Performance Analysis of Parallel Strategies for Localized N-body Solvers

Silvia M. Figueira [*] Scott B. Baden [†]

Abstract

Although there exist several approaches to rapidly solving the N-body problem, and a diversity of implementation strategies, the performance tradeoffs of the various strategies with respect to problem-specific data distributions is poorly understood on a parallel computer. We present a synthetic workload model and a simulator that enables us to evaluate the performance tradeoffs encountered in implementing particle methods on MIMD computers. These results can be used to evaluate designs early on in the implementation process.

1 Introduction

We present a comparative performance analysis of various strategies for implementing localized N-body solvers on MIMD distributed memory parallel computers. In localized N-body solvers, particle interactions beyond a certain *cutoff radius* are either ignored or computed separately. Often the short range interactions predominate. We will consider only local interactions, i.e., each particle interacts over a small neighborhood only. Our results apply to single-level particle methods in which the particle density variation is not too great; thus we exclude hierarchical methods [3, 5].

A synthetic workload model was developed that characterizes an N-body system in terms of its properties. We built a simulator on the Intel Paragon that takes the workload parameters as inputs, builds the corresponding synthetic workload, and models the cost of evolving the particles over a sequence of timesteps. The simulator implements three strategies for the calculation of the forces: link-cell; link-cell with per-particle lists of neighbors; and link-cell with per-cell lists of neighbors. The effects caused by the parameters on the performance of the strategies and the costs associated with different kinds of N-body systems using different implementation strategies are presented.

Section 2 presents the synthetic workload model, and section 3 describes the implementation strategies implemented by the simulator. Section 4 presents computational results, and section 5 discusses these results.

2 Synthetic Workload Model

We have developed a synthetic workload model that characterizes an N-body system in terms of various properties, including: number of particles; average particle density $\bar{\rho}$;

[*]Graduate student, Dept. of Computer Science and Engineering, U.C. San Diego, USA. Electronic mail address: silvia@cs.ucsd.edu, supported by a grant from CAPES (Federal Agency for Graduate Education, Brazil). Computer time on the San Diego Supercomputing Center's Intel Paragon was provided by a UCSD Division of Engineering Block Grant.

[†]Assistant Professor, Dept. of Computer Science and Engineering, U.C. San Diego, USA. Electronic mail address: baden@cs.ucsd.edu, supported by ONR contract N00014-93-1-0152.

distribution (uniform or Gaussian); maximum displacement of the particles in one timestep Δ_{MAX}; cutoff radius c; computational cost, in FLOPS, of an interaction between two particles Φ; and the amount of memory required per particle η. For Gaussian distributions, there is an additional parameter characterizing the density of the distribution. Using this model we may construct synthetic particle workloads that are representative of real world distributions with characteristics requiring different implementation strategies.

We built a simulator on the Intel Paragon that takes the aforementioned workload parameters as inputs, builds the corresponding synthetic workload, and models the cost of evolving the particles over a sequence of timesteps. At each timestep, the simulator transmits all required off-processor dependence data, and simulates the computation of local direct interactions with an idle loop. The time evolution of the particles is simulated using a random walk, determined by Δ_{MAX}.

3 Implementation Strategies

The force computation is organized around the procedure for enumerating interactions. Fundamentally, two approaches have been taken: the link-cell method (also called "chaining mesh" [6]), and the Verlet Neighbor List method [12]. The link-cell method uses a mesh to organize the computation. Each box of the mesh contains a list with the particles found in its corresponding subregion of the domain, and the interacting neighbors lie in neighbor boxes, avoiding a costly $O(N^2)$ search for neighbors. After the new positions of the particles are computed at the end of each timestep, any particle that has moved to a new box is *repatriated* to its proper owner.

A problem with the link-cell method is that traversal of the mesh can be expensive. The Verlet Neighbor List method can avoid the difficulty, by creating a *neighbor list* for each particle. The neighbor list is periodically reconstructed, but less frequently than every timestep, in order to reduce the cost. This implies that we must expand the neighbor list out to a second *extended* cutoff c_x to include particles that *may* move to within c before the list is next reconstructed.

