JOURNAL OF COMPUTATIONAL PHYSICS 106, 108–114 (1993)

Efficient Parallel Implementation of Molecular Dynamics on a Toroidal Network. Part II. Multi-particle Potentials

K. ESSELINK AND P. A. J. HILBERS

Koninklijke/Shell-Laboratorium, Amsterdam, Shell Research B.V., Badhuisweg 3, 1031 CM Amsterdam, The Netherlands

Received October 17, 1991; revised July 6, 1992

Implementations for molecular dynamics on parallel computers generally use either particle parallelism or geometric parallelism. For short-range potentials, geometric parallelism has the advantage that communication can stay restricted to processors nearby. Usually, half the environment around a processor is communicated, using Newton's third law. This poses a problem for the implementation of multi-particle potentials (e.g., "bending" and "torsion"). For instance, if it is said that only one processor should actually calculate the forces on the particles involved, it will be difficult to determine which processor this should be, given that the particles are distributed over two or more processors. We present an efficient technique to do so and prove that it is correct. The technique requires no more communication than the computation of two-particle interactions and ensures that potentials are only evaluated once.

1. INTRODUCTION

The aim of molecular dynamics (MD) is to study the macroscopic behavior of systems of particles by simulation. Newton's equations are integrated over time for all particles. The forces on the particles are determined by the microscopic interaction potentials. Several potentials are widely used in MD codes.

Parallel computers are a very powerful tool for performing large-scale computations. Designing efficient parallel programs involves the distribution of even amounts of work over the various processors in such a way that the communication overhead is not too large. In general, this poses new problems and requires new techniques.

This article is the second of a series of two articles, in which we describe the design of a parallel implementation of molecular dynamics. The first article ([5], hereinafter I) discusses the distribution of work onto the processors of the network, and shows the column mapping to be efficient for networks of up to a certain size. Under the requirement that all potentials be evaluated only once, this mapping of space makes evaluation of multi-particle potentials somewhat complicated, since the particles involved may reside on different processors. In Section 2 we explain the problems encountered and introduce a technique to solve them. We prove this technique to be valid.

Results from timings are presented in Section 3 and Section 4 contains concluding remarks.

In this article, we restrict ourselves to short-ranged potentials and toroidal networks of processors (MIMD). In the following subsections we start by introducing a few well-known potentials and explaining the concept of geometric parallelism.

1.1. *Potentials*

To familiarize the reader with some potentials actually used in MD [1], we introduce them here. In the following, $\mathbf{r}_{ij}$ denotes the vector from particle i to particle j, and $r_{ij} \overset{\text{def}}{=} |\mathbf{r}_{ij}|$. The Coulomb potential is defined by

$$C_{ij} = \frac{c_i c_j}{4\pi\varepsilon_0 r_{ij}}. \tag{1}$$

It is classified as a long-range potential, for even at a considerable distance its influence is not negligible. In a simulation, this means that particles have an interaction with all other particles in the system. Since there are $O(N^2)$ pairs of particles (N being the number of particles), the total simulation time increases rapidly with the number of particles, unless refined methods are used to reduce this to $O(N \log N)$ [3] or $O(N)$ [2, 4, 7].

The Lennard–Jones potential has the following form:

$$LJ_{ij} = 4\varepsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right). \tag{2}$$

After some distance, it decays to zero with the sixth power. Usually, the potential is truncated, for instance at $R_c = 2.5\sigma$. This means that there is no Lennard–Jones interaction between particles at distances greater than R_c. For constant density, this means that the amount of interactions scales

0021-9991/93 $5.00

linearly with N. Well-known techniques [1, 13], also of $O(N)$, can be used to exploit this.

To simulate the behavior of chains of particles, potentials are introduced to couple specific particles. To model a chemical bond between two particles, a harmonic bonding potential can be used:

$$BO_{ij} = \tfrac{1}{2} B_c(r_{ij} - B_l)^2. \tag{3}$$

Bending forces in a chain of bonded particles are modeled by the three-particle potential

$$BE_{ijk} = F_{\mathrm{BE}}(\theta_{ijk}), \tag{4}$$

in which θ_{ijk} is the angle between $\mathbf{r}_{ji}$ and $\mathbf{r}_{jk}$ in the chain (i, j, k).

