A Load Balancing Technique for Multiphase Computations

Jerrell Watts
Syracuse University, Scalable Concurrent Programming Laboratory, jwatts@scp.syr.edu

Marc Rieffel
Syracuse University, Scalable Concurrent Programming Laboratory, marc@scp.syr.edu

Stephen Taylor
Syracuse University, Scalable Concurrent Programming Laboratory, steve@scp.syr.edu

A LOAD BALANCING TECHNIQUE FOR MULTIPHASE COMPUTATIONS

Jerrell Watts, Marc Rieffel and Stephen Taylor
Scalable Concurrent Programming Laboratory
Syracuse University
Syracuse, New York 13244
email: {jwatts, marc, steve}@scp.syr.edu

Keywords: parallel computing, dynamic load balancing, multiphase computations, irregular applications

ABSTRACT

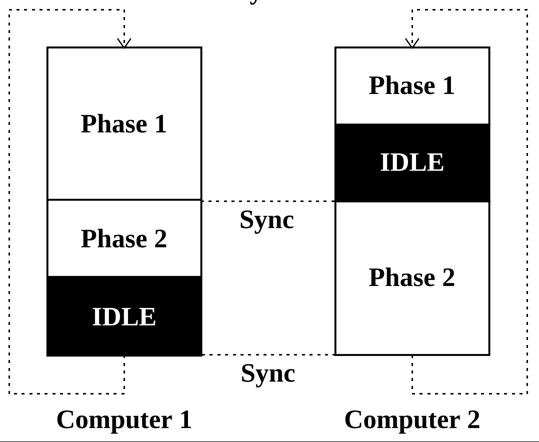
Parallel computations comprised of multiple, tightly interwoven phases of computation may require a different approach to dynamic load balancing than single-phase computations. This paper presents a load sharing method based on the view of load as a vector, rather than as a scalar. This approach allows multiphase computations to achieve higher efficiency on large-scale multicomputers than possible with traditional techniques. Results are presented for two large-scale particle simulations running on 128 nodes of an Intel Paragon and on 256 processors of a Cray T3D, respectively.

INTRODUCTION

Load balancing techniques already in the literature have concentrated entirely on single-phase computations (Boillat 1990; Cybenko 1989; Evans and Butt 1993; Heirich and Taylor 1995; Horton 1993; Kohring 1995; Lin and Keller 1987; Muniz and Zaluska 1995; Song 1994; Walshaw and Berzins 1995; Watts *et al.* 1996; Willebeek-LeMair and Reeves 1993; Williams 1991; Xu and Lau 1997). That is, they work only for applications which are comprised of a single mode of computation between synchronization points. Examples of such applications include Navier-Stokes flow solvers and particle simulations without self-consistent electromagnetic fields. As concurrent simulation techniques become more advanced, however, multiphase computations will appear with increasing frequency. Such applications involve two or more tightly interleaved computational phases separated by synchronization points. Applications of this type include particle simulations with self-consistent fields (such as particle-in-cell (PIC) techniques) and flow solvers using grid adaptation.

The reason why traditional load balancing techniques fail for multiphase applications is not immediately clear. A simple example illustrates the problem: Consider a two-phase computation running on only two computers. The first phase of computation takes 20 seconds on computer one and 10 seconds on computer two. The second phase of computation requires 10 seconds on computer one and 20 seconds on computer two. Most load balancing techniques, which do not consider the phases separately, would see each computer as having 30 total seconds of work; thus, the computation appears to be balanced. If a synchronization point exists between the phases, however, then the computation is not actually balanced. Computer two must wait 10 seconds for computer one during the first phase, and computer one must wait 10 seconds for computer two during the second phase. (See Figure 1.) Hence, the computation is only 75 percent efficient. Because the phases are brief, one cannot simply load balance whenever the phases alternate; the cost of load balancing would exceed the benefits it could provide. Instead, one must load balance in such a way that the phases are jointly balanced. To accomplish this, one can consider the loads of the two computers as vectors, rather than as scalars, where each component of the vector is the load of one of the computational phases. Then, by balancing the components of the vectors, one guarantees that the work for each individual phase of the computation is balanced, as well as the total work for all of the phases. For most load balancing algorithms, the modification to accommodate vectors

Figure 1: A "Balanced" System



is a fairly easy one.

