Parallelization of Irregular Problems Based on Hierarchical Domain Representation

Fabrizio Baiardi, Sarah Chiti, Paolo Mori, and Laura Ricci

Dipartimento di Informatica, Universitá di Pisa
Corso Italia 40, 56125 - PISA
{baiardi, chiti, mori, ricci}@di.unipi.it

Abstract. Irregular problems require the computation of some properties for a set of elements that are irregularly distributed in a domain. The distribution may change at run time in a way that cannot be foreseen in advance. Most irregular problems satisfy a locality property because the properties of an element e depend on the elements that are "close" to e. We propose a methodology to develop a highly parallel solution based upon a load balancing strategy that respects locality because e and most of the elements close to e are mapped onto the same processing node. We also discuss the update of the mapping at run time to recover an unbalancing, together with strategies to acquire data on elements mapped onto other processing node. The proposed methodology is applied to the multigrid adaptive problem and some experimental results are discussed.

1 Introduction

Several problems in computer science require the computation of some properties, i.e. speed, position, temperature, illumination etc., for each element in a domain of interest. The computation is iterated either to simulate the system evolution in an interval of time or to improve the accuracy of the results. A problem is irregular if the elements are distributed in the domain in a non homogeneous and dynamic way that cannot be foreseen in advance.

Some important examples of irregular problems are: the Barnes-Hut method for n-body problems [2], the adaptive multigrid method for the solution of partial differential equations [5] and the hierarchical radiosity methods to determine the global illumination of a scene [8].

In all these problems, the properties of an element e_i depend upon those of some other elements, the neighbors of e_i. A problem dependent neighborhood relation determines which elements affect the properties of e_i, but the probability that e_j affects the properties of the e_i is inversely related to the distance between e_i and e_j. In the following, this property will be denoted as *locality*.

A key point in the development of a highly parallel solution of an irregular problem is a load balancing strategy that maps the elements onto processing nodes (p-nodes) while respecting locality, i.e. that maps most of the neighbors of e_i onto the same p-node of e_i . Furthermore, the strategy should also define how the mapping is updated when and if the distribution changes.

Several techniques have been developed to parallelize irregular problems on parallel architectures with distributed memory. Two different parallelization strategy called *Costzone* and *Orthogonal Recursive Bisection* have been described in [15], [16] and [13]. These techniques are based on two different kind of hierarchical decomposition of the domain. The ordering of domain elements throught *space-filling curves* has been adopted in [7], [10] and [18]. Another parallelization approach for irregular problems, called CHAOS, is described in [14] and [9]. This approach requires that the problem consists of a sequence of clearly demarcated concurrent loop-nests.

This paper proposes a parallelization methodology for irregular problems that integrates two strategies: a load balancing strategy that respects locality and one to collect remote data, i.e. data of elements mapped onto a distinct p-node. The methodology is independent of the distributed memory parallel architecture, and it only assumes that the p-nodes are connected by a sparse interconnection network. The rest of the paper is organized as follows: sect. 2 describes a hierarchical representation of the domain, sect. 3 discusses the data mapping and the load balancing technique, sect. 4 presents the strategy collect remote data. The application of our methodology to the adaptive multigrid method and some experimental results are presented in Sect. 5. The application to the Barnes-Hut method has already been described in [1]

2 A Hierarchical Representation of the Domain

In the following, we assume that the problem domain belongs to a space with n dimensions. The proposed methodology exploits a hierarchical representation of the problem domain. At each level the domain is partitioned into a set of n-dimensional *spaces*. The procedure that partitions a space S is recursive and it starts from the space that represents the whole domain. If a problem dependent condition is satisfied, S is partitioned by halving each side to produce 2^n equal subspaces, and the procedure is applied to each resulting space. The spaces including a large number of elements are partitioned into finer spaces than the other ones. The whole decomposition is represented through the *Hierarchical Tree*, H-Tree. Each node of the H-Tree, hnode, represents a space and the root represents the whole problem domain. Each space S considered in the decomposition is represented by one hnode and this hnode records information on the elements in S. Larger spaces are paired with abstract information, while smaller ones are paired with a more detailed information. In the following, *space(N)* denotes the space represented by the hnode N, while *node(S)* denotes the hnode representing the space S. We notice that each hnode either is a leaf or has 2^n sons. If N is a leaf, *space(N)* is not decomposed.

