N)$ methods have been presented by Barnes and Hut [2] and Callahan and Kosaraju [4].

The methods of Appel, and Barnes and Hut, and Callahan and Kosaraju are readily extended to nonuniformly distributed particles. Carrier, Greengard, and Rokhlin [5] presented an adaptive version of the Greengard–Rokhlin method. Similar extensions can be made to Anderson's (see Section 2) and Zhao's methods. As particles move close to each other nonuniformly, the arithmetic complexity of these adaptive methods can become superlinear – even exceed $O(N \log N)$ for some distributions. In practice, the depth of the hierarchy used in these N–body methods is limited by the machine precision as pointed out in [5], and the above adaptive method retains $O(N)$ arithmetic complexity, i.e. with a large constant $-log_2\epsilon$ in the big–O. Callahan and Kosaraju's formulation requires $O(N \log N)$ operations independent of the particle distribution.

Hierarchical N–body methods pose great challenges to parallel implementations. First, the complex computational structures and mathematics involved demand significant programming efforts; second, the highly irregular and extensive communication patterns make partitioning for locality and load–balancing on parallel platforms harder than partitioning unstructured meshes.

*Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138. Supported by the Air Force Office of Scientific Research through grants F49620-93-1-0480 and F49620-96-1-0289.

†Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138 and University of Houston, Houston, TX 77204. Supported by the Air Force Office of Scientific Research through grants F49620-93-1-0480 and F49620-96-1-0289.

‡Department of Computer Science, University of Minnesota, Minneapolis, MN 55455. Supported by an NSF CAREER award (CCR-9502540), an Alfred P. Sloan Research Fellowship, and an Intel research grant.

Previous work by Salmon and Warren [17, 22], and Liu and Bhatt [13] have shown that the Barnes–Hut method can be implemented efficiently on parallel scalable architectures, using the message–passing programming model. Nonadaptive versions of the $O(N)$ methods have also been implemented on parallel machines in both the message–passing and the shared–memory programming models [3]. In contrast, due to the more complicated computational structure, little progress has been made on data–parallel implementations of adaptive versions of the $O(N)$ methods.

In our previous work [9, 10], we have shown that efficient scalable code can be produced in data–parallel languages for nonadaptive versions of Anderson's method. In this paper, we present a data–parallel formulation of the adaptive version of Anderson's method. We have developed an implementation in High Performance Fortran (HPF) [8]. The key contributions of this paper are:

- A data–parallel formulation of the adaptive Anderson's method and an implementation in HPF.
- Data–parallel implementations of an extensive set of partitioning algorithms for preservation of locality and load–balancing, and therefore readily callable from data–parallel adaptive N–body codes.

2 Adaptive $O(N)$ Hierarchical N–body Algorithms

There are two key ideas in $O(N)$ hierarchical methods which together achieve the reduction of the arithmetic complexity: (1) Approximate the force or potential due to a cluster of particles with a single computational element; (2) Hierarchically form and use the computational elements.

All $O(N)$ methods [7, 1, 24] share the same computational structure and differ only in the approximate computational elements they use. There are two kinds of computational elements used in $O(N)$ methods: *far–field potential representation* and *local–field potential representation*. Both are used to represent the potential due to a cluster of charged particles at evaluation points far away from the cluster. The elements differ in whether the mathematical representation is with respect to the center of the domain containing the cluster of particles whose action is approximated (for far–field potential) or whether it is with respect to the geometric center of the domain of the set of particles upon which the action shall be evaluated (for local–field potential). The geometric centers of domains are used in both Anderson's method and the Rokhlin–Greengard method.

In Anderson's method [1], Poisson's formula is used for representing solutions of the underlying Laplace equation. Given a numerical integration formula, the discretized integration becomes a summation of expansions with Legendre functions as the base functions and only relies on the potential values at the integration points. The far–field and local field potentials are called *outer–sphere approximation* and *inner–sphere approximation*, respectively. The three translation operations used in $O(N)$ methods, namely, shifting far–field potential, converting far–field to local–field potential, and shifting local–field potential become simple evaluations of the outer–sphere and inner–sphere approximations of the source boxes at the destination boxes of the translation.