When calculating the forces, only particles within c are considered. The remaining particles are ignored, though a distance computation is required to identify them. If the cost of computing an interaction is large (i.e. 100 flops) as compared with the distance check ($3k$ flops in k dimensions), then the cost of skipping over spurious neighbors may not be noticed, in other cases the overhead may be significant. The choice of the extended cutoff c_x and of the *list reconstruction rate* τ depend on Δ_{MAX}, and must be adjusted to minimize the cost of reconstructing the lists and of handling spurious interactions, subject to the constraint that no particle initially outside c_x can move inside c between list reconstruction phases.

A disadvantage of the Verlet Neighbor List method is that an $O(N^2)$ search is required to construct the neighbor lists. We may employ a link-cell mesh to accelerate the search to an $O(N)$ sort and $O(N)$ search, which we call the "link-cell with per-particle lists of neighbors" approach. This approach avoids the repatriation of particles at every timestep, as required by the link-cell method; the particles need to be repatriated only when lists are reconstructed.

A major issue in implementing particle methods is to avoid excessive spurious interactions. For the link-cell method we must choose an appropriate mesh spacing; for the hybrid (link-cell with Verlet lists) method we want to keep c_x small. As c_x increases so does the amount of off-processor dependence data transmitted between processors, which

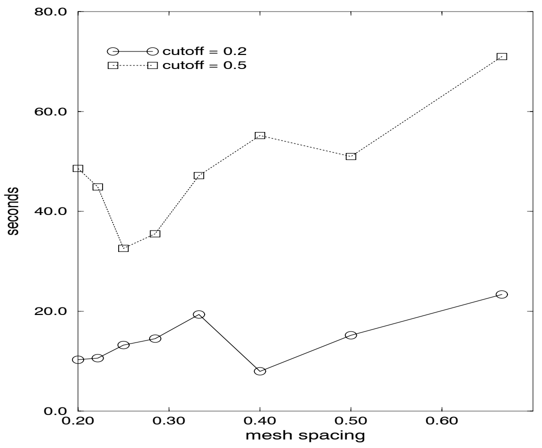

Fig. 1. *Cutoff radius affecting the optimum mesh spacing in strategy 1.*

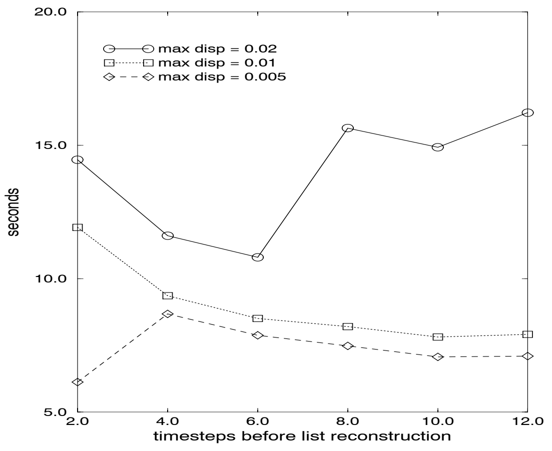

Fig. 2. *Maximum displacement of the particles affecting the optimum list reconstruction rate in strategy 2.*

incurs both a time penalty and a space penalty, to store the off-processor data.

The size of the lists depend on the $\bar{\rho}$ and c: distributions with a low $\bar{\rho}$ or small c require shorter lists than distributions with high $\bar{\rho}$ or large c, because the number of neighbors is smaller. In the latter case we may need to conserve space by using a single neighbor list for each bin, such that all particles in a bin share the same list. We refer to this as the "link-cell with per-cell lists of neighbors" approach. This approach increases the cost of computing interactions, since the number of spurious neighbors increases. Whether the space-time tradeoff offered by this strategy is useful depends on the circumstances, but generally finer grained, per-particle lists will be faster.

In this paper we will consider three implementation strategies for the calculation of the forces: (1) link-cell; (2) link-cell with per-particle lists of neighbors; and (3) link-cell with per-cell lists of neighbors. To parallelize these computations, we employ spatial decomposition to partition the link-cell mesh into contiguous simply-connected regions. As compared with methods that partition on the basis of particles or interactions, spatial decomposition strategies avoid global communication, achieving better scalability [10].