Because of the non uniform distribution of the elements, two distinct subtrees rooted in the same hnode may have different heights. If the number of elements and/or their distribution change during the computation, the partition of the domain and the H-Tree are to be updated according to the current distribution. As soon as *space(N)* is partitioned, 2^n sons of N are inserted, while as soon as *space(N)* is no longer partitioned, the sons of N are pruned.

Our methodology assumes that the H-Tree cannot be replicated in each p-node, because of its memory requirement. Instead, we consider *np + 1* subset of the H-Tree, where *np* is the number of the p-nodes. One subset is the replicated H-Tree, replicated in all the p-nodes. Each of the other subsets is stored in one p-node only and it is the private H-Tree of the p-node.

3 Data Mapping and Runtime Load Balancing

To take locality into account, we propose a three step mapping: *i)* spaces ordering; *ii)* determination of the computational load of every hnode and *iii)* order preserving mapping of the spaces onto the p-nodes. The spaces are ordered through a space filling curve built on the spaces hierarchy representing the domain decomposition. Space filling curves are a family of curves that visit any point of a given space [11]. The curve is built starting from the lowest level spaces, i.e. from the first partition of the problem domain. These spaces are visited in the order stated by the characteristic figure of the adopted curve. If a space has been partitioned, then all its subspaces are visited in a recursive way, before the next space at the same level. Any space filling curve *sf* also defines a visit *v(sf)* of the H-Tree that returns a sequence $S(v(f)) = [N_0, N_1, ..., N_m]$ of hnodes. Alternative space filling curves may be adopted because the curve dependent aspects of *v(sf)* may be encapsulated into a function *next_son(N)*, that determines the next son of N to be visited. If implemented through a table look-up, *next_son(N)* is computed in a constant time.

The *computational load* of a hnode N is a problem dependent metric that evaluates the amount of computations on the elements in *space(N)*. According to the considered problem, the load can be *i)* constant during the computation and equal for all the hnodes, *ii)* constant during the computation but distinct for each hnode, or *iii)* variable during the computation and distinct for each hnode.

The *np* p-nodes of the distributed memory architecture are ordered too. A p-node P_j immediatly precedes P_k in the ordered sequence SP, if the cost of an interaction between P_j and P_k is not larger than the cost of the same interaction between P_j and any other p-node. Since each p-node executes one process, P_k also denotes the process executed on the k-th p-node of SP.

To preserve the ordering among the spaces, they are mapped by partitioning the hnodes through a blocking strategy. The sequence $S(v(sf))$ is partitioned into *np* segments, i.e. into *np* subsequences of consecutive hnodes. The overall load of a segment should be as close as possible to *average_load*, i.e. to the ratio between the overall computational load and the number of p-nodes. We cannot require that the load of each segment is equal to *average_load*, because each hnode, and its load, is assigned to one process only. In the following, $= (S, C)$, where S is a segment and C is a constant, denotes that the load of S is as close as possible to C. Due to the large number of elements, the difference between *average_load* and the assigned workload is negligible. Then, the first segment is mapped onto the p-node P_0, the second onto P_1 and so on. The resulting mapping satisfies the *range property*: *if the hnodes N_i and N_{i+j} are assigned to process P_h, then all the hnodes in-between N_i and N_{i+j} in $S(v(sf))$, are assigned to P_h as well.*

After the data distribution, each process P_h builds the replicated H-Tree and its private H-Tree. The private H-Tree of P_h includes a hnode N if *space(N)* is assigned to P_h. The replicated H-Tree is the union of the paths from the H-Tree root to the root of each private H-Tree. Each hnode N of the replicated H-Tree records the position of *space(N)* in the domain and the identifier of the owner process. In some problems, the intersection among the private H-Tree and the replicated H-Tree includes the roots of the private H-Tree only. In other problems, the private H-Trees and the replicated H-Tree are partially overlapped.