This paper presents the vector load balancing technique as an effective mechanism for improving the efficiency of multiphase simulations. In particular, we applied the technique to a concurrent three-dimensional Direct Simulation Monte Carlo (DSMC) application (Rieffel *et al.* 1997). This application is used to model plasmas in situations ranging from reactor chambers used in silicon wafer fabrication to a satellite's reentry in the upper atmosphere. In the former simulations, electromagnetic fields are of particular importance. Since the workload for the field calculation is only a function of the number of grid cells assigned to a computer, whereas in the particle movement phase it is a function of both the number of grid cells and the number of particles, the two phases have very different load distribution properties. As a result, scalar load balancing techniques fail to improve the efficiency as much as one would expect. A similar problem occurred in a PIC simulation of ion thruster backflow around a satellite (Samanta Roy 1996). Like the DSMC application, the PIC simulation was comprised of two interwoven phases with very different load distribution characteristics. Scalar load balancing thus faired poorly in improving the efficiency of the computation. We demonstrate how vector load balancing techniques can circumvent these limitations.

BASIC METHODOLOGY

Our approach to dynamic load balancing is based on the following decomposition of the load balancing problem (Watts *et al.* 1996; Willebeek-LeMair and Reeves 1993)

1. **Load Measurement:** The load associated with each computer is determined, either by having the programmer provide an estimate of the workload associated with its tasks or by actually timing the amount of computation required by those tasks.
2. **Load Imbalance Detection and Profitability Calculation:** Based on the total load measured at each computer, the degree of load balance is calculated, and based on the estimated cost of load balancing, a profitability calculation is used to determine if the estimated improvement possible with load balancing exceeds that cost.
3. **Work Transfer Quantity Calculation:** Using the computer loads measured in the first step, the ideal amount of work to be transferred between pairs of computers is calculated. Work transfers equal to these pairwise, directed quantities should result in an improved load balance.
4. **Task Selection:** Using the work transfer quantities calculated previously, tasks are selected for transfer or exchange. This phase may be repeated several times until the transfer quantities have been adequately met.
5. **Task Migration:** Once the tasks' new locations are determined, any data structures associated with those tasks are transferred from their old locations to their new locations, and the computation resumes.

The above strategy is designed for single-phase computations. To adapt the methodology for multiphase computations, we must modify the methodology to incorporate the vector view of load.

VECTOR METHODOLOGY

This section presents the vector version of the dynamic load balancing framework presented above. Changes are required to the first four phases of the basic methodology.

Load Measurement

Load measurement becomes slightly more complex under the vector model. Under the scalar model, the load of a task could be readily obtained using standard operating system routines to measure its resource usage. While such facilities still prove useful for the vector model, additional effort is required to determine the appropriation of load to phases. The simplest way to do this is to have the programmer tell the load balancing system where each phase begins and ends by calling a routine before and after the code for that phase. More complex strategies might analyze the runtime between synchronization points, taking those portions with low statistical correlation or even anticorrelation to be separate phases.

Load Imbalance Detection and Profitability Calculation

Once the load for each phase has been totaled at each computer, the computers must communicate to detect the presence of a load imbalance. For scalar load methods, the global load balance is given by

$$eff = \frac{L_{\text{avg}}}{L_{\max}} = \frac{\sum_{i=0}^{P-1} L_i}{P \cdot \max_{i=0}^{P-1} L_i}$$

Equation 1: Scalar Efficiency

where L_i is the load of computer i of P total computers. Under the vector model, the efficiency equation changes

$$eff = \frac{\sum_j \vec{L}_{\text{avg},\,j}}{\sum_j \vec{L}_{\max,\,j}} = \frac{\sum_j \left(\sum_{i=0}^{P-1} \vec{L}_{i,j} \right)}{P \cdot \sum_j \left(\max_{i=0}^{P-1} \vec{L}_{i,j} \right)}$$

Equation 2: Vector Efficiency

where $\vec{L}_{i,j}$ is the load of phase j on computer i. Once the efficiency of the computation has been calculated, the profitability of load balancing is determined as described in (Watts *et al.* 1996; Willebeek-LeMair and Reeves 1993).

Work Transfer Quantity Calculation

If load balancing is deemed profitable, the ideal amounts of work to transfer between computers are calculated next. This calculation typically treats work to be a continuous quantity, denying its discrete nature until the next step. For most work transfer quantity calculations, such as those based on heat diffusion, the hierarchical balancing method, or the generalized dimensional exchange method (Boillat 1990; Cybenko 1989; Heirich and Taylor 1995; Horton 1993; Watts *et al.* 1996; Willebeek-LeMair and Reeves 1993; Xu and Lau 1997), the modification simply entails replacing all of the scalar load quantities in the calculation with vector load values. Alternatively, the scalar algorithm can be run separately for each component of the load vector.