Strategies 1 and 2 have been used to calculate the non-bonded forces in Molecular Dynamics systems, like EulerGROMOS [4], and have been discussed by Tamayo, Mesirov and Boghosian [11], and by Plimpton and Heffelfinger [10]. Strategy 1 has also been employed in vortex dynamics calculations [2].

4 Results

Our simulator was written in C++ (gcc compiler version 2.5.7) and runs under LPARX (version 1.1) [7]. It has been used in experiments simulating three dimensional N-body systems on 16 processors of the Intel Paragon running the operating system Paragon OSF/1 R1.2. The peak bandwidth of the Paragon is around 70 MB/sec, and the average message startup cost is 100 microseconds [9].

We explore the effect that workload variations have on the performance of the different implementation strategies. We varied c and Δ_{MAX}. All simulations were conducted with 5000 particles and run for 50 timesteps, unless otherwise specified. The maximum number of flops per interaction was 19 and the number of bytes of storage per particle was 72. Dynamic recursive bisection was used for load balancing [1].

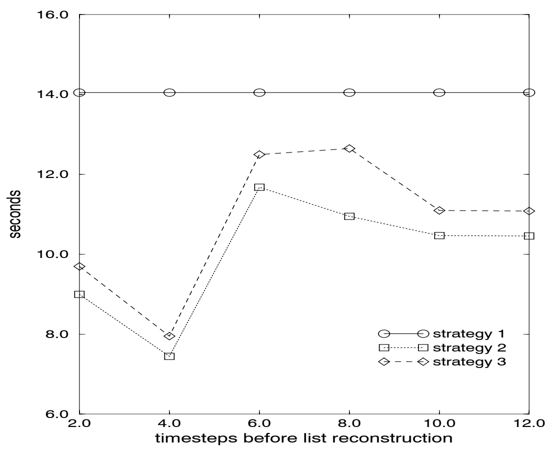

FIG. 3. *Execution time of strategies 1, 2 and 3. The short displacement of the particles favors strategies 2 and 3.*

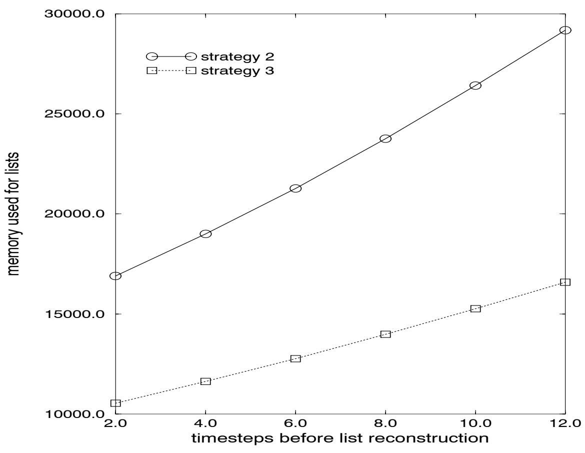

FIG. 4. *Memory required (in total number of neighbors in a timestep) for the lists, in strategies 2 and 3.*

Our main result is that no one implementation strategy excels under all conditions, and that there exist conditions under which each strategy is preferred to the others.

Our first observation is that performance is sensitive to the mesh spacing as shown in figure 1. The mesh spacing must be adjusted to minimize the cost of traversing the mesh and of treating spurious neighbors: if the mesh is too fine, mesh traversal is expensive, if too coarse, the number of spurious neighbor particles is large. The optimum mesh spacing is affected by c (c_x, for list strategies). This can be seen by noting the minima on the curves in figure 1, which presents data for two different distributions each with a different cutoff radius. The distributions in this figure have particles uniformly distributed in a domain of size 4x4x4, and a large Δ_{MAX} (0.02).

For non-uniform systems, the average number of particles in each box of the mesh varies. In this case, a non-uniform mesh, i.e., a mesh with different sizes of boxes, might help improve the performance. We have not tried that though.