In all the problems that either emulates the evolution of a system or achieves the required accuracy of the results through iteration, the data mapping chosen at the *i*-th iteration could result in an unbalanced load at a later iteration. We define a procedure to correct an unbalance while minimizing the correspondly overhead. To detect when the procedure has to be applied, each process periodically broadcasts its workload, and it computes *max_unbalance*, the current maximum load unbalance, that is the largest difference between *average_load* and the workload of each process. If *max_unbalance* is larger than a tolerance threshold *T*, then each process executes the procedure. The threshold avoids that the procedure is executed to correct a very low unbalance. Let us suppose that the workload of P_i is *average_load* + *C*, $C > T$, while that of P_j, $i < j$, is *average_load* - *C*. To solve the unbalance P_i cannot send to P_j a set S of hnodes where $= (S, C)$ because the resulting allocation violates the range property. The correct procedure can be sketched as a shift of the spaces that involves all the processes in-between P_i and P_j. Let us define $Prec_i$ as the set of processes $[P_0 ... P_{i-1}]$ that precede P_i in the sequence SP, and $Succ_i$ as the set of processes $[P_{i+1} ... P_{np}]$ that follow P_i in SP. Furthermore, $Sbil(Prec_i)$ and $Sbil(Succ_i)$ are, respectively, the global load unbalances of the sets $Prec_i$ and $Succ_i$. If $Sbil(Prec_i) = C > T$, i.e. processes in $Prec_i$ are overloaded, P_i receives from P_{i-1} a segment S where $= (S, C)$. If, instead, $Sbil(Prec_i) = C < -T$, P_i sends to P_{i-1} a segment S where $= (S, C)$. The same procedure is applied to $Sbil(Succ_i)$ but, in this case, the hnodes are either sent to or received from P_{i+1}.

To respect the range property, if $[N_q .... N_r]$ is the segment of hnode it has been assigned, P_i sends to P_{i-1} a segment $[N_q .... N_s]$, with $q \leq s \leq r$, while it sends to P_{i+1} a segment $[N_t .... N_r]$, with $q \leq t \leq r$.

4 Fault Prevention

Each process computes the properties of all the elements in the spaces it has been assigned. To compute the properties of e, the process needs those of the neighbors of e that may have been allocated onto other p-nodes. A simple and general strategy to collect such remote data is *request/answer*. During the computation, as soon as P_h needs data of a space S assigned to P_k, it suspends the computation and sends a request to P_k. This strategy requires two communications for each remote data, one from P_h to P_k, and one from P_k to P_h.

To reduce this overhead, we introduce the *fault prevention* strategy. P_k, the owner of the space S, determines, through the neighborhood stencil, which processes require the data of S, and it sends to these processes the data, without any explicit request. To determine all the data to be sent to P_h, P_k exploits the information in the replicated H-Tree on the subspaces assigned to P_h. In general, P_k approximates the data that P_h requires, because the replicated H-Tree records a partial information only. The approximation is always safe, i.e. it includes any data P_h needs.

Fault prevention requires at most one communication for each other p-node to collect the remote data but, if the accuracy of the approximation is low, most of the data sent are useless. In some problems, the information in the replicated H-Tree enables P_k to determine a set of data that is not much larger than the one required by P_h. In other problems, instead, to improve the accuracy of the approximation, processes exchange some information about their private H-Trees before the fault prevention phase. Also the time to compute the data to be sent is rather important, because it cannot be larger than the one to explicitly request the data to the other p-nodes.

5 Adaptive Multigrid Methods

Adaptive multigrid methods are fast iterative methods based on multi level paradigms to solve partial differential equations in two or more dimensions [5], [6]. They may be applied to compute the turbulence of incompressible fluids [4], for macromolecular electrostatic calculation in a solvent [17], to solve plane linear elasticity problems [3] and so on.

The adaptive method builds the grid hierarchy during the computation, accordingly to the considered partial differential equation. Each grid of the hierarchy partitions the domain, or some parts of it, into a set of square spaces; the value of the equation is computed in the corners of each square. The hierarchy is built during the computation starting from an uniform grid, the level 0 of the hierarchy. Let us suppose that, at a level l, a square A has been discretized though the grid g. To improve the accuracy of the values in A, a grid finer than g is added at level $l+1$. Also the new grid represents A, but it doubles the points of g on each dimension. This doubles the accuracy of the discretization in A. As the computation goes on, finer and finer grids may be added to the hierarchy until the desidered accuracy has been reached in each square.

5.1 Multigrid Operators and V-cycle

In the following we consider the solution of the second order *Poisson differential equation* in two dimensions, subject to the *Dirichlet boundary conditions*:

$$-\frac{d^2u}{dx^2} - \frac{d^2u}{dy^2} = f(x,y) \qquad \Omega = 0 < x < 1,\ 0 < y < 1 \qquad (1)$$

$$u(x,y) = h(x,y) \qquad\qquad (x,y) \in \delta\Omega \tag{2}$$

To solve the considered equation, multigrid operators are applied on each grid of the hierarchy in a predefined order, the V-cycle. We briefly describe the main multigrid operators and the V-cycle; for a complete description see [6].