The $O(N)$ methods form and evaluate the computational elements hierarchically. They recursively subdivide the physical domain into a hierarchy of subdomains, hereafter called *boxes*. Every subdivision results in four or eight equal boxes for two– and three–dimensions, respectively. A box is subdivided if it contains more than a predetermined number of particles. For nonuniform distributions, this rule results in a potentially unbalanced hierarchy, as shown in Figure 1 for two dimensions..

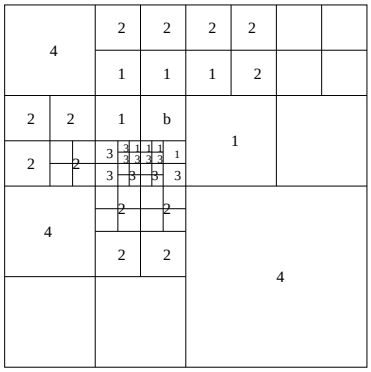

FIG. 1. *Adaptive domain decomposition, with one–separation near–field.*

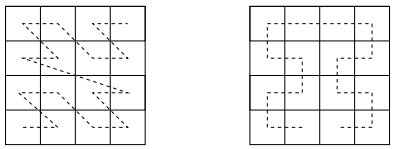

FIG. 2. *Morton and Peano–Hilbert ordering.*

TABLE 1
Four interaction lists in adaptive $O(N)$ methods.

List	Valid for	Definition	Interactions
List–1	leaf boxes	non–well–separated boxes	direct evaluation
List–2	all boxes	interactive–field boxes as in nonadaptive method	far–field to local–field
List–3	leaf boxes	well-separated offsprings of same–level neighbor boxes	far–field to particle
List–4	all boxes	inverse of List–3	particle to local–field

The adaptive methods associate with each box in the hierarchy four lists of boxes, referred to as List 1, 2, 3, and 4, which have different relative locations and will perform different interactions with the box "b" under consideration, as defined in Table 1. With *one–separation*, where only adjacent boxes are considered as near–field, the four lists of boxes relative to box "b" are shown for two dimensions in Figure 1. The computational structure of a generic adaptive $O(N)$ method is summarized in Table 2. Details can be found in [11].

3 A Data–Parallel Formulation

3.1 An Array–Based Hierarchy Representation

Our HPF formulation of the adaptive methods embeds the hierarchy in a single array data structure. Given the input particle coordinates and the maximum number of particles per leaf–level box, the following data–parallel method builds the adaptive hierarchy and store the boxes of the hierarchy in a one–dimensional (1–D) array in level–by–level order, with a Morton ordering (see Figure 2) at each level.

Algorithm *data–parallel hierarchy building*
Input particle coordinates with one 1–D array for each coordinate and the maximum number of particles per leaf–level box s.

1. Sort the particles so that for any threshold s, the particles belonging to the same leaf–level box are stored contiguously. Morton ordering or Peano–Hilbert ordering (see Figure 2), with a degree of refinement sufficient to separate all particles (into different boxes) can be used to achieve this property. In practice, the degree is bounded by $-\log_2 \epsilon$, where ϵ is the machine precision.
2. Construct the hierarchy via scanning on the sorted arrays, counting the number of particles per box at the current level and using a mask array to record whether further subdivision of a box (a segment of the particle arrays) is necessary. For every box (corresponding to an array element in the array representation), the construction builds and stores the parent

TABLE 2
Computation structure of adaptive $O(N)$ methods.

Stage	Computation	Active boxes
1. Build adaptive hierarchy		
2. Construct interaction list–1,2,3		
3. Form leaf–level far–field potential	particle to far–field	childless
4. Upward traversal	far–field to far–field	level–by–level
5. List–1 interactions	particle to particle	childless
6. List–2 interactions	far–field to local–field	all
7. List–3 interactions	far–field to particle	childless
8. List–4 interactions	particle to local–field	childless
9. Downward traversal	local–field to local–field	level–by–level
10. Evaluate leaf–level local–field potential	local–field to particle	childless