The performance of strategies 2 and 3 is influenced by the list reconstruction rate τ. If τ is high, the lists are rebuilt expensively many times. If low, then c_x is large, and the traversal of both the mesh and the lists is expensive. The optimum value of τ is related to Δ_{MAX} and $\bar{\rho}$. Short displacements and low densities favor low τ, because few extra neighbors need to be added to the lists; large displacements and high densities favor high τ. Figure 2 shows the effect of τ on the performance of strategy 2, and the effect of Δ_{MAX} on the optimum value of τ. It presents the simulation of three distributions, each with a different Δ_{MAX}. The particles were generated uniformly in a 4x4x4 domain, and $c = 0.2$.

We next make a direct comparison of the three different strategies, using two different distributions. The first distribution has short displacements ($\Delta_{MAX} = 0.002$), and the second one has large displacements ($\Delta_{MAX} = 0.02$). Both distributions are dense (5000 particles, generated according to a Gaussian distribution with mean 0.0 and variance 1.0, in a domain of size 3x3x3) and have $c = 0.2$.

Figures 3 and 4 show respectively the time and space performance of each strategy with the short displacement distribution. Both strategy 2 and 3 are faster than strategy 1; Δ_{MAX} is small, so is c_x and hence the neighbor lists are not too long either. Strategy 2 and 3 have very similar running times—the former is a little faster, owing to the shorter lists—but strategy 3 requires less storage and hence it can accommodate larger workloads.

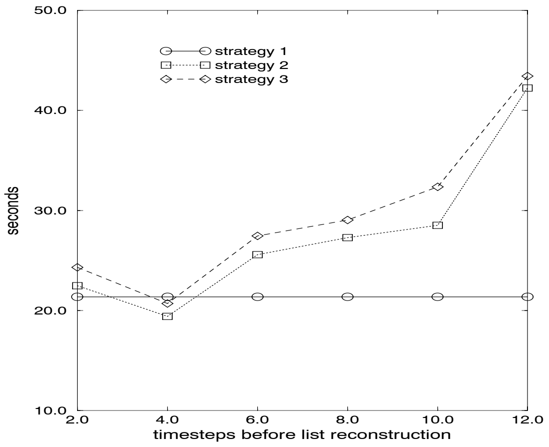

Fig. 5. *Execution time of strategies 1, 2 and 3. The large displacement of the particles favors strategy 1.*

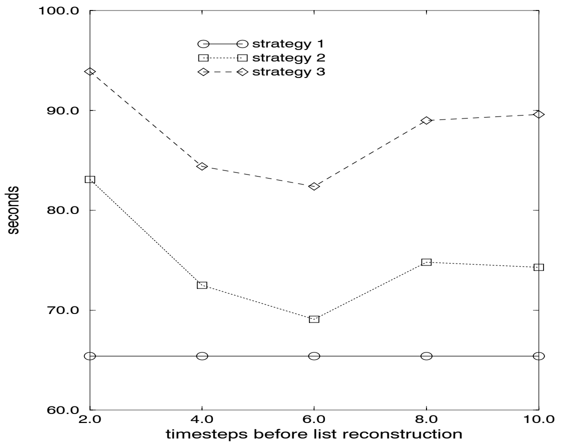

Fig. 6. *Execution time of strategies 1, 2 and 3 in a simulation of smoothed particle hydrodynamics in 3 dimensions.*

However, τ was fairly high owing to the high density of the distribution.

Figure 5 shows the time performance of each strategy with the large displacement distribution. The three strategies achieved similar time performance when the lists were reconstructed frequently— every 4 timesteps. Lists do not enhance the performance of the link-cell strategy in this example, because the large displacements force a large c_x, which increases the size of the lists and the cost of handling them. Since strategy 1 does not require storage for lists, it is preferred in this case.

5 Conclusion

Experiments have shown that, in general, there is no strategy that is the best in all circumstances. In fact, each strategy can perform better than the others depending on the properties of the distribution.

Δ_{MAX} is the critical factor when selecting an implementation strategy. Short displacements generally favor list strategies over the link-cell method, but list strategies perform poorly when simulating large displacement distributions, in which case the link-cell method may be more appropriate. Experiments have also shown the importance of choosing the optimum mesh size and neighbor list reconstruction rate.