The *smoothing (or relaxation) operator* usually consists of some iterations of the Gauss-Seidel method or the Jacobi one and it is applied on each grid g to improve the approximation of the current solution on g.

The *restriction operator* maps the current solution on the grid at level l onto the grid at level *l-1*. The value of each point p on the grid at level *l-1* is a weighted average of the value of the neighbors of p in the grid at level l.

The *prolongation operator* maps the current solution on the grid at level *l-1* onto the grid at level l. If a point exists on both grids, its value is copied. The value of any other points at level l is an interpolation of the value of the neighbor of p on the grid at level *l-1*.

The *refinement operator*, if applied to a grid, or to a part of it, at level l, adds a new grid to the hierarchy at level *l+1*. The new grid represents the same square but it doubles the number of points on each dimension.

The V-cycle includes a downward phase and an upward one. The downward phase applies the smoothing operator to each grid, from the highest level to the one at level 0. Before applying this operator to the grid at level l, the restriction operator maps the values on the grid at level *l+1* onto the one at level l. The upward phase is symmetric to the downward one; the smoothing operator is applied to each grid, from that at level 0 to the highest level ones. Before applying the smoothing operator to a grid at level l, the prolongation operator maps the values of the grid at level *l-1* to the one at level l.

At the end of the V-cycle, the results are evaluated through an error estimation criteria. The refinement operator is applied to all the squares violating the criteria before starting another V-cycle.

5.2 Data Mapping

The resolution of partial differential equations through the adaptive method is an irregular problem, because the discretization of the domain and, consequently, the distribution of the points, are not uniform and not foreseeable. Moreover, adaptive methods are highly dynamic because the grid hierarchy is function of the considered domain.

The load balancing procedure should take into account two aspects of locality because the value of a point p on the grid g at level l is function of the values of the neighbors of p *i)* on the same grid g for the smoothing operator (intra-grid or horizontal locality) *ii)* on the grids at level *l+1* (if it exists) and *l-1* for the prolongation and restriction operators (inter-grid or vertical locality).

To apply the proposed methodology to this problem, the square spaces of all the grids of the hierarchy are ordered by visiting the domain through a space filling curve, and the are mapped as shown in sect. 3.

In the adaptive method, the hnodes at level l represent all the squares of the grids at level l, each hnode has either 4 sons or none and the squares associated to the sons of the hnode N represent the same square of N, but the number of points on each dimension is doubled.

To determine the computational load of an hnode we notice that the number of operations is the same for each point of a grid and does not change during the computation. Hence, the same computational load is assigned to each point, i.e. to each hnode, then we assign to each p-node the same number of squares.

In general, the domain subset assigned to each process is a set of squares that belong to grids at distinct levels. We denote by $Do(P_h)$ the subdomain assigned to process P_h. For each square it has been assigned, a process has to compute one point, the rightmost downward corner of the square.

The private H-Tree of the process P_h includes all the hnodes representing the squares assigned to P_h, while the replicated H-Tree includes all the hnodes on the paths from the root of the H-Tree to the roots of the private H-Trees .

A hnode can belong to more than one tree, because the computation is executed both on intermediate hnodes and on the leaves. To show that the replicated H-Tree and the private H-Tree are not disjoint, consider a hnode N assigned to the P_h. If one of the descendant of N has been assigned to P_k, $h \neq k$, N belongs to the private H-Tree of P_h, because P_h computes the value of the points in *space(N)*, and to the replicated H-Tree because it belongs to the path from the H-Tree root to the root of the private H-Tree of P_k.

5.3 Fault Prevention

Each process P_h applies the multigrid operators, in the order stated by the V-cycle, to the points of the squares in $Do(P_h)$.

Let us define $Pe(P_h)$, the boundary of $Do(P_h)$, as the sets of the squares in $Do(P_h)$ such that one of the neighbors does not belong to $Do(P_h)$. $Pe(P_h)$ depends upon the neighborhood relation of the operator *op* that is considered. To apply *op* to the points in $Pe(P_h)$, P_h has to collect the values of points in squares assigned to other processes. Let us define $I_{h,op,liv}$ as the set of squares of the domain not belonging to $Do(P_h)$ and including the points whose values are required by P_h to apply *op* to the points in the subgrid at level *liv* of $Do(P_h)$. Each of these squares belongs to $Pe(P_z)$, for some $z \neq h$. The values of points in $I_{h,op,liv}$ are exchanged among the processes just before the application of *op*, because they are updated by the operators applied before *op* in the V-cycle.