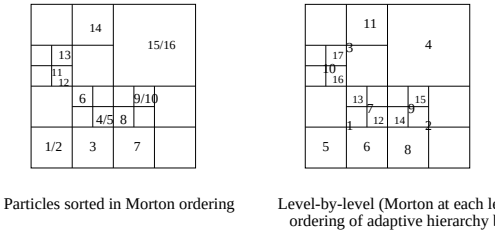

FIG. 3. *An example of array representation of an adaptive hierarchy. Threshold is 2 particles/leaf–level box.*

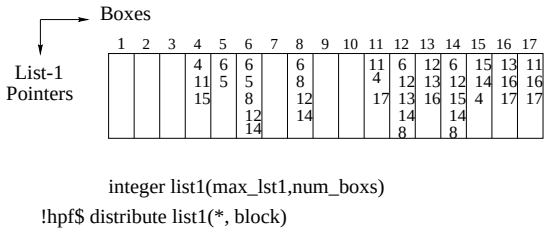

FIG. 4. *Array representations for pointers for List–1 interactive boxes.*

and first child pointers, a pointer in the particle arrays to the first particle in each box, and the number of particles in that box. Also stored are the coordinates of the box centers and a logical array for recording whether the box is a leaf box. Figure 3 shows an example of nonuniformly distributed particles sorted in Morton ordering and the linear ordering of boxes in the corresponding adaptive hierarchy.

Construction of the three interaction lists in Step 2 is performed via several downward traversals of the hierarchy each scanning the entire array storing the hierarchy. The lists constructed are stored in 2–D pointer arrays. A parallel axis is use to represent all the boxes in the hierarchy and a local axis is used to store pointers for the same box. Figure 4 shows the representation of List–1 for the example hierarchy in Figure 3. Since the number of each kind of interactive boxes in general is different for different "b" boxes, the memory usage of the above 2–D array representation may appear inefficient. However, for nonuniform distributions having reasonable continuity in the density distribution, the average and the largest numbers of interactive boxes is expected to differ by a small constant. In such cases, the inefficiency in our approach is considered acceptable. For example, for one million particles with the 3–D Plummer distribution, the average and largest number of types boxes per box "b" are 25.5 and 47 for List–1, and 149 and 189 for List–2.

3.2 List Interactions

We use List–1 interactions as an example to illustrate the data–parallel formulation of list interactions in HPF. List–2, –3, and List–4 interactions are implemented similarly. Details can be found in [11].

First, columns of the List–1 pointer array corresponding to leaf–level boxes are extracted into a compressed List–1 array upon which all the List–1 interactions will be performed.

List–1 interactions involve direct evaluation between particles in each leaf–level box and particles in its List–1 boxes. Using the owner compute rule, each box needs to first fetch the particles from its Lists–1 boxes, then perform the direct evaluation. Therefore,

associated with each box is a weight for the amount of communication in fetching nonlocal List–1 boxes and a weight for the amount of computation. The local copying of local List–1 boxes necessary in the data–parallel formulation is insignificant compared with the nonlocal fetch on most parallel platforms. The partitioning algorithms described in the next section try to balance the aggregate computation weights of all the boxes on each processor while minimizing the aggregate nonlocal data fetching.

The data fetching (gather communication) can be performed by scanning one slice of the List–1 pointer arrays at a time. The scanning is expressed in HPF using array indirect addressing. This approach is simple, but both the communication and the subsequent computation, if performed after every fetch, may be unbalanced since the partitioning algorithms balance the communication and computation for *all* List–1 interactions. A more efficient approach is to prefetch and store all particles in the interactive boxes, then perform the direct evaluation. The drawback with this approach is that the memory usage required to store the prefetched particles will be high. Our implementation takes an input parameter specifying the total memory per node that can be used for the prefetch. The code automatically decides how many batches it will divide the total fetches into and interleave the direct evaluation with the batches.

The particles belonging to the List–1 interactive boxes fetched in each batch are stored and compressed into a 2–D array with a parallel axis for the boxes and a local axis for the particles in each box. Since each fetch may receive different numbers of particles for different boxes, the resulting 2–D array may be ragged. The direct evaluation now becomes all–to–all interactions between the corresponding columns of the 2–D particle array (ragged) and the prefetched List–1 interactive particle array. This all–to–all interaction is expressed using an independent do loop for the parallel axis for the boxes and two sequential do loops for looping through the two corresponding columns.