Another important factor is the number of processors: increasing the number of processors may benefit the link-cell strategy more than the list strategies, because the gain is more significant in the calculation of forces and traversal of the mesh, which is more expensive for the link-cell strategy. This should be taken into consideration when selecting a strategy because, for some systems, the list strategies may be the best choice with few processors, but the link-cell strategy may be more appropriate if a larger number of processors is used.

An important issue when employing a list strategy is storage. If the list reconstruction rate is low, the lists are long and more storage is required. Sometimes the speedup obtained by lowering the list reconstruction rate is not worth while, because a high storage overhead is incurred. This happens when the $\bar{\rho}$ is low and Δ_{MAX} is small. When running times are comparable, the link-cell strategy is preferred to neighbor lists. We introduced a coarse-grain Verlet list, that relies on a link-cell mesh to maintain a single neighbor list for all particles in a given mesh box. While this approach is generally slower than the traditional (fine-grain) Verlet list, it offers reduced storage overheads.

To validate our model, we have compared the performance of the strategies in a computation of smoothed particle hydrodynamics in 3 dimensions [8, 7] on the Intel Paragon. The computation was written in mixture of C++ and Fortran 77 and ran under LPARX. The initial data consists of 12190 particles distributed in a disk with a hole in the center and declining density; they are assigned a circular velocity. The domain is a box of size approximatedly 5x5x1, Δ_{MAX} is large (0.021), and $c = 0.2$. Figure 6 shows the execution times for each strategy. In this case, the use of lists does not enhance the performance of the link-cell strategy, because of the large displacements. As predicted by our model, the link-cell method 1 is preferred in this case.

Our results show that many factors and choices affect the performance of the strategies. Our simulator proved to be useful in tuning various simulation parameters, and more importantly can be used to select an appropriate implementation strategy early on in the code development process.

References

[1] S. Baden, *Programming Abstractions for Dynamically Partitioning and Coordinating Localized Scientific Calculations Running on Multiprocessors*, SIAM J. on Sci. and Stat. Comput., 12 (1991), pp. 145–157.

[2] S. Baden, *Very Large Vortex Calculations in Two Dimensions*, Proceedings of the UCLA Workshop on Vortex Methods, May, 1987.

[3] J. Barnes and P. Hut, *A Hierarchical O(NlogN) Force Calculation Algorithm*, Nature, 324 (1986), pp. 446–449.

[4] T. Clark, R. Hanxleden, J. McCammon and L. Scott, *Parallelizing Molecular Dynamics using Spatial Decomposition*, Proceedings of the 1994 Scalable High Performance Computing Conference, May, 1994, pp. 95–102.

[5] L. Greengard and V. Rokhlin, *A Fast Algorithm for Particle Simulations*, Journal of Computational Physics, 73 (1987), pp. 325–348.

[6] R. Hockney and J. Eastwood, *Computer Simulation using Particles*, McGraw-Hill Inc., 1981.

[7] S. Kohn and S. Baden, *A Robust Parallel Programming Model for Dynamic Non-Uniform Scientific Computations*, Proceedings of the 1994 Scalable High Performance Computing Conference, May, 1994, pp. 509–517.

[8] J. Monaghan, *Smoothed Particle Hydrodynamics*, Annual Review of Astronomy and Astrophysics, 30 (1992), pp. 543–574.

[9] P. Pierce and G. Regnier *The Paragon Implementation of the NX Message Passing Interface*, Proceedings of the 1994 Scalable High Performance Computing Conference, May, 1994, pp. 184–190.

[10] S. Plimpton and G. Heffelfinger, *Scalable Parallel Molecular Dynamics on MIMD Supercomputers*, Proceedings of the 1994 Scalable High Performance Computing Conference, April, 1992, pp. 246–251.

[11] P. Tamayo, J. Mesirov and B. Boghosian, *Parallel Approach to Short Range Molecular Dynamics Simulations*, Proceedings of Supercomputing 91, November, 1991, pp. 462–470.

[12] L. Verlet, *Computer "Experiments" on Classical Fluids*, Physical Review, 159 (1967), pp. 98–103.