If fault prevention is adopted, P_h does not compute $I_{h,op,liv}$; instead, each process P_k determines the squares in $Pe(P_k)$ belonging to $I_{h,op,liv}$, $\forall h \neq k$. P_k exploits the information in the replicated H-Tree about $Do(P_h)$ to determine $I_{h,op,liv}$. Since this information could be not detailed enough, P_k computes an approximation AI of $I_{h,op,liv}$; in order to make a safe approximation P_k includes in AI all the points that could have a neighbor at level *liv* in $Do(P_h)$ according to the neighborhood stencil of *op*. Then P_k sends to P_h, without any explicit request, the values of the points in AI. To show that, due to the approximation, some of these values may be useless for P_h, suppose that at least one square in

$Do(P_k)$ belongs to $I_{h,op,liv}$, $h \neq k$, that $Do(P_k)$ and $Do(P_h)$ have been uniformly partitioned until, respectively level l and level l-m, and that the neighborhood stencil of op for the point p involves only the points on the same level of p. Since P_k does not know l-m, it could send to P_h some of its square on $Pe(P_k)$ at level higher than l-m that are useless for P_h because it has no point on these levels.

To reduce the amount of useless data, we introduce *informed fault prevention*. If P_h and P_k share a side, P_h sends to P_k, before the fault prevention phase, the depth of the deepest squares in $Pe(P_h)$ that could have a neighbor in $Pe(P_k)$, l-m in the previous example. This information allows P_k to improve the approximation of the set of points to be sent to P_h. The information on the depth of the squares in $Pe(P_k)$ is sent by P_k at the beginning of each V-cycle and it is correct until the end of the V-cycle, when the refinement operator may add a new grid. If the load balance procedure, that updates $Pe(P_k)$, is applied, then at the beginning of the V-cycle P_k has to send the depth of all the points on $Pe(P_k)$. Otherwise, since the refinement operator cannot remove a grid, each process has to send information on the new grids only.

The informed fault prevention technique is applied to the refinement operator too, but in this case the set of data to be sent to each process is always approximated. In fact, whether the process P_h, that owns the square of a point p, needs or not the square of the point q, owned by P_k, depends upon the value of p. Since P_h sends to P_k the depth of squares in $Pe(P_h)$, but not the actual values of the points, it could receive some useless values.

5.4 Experimental Results

We present some experimental results of the parallel version of the adaptive multigrid algorithm resulting from our methodology. The parallel architecture we consider is a Cray T3E; each p-node has a DEC 21164 EV5 processor and 128Mb of memory. The interconnection network is a torus. The programming language is C extended with the Message Passing Interface primitives [12].

The simulations solve two problems derived from the equation (1), with $f(x,y) = 0$ and two different boundary conditions (2), denoted by $h1$ and $h2$:

$$h1(x,y) = 10 \qquad\qquad h2(x,y) = 10\cos(2\pi(x-y))\frac{\sinh(2\pi(x+y+2))}{\sinh(8\pi)}$$

The solution of the Poisson equation is simpler than other equations such as the Navier-Stokes one. Hence, the ratio between computational work and parallel overhead is low and this is a significant test for a parallel implementation.

Figure 1 compares the remote data collecting techniques. We plot the total number of communications for request/answer (req/ans), for fault prevention (fp) and for informed fault prevention (ifp). In both problems, the number of communications of fault prevention and of informed fault prevention are, respectively, less than 61% and 52% than those of request/answer. As previously explained, because of the refinement operator, the number of communications of informed fault prevention is larger than 50% of the request/answer one, but the amount of useless data is less than 2%.

