NOTE

N-Body Simulations on Massively Parallel Architectures

Simulating the properties of N particles, mutually interacting through a pair-wise force, is the most common and important computational physics problem in classical statistical mechanics. These simulations yield information that is inaccessible by other means and that leads to insight and predictive behavior for a wide range of problems and properties. In this note, we will report on our discovery of several simple procedures that reduced by an order of magnitude the computation time of our implementation of such a simulation on the massively parallel CM-200. The simple philosophy behind these procedures might be useful for optimizing similar simulations on the CM-200 and other massively parallel machines.

In N-body simulations, the positions of the particles X_i (and their momentum P_i) are evolved in time by numerically integrating Newton's (Hamilton's) equations of motion or Langevin's equations for Brownian dynamics. In either case, the problem reduces to solving a system of first-order differential equtions similar to

$$\frac{dP_i}{dt} = G_i + H_i, \tag{1}$$

where G_i is the net force on particle i due to its interaction with all other particles. This force is defined by

$$G_i = \sum_{j \neq i} F_{ij} \tag{2}$$

with F_{ij} being the pair-wise force between particles i and j and H_i being the net force on particle i due to all other interactions. Often, F_{ij} depends only on the distance between the particles, i.e., $F_{ij} = F(|X_i - X_j|)$. This force might be, for example, Coulomb's law or a more phenomenological law like the Lennard–Jones interaction. The H_i may be, for example, external fields or random forces simulating contact with a heat bath. The equations of motion can be integrated by a variety of means that are embarassingly parallel. The computational bottleneck is the calculation of G_i.

Two frequently used classes of techniques for computing G_i are tree methods and direct methods. The tree methods recursively decompose the system of particles into subsystems and express the interaction between the subsystems by a multipole expansion [1]. These methods are particularly suitable for large systems because their asymptotic complexity is proportional to $O(N \log(N))$ for adaptive methods [1–4] or proportional to $O(N)$ for non-adaptive methods [1]. The direct method, on the other hand, simply sums the forces between all pairs of particles so its complexity is proportional to $O(N^2)$. However, the proportionality constant in the direct method is smaller than that for the multipole methods.

The direct method is often used instead of the asymptotically more efficient multipole method because

1. The multipole expansion may be unknown.
2. The system is sufficiently small that the direct method is faster than the multipole method.

In addition, at some sufficiently fine granularity in the decomposition in the multipole method, the direct method becomes faster because of the lower proportionality factor and, thus, becomes preferred. This crossover generally is true at the leaves in a multipole method. Thus, efficient implementation of the direct method is an essential aspect of any direct net force computation.

The parallel calculation of the net force has a computation part and a communication part. The computation part is mainly concerned with the calculation of F_{ij}. Its optimization, which normally involves the computation of some function of the distance between X_i and X_j, is essentially architecture independent. On a parallel machine, once the distance between particles is known, this force can often be computed with no inter-processor communication.

The optimization of the communication part of the direct solver is the main result of this note. It depends on the structure of the direct solver. Two common methods are used: one is the "all-to-all broadcast method" [5, 6]; the other is what we will call the "Fortran 90 method."

The all-to-all broadcast method presumes that the locations X of the particles are distributed throughout the nodes of the architecture. A local copy Y of X is made; then for each Y_i a Hamiltonian path through the processor network is computed. The location of each particle is then successively routed along its path which ensures that each particle location will eventually be transmitted through each processor. The force on a particle is computed simply by summing the pair-wise interactions of all the particles that pass through the processor in which it is stored. For systems with a small number or particles, this method performs poorly on the CM-200 because of processor under-utilization.

The Fortran 90 method involves the use of standard Fortran 90 intrinsics, such as SUM and SPREAD. If A is a matrix, then SUM(A, DIM = 2) is the vector whose ith element is the sum of the ith row of A. If X is an N-dimensional vector, then SPREAD(X, DIM = 2, NCOPIES = N) is the $N \times N$ matrix A whose ijth element is X_i. Similarly, SPREAD(X, DIM = 1, NCOPIES = N) is the matrix whose ijth component is X_j. The Fortran 90 method for computing the net force on each particle in a one-dimensional system is shown in the following pseudo-code:

```
$S1 = \mathtt{SPREAD}(X, \mathtt{DIM} = 1, \mathtt{NCOPIES} = N)$
$S2 = \mathtt{SPREAD}(X, \mathtt{DIM} = 2, \mathtt{NCOPIES} = N)$
$R = \mathtt{ABS}(S2 - S1)$
$F = \mathtt{FUNC}(R)$
$G = \mathtt{SUM}(F, \mathtt{DIM} = 2)$
```

The ijth element of the matrix R is the distance between particles at positions X_i and X_j. This matrix is then used to obtain the forces F_{ij} from the user-defined function FUNC. Finally, SUM is used to sum the forces.

Although the Fortran 90 method has the advantage of portability to any Fortran 90 platform, the communication functions SUM and SPREAD can be fairly slow. Furthermore, the SPREAD and SUM syntax seems to lack a certain naturalness. The first optimization trick that we tried was to replace the communication operations of the Fortran 90 method with inner and outer products. Since these products are basic vector (matrix) computational tools, we thought it was reasonable to assume that they would be well optimized on the CM-200. The outer product of two vectors X and Y of length N is the $N \times N$ matrix whose ijth element is $X_i * Y_j$. The inner product of these two vectors is the number SUM($X * Y$). Such an operation is intrinsic to matrix multiplication. Letting U be a vector whose components are all unity, we then observe that SUM(A, DIM = 2) is equivalent to the Fortran 90 expression MATMUL(A, U). Similarly, we note that SPREAD(X, DIM = 2, NCOPIES = N) is the outer product of X and U and SPREAD(X, DIM = 1, NCOPIES = N) is the outer product of U and X. We can rewrite our pseudo-code as

```
$U = 1$
$S1 = \mathtt{OPROD}(U, X)$
$S2 = \mathtt{OPROD}(X, U)$
$R = \mathtt{ABS}(S2 - S1)$
$F = \mathtt{FUNC}(R)$
$G = \mathtt{MATMUL}(F, U)$
```