A four-particle torsion potential can be associated with an angle τ_{ijkl} between the planes ijk and jkl:

$$TO_{ijkl} = F_{\mathrm{TO}}(\tau_{ijkl}). \tag{5}$$

The Lennard–Jones, bonding, bending, and torsion potentials are short-range. Assuming constant density and a homogeneous distribution of particles, the amount of work done for each particle does not depend on the total number of particles in the universe. Implementations of order $O(N)$ are therefore feasible.

The potentials shown appear in many simulations. It is not the topic of this article to examine their specific properties. We are interested in potentials in their generic form, being a function of properties (e.g., position) of T particles. The only requirement is that the potentials be short-range, i.e., the distance between any two particles of the T-tuple is at most R_c.

1.2. *Parallel Computing*

Molecular dynamics is very suited for being done on parallel computers, as is explained in I. There we argued that geometric parallelism, the distribution of space rather than particles, can lead to small communication overhead and that it is possible to derive a mapping of space onto the processor network that yields minimal communication. During the computation, a processor calculates the trajectories of all particles it finds in its space. If all interaction potentials are short-range, it is not necessary to exchange information over long distances, because distance in the simulation universe is related to distance in the processor network. For two dimensions, Fig. 1 clarifies the procedure.

Recall that we use a network of toroidally connected processors. It is used "in two dimensions" and not in a line [9] to reach better scale-up properties. In the example, the x-size of a processor's cell equals $\frac{1}{2}R_c$, the y-size $\frac{1}{3}R_c$. This means that a processor needs information about particles in

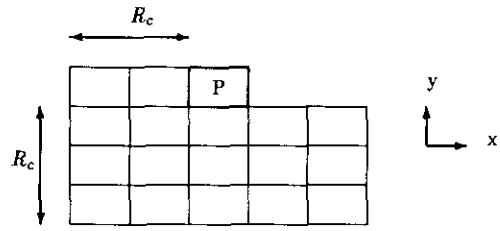

FIG. 1. Processor P and its environment.

processors at distance 2 in the x-direction and distance 3 in the y-direction. We call this distance "stretch," and speak of x-stretch and y-stretch. During the communication phase, each processor has to receive information from its communication environment and simultaneously send its own information to the other half of the surrounding processors. During the force evaluation phase, all Lennard–Jones interactions can be computed, storing the calculated forces for the own particles as well as the communicated particles (Newton's third law). Next, the forces for the neighboring particles need to be sent back. During the communication phases (particles and forces), all processors perform the same communication scheme, and it can be implemented very efficiently (see the Appendix).

In the force evaluation phase, a processor has to perform potential evaluation for all pairs of particles in which at least one particle belongs to itself. We note that the communication environment $\mathscr{C}'$ would also suffice for two-particle potentials (Fig. 2).

$\mathscr{C}'$ is certainly more efficient as far as the communication phase is concerned. The force evaluation phase also looks different: a processor needs to evaluate potentials of any pair of particles in which either at least one particle belongs to the processor itself or in which one particle has an x-coordinate within the processor's x-range and the other a y-coordinate within the processor's y-range. For example, in Fig. 2, processor P also has to evaluate the potential between particles a and b. However, the communication environment of Fig. 2 leads to extensive (and expensive) particle searches for multi-particle potentials. (This is explained in Subsection 2.4.) We have not been able to find any literature on implementations using $\mathscr{C}'$.

If we assume constant density and a homogeneous distribution of particles, the amount of work done by each processor for two-particle potentials is roughly of the order

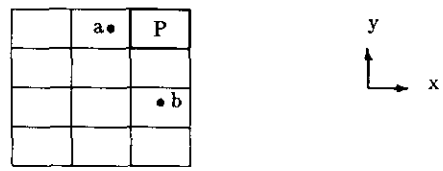

FIG. 2. Another communication environment $\mathscr{C}'$ for processor P.

of N_p, the amount of particles per processor. Note that an increase of the number of particles while maintaining constant density implies an increase of the simulation universe size. Therefore, the size of the sub-universe per processor increases, which has an effect on the stretch, and the cost, of the communication phase. Something similar holds for constant-pressure simulations, where universe size, and sometimes stretch, changes during the simulation. Note also that, by construction, the parallel algorithm is an implementation of the linked-cell method for particle tracking [1, 8].