Task Selection

When loads are taken to be scalars, task selection may consider either one-way transfers of tasks or two-way exchanges. (The latter allows one to potentially satisfy small ideal transfer quantities by exchanging two tasks of roughly equal load.) The vector load model, however, actually necessitates two-way exchanges since the components of the ideal work transfer may in general occur in different directions. (E.g., phase one work may need to be transferred from computer one to computer two, and phase two work transferred from computer two to computer one.) As a result, a more complicated task selection mechanism is typically required.

IMPLEMENTATION

The vector dynamic load balancing methodology described above was implemented in terms of the Scalable Concurrent Programming Library (SCPlib). This library is the successor to the Concurrent Graph Library (Taylor *et al.* 1996). The latter library was successfully applied to a number of large-scale industrial problems, including fluid flow calculations for the Titan IV and Delta II launch vehicles, the PIC ion thruster simulation described in the introduction and an earlier version of the DSMC application. SCPlib itself has been used to implement the DSMC application as well as the integrated field solver. This section briefly describes SCPlib and gives details of the implementation of each of the steps in the vector load balancing methodology.

SCPlib

SCPlib provides basic programming technology to support irregular applications on scalable concurrent hardware. The library provides architecture- and application-independent facilities for communication, synchronization, thread management and performance monitoring. The library runs on a wide variety of platforms, ranging from large-scale multicomputers such as the Intel Paragon and the Cray T3D, to medium-sized systems such as the Silicon Graphics PowerChallenge and Avalon A12, to networks of Unix workstations.

The SCPlib programming model is based on the concept of a concurrent graph of communicating tasks. A task is comprised of a thread of execution, a communication list and user state. The mapping of these tasks to computers is transparent to the user. This allows the mapping of tasks to computers to be changed at runtime, providing the basis for dynamic load balancing.

Also, under SCPlib, all communication and I/O occurs through objects called ports. These ports are roughly analogous to Unix descriptors in that the same routines can be used to write to a channel port or a file port. This allows considerable reuse of user application code; the same routines used to read and write a data structure can be used for communication and I/O (as well as load balancing, as described below).

Load Measurement

SCPlib provides routines that allow the user to mark the beginning and end of computational phases. Execution time between these marker routines is automatically accumulated into the appropriate counter. Another routine allows the user to extract these values and pass them on to the load balancing routine in the form of a load vector.

Load Imbalance Detection and Profitability Calculation

A load imbalance is detected by performing a global operation to calculate the average and maximum vector load over all of the computers. If the efficiency is less than that requested by the user, $eff_{\min}$, and if, based on previous load balancing attempts, the estimated improvement in runtime possible with load balancing exceeds its cost, load balancing is deemed profitable. (On the first attempt to load balance, SCPlib assumes that at least the efficiency requested by the user will be achieved, and that the cost of load balancing is zero.)

Work Transfer Quantity Calculation

Heat diffusion provides an intuitive, correct and scalable mechanism for calculating the volume of work to be transferred between computers (Boillat 1990; Cybenko 1989; Heirich and Taylor 1995; Horton 1993; Watts *et al.* 1996; Willebeek-LeMair and Reeves 1993; Xu and Lau 1997). Its inherent nearest neighbor approach also tends to preserve existing communications locality, and it is robust in the presence of asynchronous load injections. The vector version of the fully implicit, first-order accurate diffusion iteration which appeared in (Heirich and Taylor 1995) appears in Figure 2. In that figure, N_i is the set of network neighbors of computer i, $N_{\max}$ is the largest such set, and $\vec{T}_{(i,j)}$ is the work transfer quantity from computer i to computer j. SCPlib uses a vector version of the second-order accurate diffusion algorithm, based on the Crank-Nicholson scheme; this algorithm was presented in (Taylor *et al.* 1996). In both of these algorithms, each computer repeatedly exchanges load information with its neighbors in the physical network, updating its load and accumulating into the work transfer quantities based on a results of a simple arithmetic operation. The accuracy and termination point of the iteration are determined directly by the efficiency requested by the user in the initial call to the load balancing routine: $\alpha = 1 - eff_{\min}$