In Table I, we present a number of timings. We first observe that MATMUL is up to 20 times faster than SUM. At first glance, this speed differential is surprising since MATMUL does more "FLOPS" than SUM. On the other hand, efficient SUM or MATMUL routines can involve numerous extremely complex pipelining and synchronization issues and are quite difficult to write efficiently in microcode. Because MATMUL is a more basic scientific operation than SUM, presumably more effort went into tuning it than into tuning SUM. Since the outer product is not part of the Fortran 90 standard, we used a library routine from the CM-200's CMSSL package [7]. Here, we denote this subroutine as OPROD. We see from Table I that a call to this routine can be up to six times faster than the call to SPREAD.

TABLE I

Timings in Milliseconds for Certain Fortran 90 Intrinsics versus Equivalent Algebraic Formulation for Varying N

N	1024	2048	4096	8192
SUM	19.8	70.9	275	—
MATMUL	4.21	7.39	11.6	28.4
SPREAD	13.1	43.7	159	622
OPROD	5.59	11.0	28.7	100.0

Note. Timings were obtained on one sequence (512 nodes) of a CM-200 running slicewise CM Fortran 1.1. SUM failed due to insufficient memory on the $N = 8K$ problem.

The final optimization that we made involved special purpose microcode but is based on the simple observation the displacement $X_i - X_j$ between two particles can be interpreted as the outer sum of the vectors X and $-X$, which could be computed by trivially replacing the multiplication call in the outer-product routine with an addition call. Thinking Machines Corporation provided this modification. (The execution time for the outer-sum routine is the same as that of the outer-product routine.) Using this routine, we can replace the two outer-product calls in our code by one outer-sum call, and our pseudo-code becomes

```
$U = 1$
$D = \mathtt{OSUM}(X, -X)$
$R = \mathtt{ABS}(D)$
$F = \mathtt{FUNC}(R)$
$G = \mathtt{MATMUL}(F, U)$
```

Compared to our original pseudo-code, which was essentially our actual original Fortran 90 coding, this new code reduced our computation time by a factor of 10.

In conclusion, we formulated the direct net force computation in the N-body problem in terms of fast primitives that are well adapted to the massively parallel architecture of the CM-200 [8]. The communication overhead of the direct N-body solver was reduced by one order of magnitude. The method consisted in replacing Fortran 90 intrinsics by inner- and outer-product functions. In one case, we used a routine in which a small modification to the library outer-product routine was made to convert it to an outer-sum routine. Although the use of this routine[1] negatively impacts portability, we used it because it should be very easy to replace it with a comparable optimized function call on other computers and because it is intrinsically elegant. The technique was implemented and tested on a molecular dynamics problem geared towards the study of flux line dynamics in superconducting thin films. Results of this calculation will be published elsewhere [9].

While our observations are, strictly speaking, specific to the CM-200, we feel that the overall philosophy of using linear algebra concepts to replace the thinking required to use Fortran 90 intrinsics bears consideration in formulating efficient coding strategies for parallel computing [10]. Linear algebra concepts match nicely to vector and parallel computation. In addition, the scientific subroutine libraries that accompany parallel computers, such as the CMSSL library for the CM-200, are likely to have highly optimized codes to execute the basic operations. With minor experimentation, significant gains in performance, such as the one reported here, might be possible both on the CM-200 and other massively parallel computers.

[1] The outer-sum routine is available from the lead author of this note.

ACKNOWLEDGMENTS

One of us (L.S.) would like to thank both the Center for Nonlinear Studies and the Advanced Computing Laboratory at the Los Alamos National Laboratory for their hospitality while this work was conducted. We thank the Advanced Computing Laboratory for the use of its computational facilities. We also thank the Thinking Machines Corporation for providing us with the outer-sum microcode. This work was supported by the U.S. Department of Energy.

REFERENCES

1. L. Greengard, "The Rapid Evaluation of Potential Fields in Particle Systems," MIT Press, Cambridge, MA, 1988.
2. J. Barnes and P. Hut, *Nature* **324**, 446 (1986).
3. L. Greengard and V. Rokhlin, *J. Comput. Phys.* **73**, 325 (1987).
4. J. Carrier, L. Greengard, and V. Rokhlin, *SIAM J. Sci. Stat. Comput.* **9** (4), 669 (1988).
5. J.-P. Brunet and S. Lennart Johnsson, *Int. J. Super. Appl.* **6** (3), (1992).
6. S. Lennart and C.-T. Ho, *IEEE Trans. Comput.* **38** (9), 1249 (1989).
7. Thinking Machines Corp., "CMSSL for CMFortran", Cambridge, MA, 1900.
8. Thinking Machines Corp., ,"The Conection Machine CM-200 Series Technical Summary," Cambridge, MA, 1900.
9. L. L. Daemen, J. E. Gubernatis, L. J. Campbell, and L. Stiller, Computer studies of flux dynamics in superconducting thin films (unpublished).
10. L. Stiller, Los Alamos Technical Report LA-UR-92-2511 (unpublished).

Received September 16, 1992; revised June 23, 1993

L. Stiller

Department of Computer Science, The Johns Hopkins University, Baltimore, Maryland 21218-2686

L. L. Daemen
J. E. Gubernatis

Theoretical Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87545