3.3 Leaf–Level Particle–Box Interactions and Hierarchy Traversal

Leaf–level particle–box interactions are expressed as interactions between the leaf–level box array and the 2–D particle array, and no communication is required if the two arrays are aligned in the parallel memory.

Hierarchy traversal are performed level–by–level (LBL). At every level, gather and scatter operations are used to extract/embed the small working arrays corresponding to the current level boxes and that of the next level, i.e. between parent boxes and children boxes. The efficiency of the communication relies on a partitioning scheme that preserves good parent–child locality. As show in the next Section, different partitioning schemes often perform well for some steps and not so well for other steps of of the adaptive method. Therefore, there is no obvious choice of method.

4 Partitioning for Locality and Load–Balancing

The communication and computation cost associated with each box can be easily calculated according to the box counts of the interaction lists and the number of particles in each leaf–level box. Our partitioning scheme assumes alignment of the 2–D particle arrays with the leaf–level box array. The cost of particle–box interactions at the leaf–level is counted as part of the leaf–level boxes; all the costs are associated with boxes.

4.1 Mathematical Bases for N-body Partitioning

The interaction lists of the boxes in the hierarchical methods define the *computation graph* for an N–body problem, the *N–body graph*. N-body graphs have a higher node degree than typical finite element meshes, especially when the distribution of particles is nonuniform.

TABLE 3
A data–parallel library of partitioning schemes for adaptive $O(N)$ N–body methods.

Method	Input	load balancing quality	
		nodal weights	edge weights
ORB	workload + coord.	good	unknown
Morton	workload + coord.	good	unknown
Peano–Hilbert	workload + coord.	good	unknown
Level–by–level	workload + coord.	good	unknown
RRB	workload + coord.+adj.	good	unknown
GEO	workload + coord.+adj.	provably good	provably good
RSB	workload + coord.+adj.	provably good	provably good

The following result of Teng [20] gives an upper bound on the amount of interaction between partitions as a function of the height of the N–body graph.

THEOREM 4.1. *Let G be an N–body graph of a set of particles located at $P = \{p_1, ..., p_n\}$ in R^d ($d = 2$ or 3). If the height of the hierarchical tree for P is h, then G can be partitioned into two equal computational weighted subgraphs by removing at most $O(h^{1/d} n^{1-1/d})$ boxes.*

4.2 Load Balancing and Partitioning Heuristics

Previously, orthogonal recursive bisection (ORB) and Morton and Peano–Hilbert ordering have been used to partition particles in the Barnes–Hut method [17, 22, 18] and to partition boxes in an adaptive fast multipole method [18].

We have developed an extensive library of partitioning schemes together with their data–parallel implementations in HPF, as summarized in Table 3. We also developed an extension of ORB called rotational recursive bisection (RRB). Instead of alternating the partitioning in the coordinate directions (x, y, z) as in ORB, RRB tries independently for each subpartition, a sequence of line or hyperplane partitionings with random angles, examines the quality of the edge cuts, and chooses the best partition. With exception of RRB, the above mentioned partitioning heuristics have no guarantee on the quality of partitions, but they are very cheap to compute; they do not make use of the edge connectivity. Recent work of Cao, Gilbert, and Teng shows that RRB provides a slightly weaker guarantee on the quality of the partitions than that given by Theorem 4.1.

Geometric partitioning (GEO) [14] and a variant (given by Spielman and Teng [19]) of recursive spectral bisection (RSB) [16] both offer guarantees on the quality of the partitions. GEO lends itself to efficient data–parallel implementations, as shown in [12]. Extensive use of *sampling* can be used to reduce the computational complexity without a significant degradation in the quality of the partitions. Using this sampling technique GEO is computationally less demanding than a straight–forward implementation of RSB.

A fundamental issue with partitioning unbalanced hierarchies is that the uneven weights associated with the boxes renders partitions with an uneven number of array elements. Ragged arrays are supported in the recently announced HPF–II [8], but no commercial compiler available today supports it. We are currently investigating ways of achieving uneven distributions of arrays in HPF-I.

5 Experimental Results

The experimental results presented in the following are for one million particles with the Plummer distribution and with a threshold of 64 particles per leaf–level box.