Figure 2: Diffusion Algorithm

```
diffuse(…)
    $m := \left\lceil \frac{\ln \alpha}{\ln \frac{|N_{\max}|\alpha}{1+|N_{\max}|\alpha}} \right\rceil$
    $\vec{T}_{(i,j)} := 0$ for all neighbors $j \in N_i$
    while $eff < eff_{\min}$ do
        $\vec{L}_i^{(0)} := \vec{L}_i$
        for $k := 1$ to $m$ do
            send $\vec{L}_i^{(k-1)}$ to all neighbors $j \in N_i$
            receive $\vec{L}_j^{(k-1)}$ from all neighbors $j \in N_i$
            $\vec{L}_i^{(k)} := \frac{\vec{L}_i^{(0)}}{1+|N_{\max}|\alpha} + \frac{\alpha}{1+|N_{\max}|\alpha} \sum_{j \in N_i} \vec{L}_j^{(k-1)}$
        end for
        $\vec{L}_i := \vec{L}_i^{(m)}$
        send $\vec{L}_i$ to all neighbors $j \in N_i$
        receive $\vec{L}_j$ from all neighbors $j \in N_i$
        $\vec{T}_{(i,j)} := \vec{T}_{(i,j)} + \alpha(\vec{L}_i - \vec{L}_j)$ for all neighbors $j \in N_i$
    end while
end diffuse
```

Task Selection

As pointed out previously, task selection is somewhat more complicated under the vector load model. As in the scalar case, the problem of selecting the subset of tasks whose exchange best approximates the ideal transfer quantity is **NP**-complete, by reduction from the subset sum problem. Fortunately, there exist fully polynomial-time approximation algorithms for the subset sum problem (Papadimitriou 1994). The load balancing framework in SCPlib uses a generalized version of such an approximation algorithm based on the notion of Euclidean distance in an n-dimensional space for n-dimensional load vectors. In short, the algorithm finds the subset of tasks whose exchange is the represented by the point closest (within the given tolerance) to the point represented by the ideal transfer vector. As the transfer quantity is fulfilled by repeated applications of the selection mechanism, the accuracy of the approximation algorithm is reduced accordingly.

Task Migration

SCPlib provides a routine which allows a task to relocate itself from one computer to another. Communication channels are automatically rerouted and any incoming messages forwarded appropriately. The protocol which does this is completely local in that only the computers on which the task originally resided, to which it is moving, and with which it directly communicates are informed of the move.

The task movement mechanism requires the user to supply routines that read and write the task's state to and from a port, as well as a routine to free the old state. Note that for applications that already support checkpointing, the same routines used to read and write a task's state to a file can be used to read and write its state between two computers. As the result of this reuse of existing code, we were able to integrate load balancing into the DSMC application within a few hours.

RESULTS

The dynamic load balancing framework described in the previous section was applied to a large-scale, concurrent particle dynamics simulation tool called Hawk (Rieffel *et al.* 1997). This application is based on a technique called Direct Simulation Monte Carlo (DSMC). The DSMC method solves the Boltzmann equation by simulating individual particles. Since it is impossible to simulate the actual number of particles in a realistic system, a smaller number of macroparticles are used, each representing a large number of real particles. The simulation of millions of these macroparticles is made practical by decoupling their interactions. First, the space through which the particles move is divided into a grid. (For complex geometries, this grid is typically made up of tetrahedra.) Particle collisions are considered only for those particles within the same grid cell. Furthermore, collisions themselves are not detected by path intersection but rather are approximated by a stochastic model whose parameters are the relative velocities of the particles in question. Statistical methods are used to recover macroscopic properties such as temperature and density. By limiting and simplifying the interactions in this fashion, the order of the computation is drastically reduced.

When the particles being simulated are ions, they contribute to and are influenced by an electromagnetic field. Consequentially, an electrostatic field solver has been incorporated into Hawk. This solver uses a face-based finite element (FEM) technique, taking as its input the charge density in each grid cell and returning the electric field for each cell (Watts, 1996). A preconditioned conjugate gradient method is used to solve the system of equations that results from the FEM.

The spatial decoupling at the core of the DSMC method makes it an ideal candidate for parallelization, since two partitions of grid cells interact only along their boundaries. The same applies to the field solver. The grid for a problem is prepared for parallel simulation by first dividing it into five to ten partitions per computer. Load balance is achieved by remapping these partitions to computers during execution. For the particle transport phase of Hawk, dynamic load balancing is necessary because the concentration of particles in a region of the grid changes during the course of the simulation. The grid may be refined in areas of high particle concentration in order to preserve the integrity of the DSMC model. This in turn affects the runtime of the field solver, which currently operates on the same grid used in the particle transport phase.