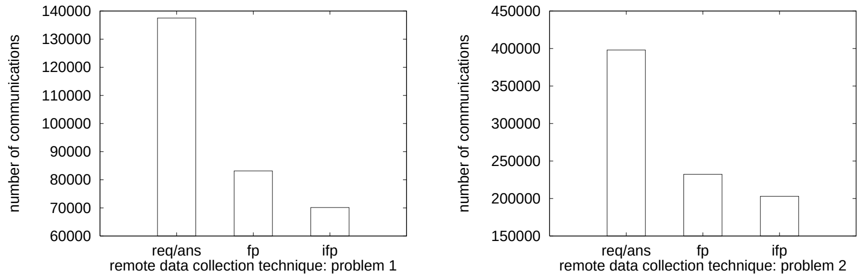

Fig. 1. A Comparison of remote data collection techniques

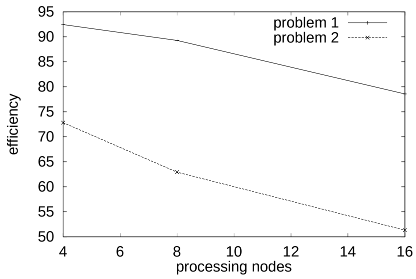

Fig. 2. Efficiency for problems with fixed data dimension

Figure 2 shows the efficiency of the parallel multigrid algorithm for the two problems, for a fixed number of initial points, 2^{14}, the same maximum grid level, 12, and a variable number of p-nodes. These simulations exploit informed fault prevention. The low efficiency resulting in the second problem is due to an highly irregular grid hierarchy. However, even in the worst case, our solution achieves an efficiency larger than 50% even on 16 p-nodes.

6 Conclusions

This paper has presented a methodology for the parallelization of irregular problems based upon the hierarchical structuring of the domain, a mapping strategy baesd upon space-filling curves and a technique, fault prevention, that reduces the communications overhead by preventing the data fault. This methodology has been previously applied to parallelize the n-body problem [1]. The experimental results of our numerical experiments show that this approach achieves good performances on high parallel distributed memory architectures.

We plan to extend the approach by considering other irregular pr s, such as hierarchical radiosity, to define a package that simplify the developement of parallel solutions to irregular pr s. A further development concerns the evaluation of our approach in the case of networks of workstations.

References

[1] F. Baiardi, P. Becuzzi, P. Mori, and M. Paoli. Load balancing and locality in hierarchical N-body algorithms on distributed memory architectures. *Lecture Notes in Computer Science*, 1401:284–293, 1998.

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

[3] P. Bastian, S. Lang, and K. Eckstein. Parallel adaptive multigrid methods in plane linear elasticity problems. *Numerical linear algebra with applications*, 4(3):153–176, 1997.

[4] P. Bastian and G. Wittum. Adaptive multigrid methods: The UG concept. In *Adaptive Methods – Algorithms, Theory and Applications*, volume 46 of *Notes on Numerical Fluid Mechanics*, pages 17–37, 1994.

[5] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[6] W. Briggs. *A multigrid tutorial.* SIAM, 1987.

[7] M. Griebel and G. Zumbusch. Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves. *Parallel Computing*, 25(7):827–843, 1999.

[8] P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, 1991.

[9] Y.S. Hwang, R. Das, J.H. Saltz, M. Hodoscek, and B.R. Brooks. Parallelizing molecular dynamics programs for distributed-memory machines. *IEEE Computational Science & Engineering*, 2(2):18–29, 1995.

[10] M. Parashar and J.C. Browne. On partitioning dynamic adaptive grid hierarchies. In *Proceeding of the 29th annual Hawaii international conference on system sciences*, 1996.

[11] J.R. Pilkington and S.B. Baden. Dynamic partitioning of non–uniform structured workloads with space filling curves. *Transaction on parallel and distributed systems*, 7(3):288–299, 1996.

[12] M. Prieto, D. Espadas, I.M. Llorente, and F. Tirado. Message passing evaluation and analysis on Cray T3E and SGI Origin 2000 systems. *Lecture Notes in Computer Science*, 1685:173–182, 1999.

[13] J.K. Salmon. *Parallel hierarchical N-body methods.* PhD thesis, California Institute of Technology, 1990.

[14] S. Sharma, R. Ponnusamy, B. Moon, Y. Hwang, R. Das, and J. Saltz. Runtime and compile-time support for adaptive irregular problems. In *Proceedings of Supercomputing*, pages 97–106, 1994.

[15] J.P. Singh. *Parallel hierarchical N-body methods and their implications for multiprocessors.* PhD thesis, Stanford University, 1993.

[16] J.P Singh, C. Holt, T. Totsuka, A. Gupta and J.L. Hennessy. Load balancing and data locality in adaptive hierarchical n-body methods: Barnes-Hut, Fatst Multipole and Radiosity *Journal of Parallel and Distributed Computing*, 27(2):118–141, 1995.

[17] Y.N. Vorobjev and H.A. Scheraga. A fast adaptive multigrid boundary element method for macromolecular electrostatic computations in a solvent. *Journal of Computational Chemistry*, 18(4):569–583, 1997.

[18] M.S. Warren and J.K. Salmon. A parallel hashed oct-tree N-body algorithm. In *Proceedings of Supercomputing '93*, pages 12–21, 1993.