The partitioning algorithms generate partitions of arrays with an unequal numbers of elements. Since no commercial HPF compiler today supports uneven array distributions, we are unable to measure the impact on the running time of HPF programs using different partitioning schemes. However, using mask arrays and segmented scan operations, we can

TABLE 4

Comparison of various partitioning algorithms for the 2–D Plummer model with one million particles. 128 partitions are generated. Arrays are initially in the LBL ordering.

Method	Remote references (Bytes/partition)		FLOPS / partition		Running time (sec.)	
	avg.	max.	avg.	max.	32-node CM-5E	16-wide-node SP2
List-1 (44144 nodes, 1118614 edges)						
LBL	8.05e+04	3.07e+05	5.42e+07	5.46e+07	N/A	N/A
Morton	6.32e+04	1.54e+05	5.42e+07	1.23e+08	0.06	0.03
Peano	6.00e+04	1.28e+05	5.42e+07	5.46e+07	0.13	0.08
ORB	6.20e+04	1.95e+05	5.42e+07	5.47e+07	0.57	3.84
RRB (10 trials)	4.86e+04	8.99e+04	5.42e+07	5.46e+07	14.7	98.3
RSB (no weights)	5.01e+04	8.70e+04	5.42e+07	9.61e+07	104.	—
GEO (10 trials)	4.91e+04	9.78e+04	5.42e+07	5.48e+07	19.6	202.
List-2 (33103 nodes, 295181 edges)						
LBL	1.02e+05	1.62e+05	9.06e+07	9.09e+07	N/A	N/A
Morton	1.62e+05	2.52e+05	9.06e+07	9.08e+07	0.05	0.02
Peano	1.53e+05	2.78e+05	9.06e+07	9.09e+07	0.10	0.06
ORB	1.70e+05	3.74e+05	9.06e+07	9.10e+07	0.44	3.66
RRB (10 trials)	1.48e+05	2.50e+05	9.06e+07	9.11e+07	8.88	64.7
RSB (no weights)	1.80e+05	2.79e+05	9.06e+07	9.66e+07	31.1	—
GEO (10 trials)	1.48e+05	2.60e+05	9.06e+07	9.10e+07	13.0	169.

simulate the distribution of the uneven partitions of the arrays and collect statistics such as the average and maximal number of remote references and floating–point operations among the partitions. Table 4 shows the above measurements for partitioning arrays representing active boxes in List–1 and List–2 interactions. The RSB result is derived from calling the RSB in CMSSL [21], which does not perform weighted partitioning.

Table 5 shows the preliminary performance results of our HPF implementation of our adaptive version of Anderson's method on a 16 wide–node IBM SP2. The communication currently accounts for 60% of the total running time for the 3–D Plummer distribution with one million particles, largely due to the poor performance of the unoptimized gather/scatter run–time system subroutines generated by the pghpf compiler version 2.1 [15]. It is expected that an order of magnitude improvement be achieved from highly optimized gather/scatter subroutines. This improvement will improve the overall performance by almost a factor of two. Further improvement can be achieved from improved cache performance by optimizing loop ordering, and from improved load–balance from uneven distribution of arrays.

6 Conclusion

We have presented a data–parallel formulation of an adaptive version of Anderson's N–body method in HPF. To our knowledge, this is the first data–parallel implementation of adaptive N–body methods. Preliminary performance results show that the data-parallel approach is promising for adaptive N-body simulations. Uneven distribution of arrays is crucial to achieving communication and computation load–balance for adaptive N–body simulations, and we are investigating ways of achieving this functionality while waiting for an HPF compiler to support this HPF extension.

Acknowledgment

We would like to thank Doug Miles and Paul Kinney of the Portland Group Inc., for their help with the performance tuning of the adaptive code using the pghpf compiler.

References

[1] C. R. Anderson. An implementation of the fast multipole method without multipoles. *SIAM J. Sci. Stat. Comput.*, 13(4):923–947, July 1992.

TABLE 5

Running time for one million particles with 3–D Plummer distribution on an 16 wide-node IBM SP2, with threshold 160 and Morton ordering but even array distribution. Expected RMS potential error against the direct method is 1.24E-05, verified for up to 10,000 particles.