The necessity of dynamic load balancing to Hawk's efficiency was illustrated during a 1.2 million-particle simulation on a 124,000-cell grid of the Gaseous Electronics Conference (GEC) reactor. This simulation was run on 128 nodes of an Intel Paragon. Each node had approximately five partitions mapped to it. (Some of the partitions generated by the static partitioner were empty due to the simple geometric partitioning mechanism used. One advantage of the overall system is that the static partitioner can be quite simple since the burden of load balancing is borne by SCPlib.) The simulation was begun with the reactor empty, and particles were injected through one of its inlets.

As a result of the rapid load change during the early timesteps, the efficiency of the computation was quite low. Without load balancing, the first 150 timesteps required 1,762 seconds, with an efficiency of 31 percent. Scalar load balancing, which considered the particle transport and field calculations in aggregate, improved the situation somewhat: efficiency was improved to 45 percent, and execution time reduced to 1,217 seconds. (83 of those seconds were required by two attempts to balance the load.) Because the load distribution characteristics of the particle transport and field calculation phases were quite different, scalar load balancing failed to generate much of an improvement. Although the scalar load balance as given by Equation 1 was high (the library estimated it to be over 75 percent), the actual efficiency, in terms of available compute cycles being usefully employed, was much lower.

To use the vector load balancing mechanisms described here, load phase measurement calls were inserted before and after both the particle transport and field solver phases. The loads of these two phases where then passed on to the load balancing routine. Using the additional information provided by the vector loads, the efficiency improvement was much greater. The runtime was reduced to 787 seconds, for an overall efficiency of 70 percent. (98 seconds were required by two instances of load balancing.) The results of DSMC load balancing are summarized in Figure 3. One would expect the efficiency improvement and runtime reduction to be even greater as the problem approaches steady state, and the particle concentration begins to change less rapidly.

The advantages offered by vector load balancing were also seen in a preliminary implementation of vector load balancing under the Concurrent Graph Library (Taylor *et al.* 1996). In that case, the application in question was a particle-in-cell (PIC) simulation of ion thruster backflow around a satellite (Samanta Roy *et al.* 1996). Like the DSMC application, this simulation involved particle transport and field calculation phases. The primary differences between the two were that the PIC code simulated collisionless plasmas and used a regular grid on a simpler geometry. The satellite grid was divided into 1,575 partitions and mapped onto 256 processors of a Cray T3D. As with the GEC reactor simulation described above, the efficiency was quite low, at 54 percent. Scalar load balancing reduced the runtime for 100 timesteps of this simulation from 2,374 seconds to 2,014 seconds, for an improved efficiency of 63 percent. (The lack of efficiency improvement was independently noted for another concurrent PIC application, but the authors suggested no solution to the problem (Ferraro *et al.* 1991)). An even greater improvement in efficiency was offered by vector load balancing. The vector approach improved the efficiency to 72 percent, reducing the runtime to 1,775 seconds. A detailed breakdown revealed that vector load balancing had improved the field calculation efficiency from 73 percent to 94 percent and the particle transport efficiency from 29 percent to 43 percent. (These results are summarized in Figure 4.) A larger improvement in the particle transport efficiency was impossible because the load for that phase in one of the partitions was so high that no matter what computer it was assigned to, that computer was overworked, slowing down the entire computation. This suggests that in general, dynamic granularity control (i.e., the ability to divide and merge tasks at runtime) must be coupled with the vector load view to achieve load balance in all cases.

CONCLUSION AND FUTURE WORK

As the previous section shows, vector load balancing provides a superior alternative to scalar load balancing for multiphase computations. By considering the loads of such applications to be scalars rather than vectors, the load balancing framework is able to make the correct decisions to achieve higher load balance. Further improvements to the load balancing framework described herein will include the incorporation of dynamic granularity control to increase or reduce the task relocation options available to the load balancing mechanism. Also, extensions to the diffusion model and task selection mechanism will allow cases in which the computers are of heterogeneous processing capacity to be handled appropriately.

ACKNOWLEDGEMENTS

This research was sponsored by ARPA under contract number DABT63-95-C-0116. The first author is partially supported by a NSF Graduate Research Fellowship. Access to an Intel Paragon was provided by the Center for Advanced Computing Research at the California Institute of Technology. Access to a Cray T3D was provided by the NASA Jet Propulsion Laboratory and was facilitated by the California Institute of Technology.

Figure 3: DSMC Load Balancing Results

	Unbalanced	Scalar Balanced	Vector Balanced
Runtime (sec)	1,762	1,217	787
Efficiency	31%	45%	70%

Figure 4: PIC Load Balancing Results

	Unbalanced	Scalar Balanced	Vector Balanced
Runtime (sec)	2,374	2,014	1,775
Total Efficiency	54%	63%	72%
Field Solve Efficiency	73%	78%	94%
Particle Trans. Efficiency	29%	41%	43%

BIBLIOGRAPHY

Boillat, J. 1990. “Load Balancing and the Poisson Equation in a Graph.” *Concurrency: Practice and Experience,* vol. 6: 101-117.

Cybenko, G. 1989. “Dynamic Load Balancing for Distributed Memory Multiprocessors.” *Journal of Parallel and Distributed Computing,* vol. 7: 279-301.

Evans, D. and W. Butt. 1993. “Dynamic Load Balancing Using Task-Transfer Probabilities.” *Parallel Computing,* vol. 19: 897-916.

Heirich, A. and S. Taylor. 1995. “A Parabolic Load Balancing Algorithm.” In *Proceedings of the 24th International Conference on Parallel Programming,* vol. 3. CRC Press, 192-202.

Horton, G. 1993. “A Multi-Level Diffusion Method for Dynamic Load Balancing.” *Parallel Computing,* vol. 19: 209-218.

Ferraro, R.; P. Liewer; and V. Decyk. 1991. “Dynamic Load Balancing for a 2D Concurrent Plasma PIC Code.” Technical Report CRPC-91-6, Center for Research on Parallel Computing, Rice University, Houston, Tx.

Kohring, G. 1995. “Dynamic Load Balancing for Parallelized Particle Simulations on MIMD Computers.” *Parallel Computing,* vol. 21: 683-693.

Lin, F. and R. Keller. 1987. “The Gradient Model Load Balancing Method.” *IEEE Transactions on Software Engineering,* vol. 1:32-38.

Muniz, F. and E. Zaluska. 1995. “Parallel Load-Balancing: An Extension to the Gradient Model.” *Parallel Computing,* vol. 21:287-301.

Papadimitriou, C. 1994. *Computational Complexity.* Addison-Wesley, Reading, Ma.

Rieffel, M.; S. Shankar; S. Taylor; and J. Watts. 1997. “Concurrent Simulation of Plasma Reactors.” To appear in elsewhere in these proceedings.

Samanta Roy, R.; D. Hastings; and S. Taylor. 1996. “Three-Dimensional Plasma Particle-in-Cell Calculations of Ion Thruster Backflow Contamination.” *Journal of Computational Physics,* vol 128:6-18.

Song, J. 1994. “A Partially Asynchronous and Iterative Algorithm for Distributed Load Balancing.” *Parallel Computing,* vol. 20:853-868.

Taylor, S.; J. Watts; M. Rieffel; and M. Palmer. 1996. “The Concurrent Graph: Basic Technology for Irregular Problems.” *IEEE Parallel and Distributed Technology,* vol. 4:979-993.

Walshaw, C. and M. Berzins. 1995. “Dynamic Load-Balancing for PDE Solvers on Adaptive Unstructured Meshes.” *Concurrency: Practice and Experience,* vol. 7:17-28.

Watts, J.; M. Rieffel; and S. Taylor. 1996. “Practical Dynamic Load Balancing for Irregular Problems.” In *Parallel Algorithms for Irregularly Structured Problems: IRREGULAR `96 Proceedings.* Springer-Verlag LNCS, vol. 1117:299-306.

Watts, J. 1996. “An Electrostatic Field Solver for Concurrent Direct Simulation Monte Carlo.” Technical Report TR-93-03. Department of Computer Science, California Institute of Technology, Pasadena, Ca.

Willebeek-LeMair, M. and A. Reeves. 1993. “Strategies for Dynamic Load Balancing on Highly Parallel Computers.” *IEEE Transactions on Parallel and Distributed Systems,* vol. 4:979-993.

Williams, R. 1991. “Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations.” *Concurrency: Practice and Experience,* vol. 3:457-481.

Xu, C. and Lau F. 1997. *Load Balancing in Parallel Computers: Theory and Practice.* Kluwer, Boston, Ma.