Stage	Running time (sec.)	% of total FLOPS	Comm. portion (sec.)	Efficiency
1. Build hierarchy	14.1	0.00%	100.0%	0.00
2. Construct interaction lists	62.1	0.00%	100.0%	0.00
3. Form leaf–level far–field	5.35	1.74%	27.2%	17.2
4. Upward traversal	3.86	0.12%	87.5%	1.59
5. List–1 interactions	188.	38.9%	51.6%	11.0
6. List–2 interactions	166.	17.2%	88.9%	5.49
7. List–3 interactions	158.	21.9%	30.8%	7.31
8. List–4 interactions	39.5	17.9%	29.2%	24.0
9. Downward traversal	3.81	0.12%	88.5%	1.81
10. Evaluate leaf–level local–field	14.7	2.12%	21.6%	7.64
11. Partitioning and reordering	1.73	0.00%	100%	0.0
Total	658.	100%	59.8%	8.03

[2] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force calculation algorithm. *Nature*, 324:446–449, 1986.

[3] J. A. Board Jr., Z. S. Hakura, W. D. Elliott, D. C. Gray, W. J. Blanke, and J. Leathrum Jr. Scalable implementations of multipole–accelerated algorithms for molecular dynamics. In *SHPCC94*. IEEE CSP, May 1994.

[4] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *JACM*, 42:67–90, 1995.

[5] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM J. of Sci. and Stat. Comput.*, July 1988.

[6] K. Esselink. The order of Appel's algorithm. *Information Processing Letter*, 41:141–147, 1992.

[7] L. F. Greengard. *The rapid evaluation of potential fields in particle systems*. MIT Press, 1988.

[8] HPF Forum. *High Performance Fortran Language Specification, Version 2.0.δ*, October 1996.

[9] Yu Hu and S. Lennart Johnsson. A data parallel implementation of hierarchical N–body methods. *Intl. J. of Supercomput. Appl. and High Perf. Comput.*, 10(1):3 – 40, 1996.

[10] ———. Implementing $O(N)$ N–body algorithms efficiently in data–parallel languages. *Journal of Scientific Programming*, 5(4):337 – 364, 1996.

[11] ———, and S.-H. Teng. A data parallel implementation of adaptive $O(N)$ hierarchical N–body methods. TR–13–96, Harvard University, Division of Applied Sciences, September 1996.

[12] Y. C. Hu, S.-H. Teng, and S. L. Johnsson. A data–parallel implementation of the geometric partitioning algorithm. In *Proc. of the 8th SIAM Conf. on Para. Proc. for Sci. Comput.*, 1997.

[13] P. Liu. *The parallel implementation of N–body algorithms*. PhD thesis, Yale University, 1994.

[14] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Finite element meshes and geometric separators. *SIAM J. Sci. Comput.*, to appear, 1997.

[15] The Portland Group, Inc. *pghpf Reference Manual, Version 2.1*, 1996.

[16] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[17] J. K. Salmon. *Parallel Hierarchical N–Body Methods*. PhD thesis, Caltech, 1990.

[18] J. Singh, C. Holt, J. Hennessey, and A. Gupta. A parallel adaptive fast multipole method. In *Supercomputing'93*, pages 54 – 65, 1993.

[19] D. A. Spielman and S.-H. Teng. Spectral partitioning works: planar graphs and finite element meshes. In *37th Annual Symposium on Foundation of Computer Science*, pages 96–107, 1996.

[20] S.-H. Teng. Provably good partitioning and load balancing algorithms for parallel adaptive n-body simulation. *SIAM J. Scientific Computing*, to appear, 1997.

[21] Thinking Machines Corp. *CMSSL for CM Fortran, Version 3.1*, 1993.

[22] M. Warren and J. Salmon. A parallel hashed oct–tree N–body algorithm. In *Supercomputing'93*, pages 12 – 21, 1993.

[23] M. S. Warren and J. K. Salmon. A portable parallel particle program. *Computer Physics Communications*, 87, 1995.

[24] F. Zhao. An $O(N)$ algorithm for 3–dimensional N–body simulations. Technical Report AI–TR–995, AI Lab, MIT, 1987.