To make explanation easier, in the following we use two dimensions.

2. MULTI-PARTICLE POTENTIALS IN 2D

2.1. *Definitions*

We now turn our attention to multi-particle potentials in a parallel algorithm based on the communication environment of Fig. 1. A discussion on this subject cannot reasonably be held in words alone; we therefore introduce a few definitions.

We have $P = P_x * P_y$ processors, fixed in a torus topology. A processor is identified by a coordinate pair (x, y). The simulation universe $\mathcal{U}$ has sizes $R_x * R_y$. It is divided into P equal parts. The domain assigned to processor p is called $\mathcal{U}_p$ and has size $B_x * B_y$, with $B_x = R_x/P_x$ and $B_y = R_y/P_y$. Potentials are to be calculated for T-tuples of particles, and they never extend over distances larger than R_c; i.e., the distance between any two particles of a potential tuple is smaller than or equal to R_c. In this article, $T \in \{2, 3, 4\}$. The x-stretch S_x equals $\lceil R_c/B_x \rceil$; likewise $S_y = \lceil R_c/B_y \rceil$. For processor p with coordinates (i, j) the communication environment $\mathcal{C}_p$ is defined as

$$\begin{aligned}\mathcal{C}_p = \bigcup (u, v: (((i - S_x \leqslant u \leqslant i + S_x) \\ \wedge (j - S_y \leqslant v \leqslant j - 1)) \\ \vee ((i - S_x \leqslant u \leqslant i - 1) \\ \wedge (v = j))) : \mathcal{U}_{(u,v)}). \end{aligned} \quad (6)$$

The work environment $\mathcal{W}_p$ for processor p is defined as the union of $\mathcal{C}_p$ and $\mathcal{U}_p$. Figure 1 provides an example of a processor's work environment for $S_x = 2$ and $S_y = 3$.

2.2. *Searching Potential Tuples*

Using particle parallelism, it is relatively easy to determine which processor should calculate the four-tuple potential for (a, b, c, d), since particles do not change processors. Where a, b, c, and d are can be determined initially, and one processor, preferably the one owning all four particles, should evaluate the potential. The same holds for a sequential implementation, where one can build a list of all tuples initially and use this during the simulation. For geometric parallelism, things are more complicated.

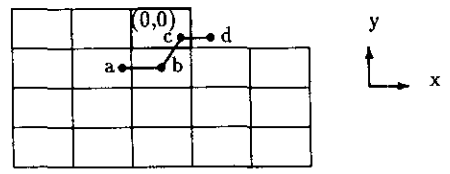

FIG. 3. The work environment for processor (0, 0).

The first problem is to determine which processor evaluates a given potential. In the example shown in Fig. 3 this cannot be done by processor (0, 0), since processor (0, 0) lacks information about particle d. This means that it is not possible to let the processor with the "first" particle of a tuple evaluate the potential. One solution would be to enlarge $\mathcal{C}_{(0,0)}$ so as to incorporate all particles around (0, 0) within R_c. This, however, would make the communication phase twice as expensive. It also suffers from the following drawback.

In Fig. 4 we see a tuple completely within $\mathcal{W}_{(0,0)}$, so processor (0, 0) has all the information to evaluate the potential. Here, there are seven processors p_i each having all four particles a, b, c, and d in their $\mathcal{W}_{p_i}$. It is, however, not very attractive to let processor (0, 0) evaluate the potential. An important reason for this is that $\mathcal{W}_p$ has to be scanned for possible tuples. This can be done by taking a particle q from $\mathcal{W}_p$ and trying to find other particles in $\mathcal{W}_p$ that belong to a potential tuple with q. If a complete tuple is found, the corresponding potential can be evaluated. If one or more particles are missing, it cannot be done. There is no way to determine beforehand whether all particles will be present; therefore this expensive search is necessary. Note that an extra criterion is necessary to prevent the potential being evaluated by more than one processor.

2.3. *Efficient Tuple Scan*

An efficient way to scan $\mathcal{W}_p$ would be to restrict the search to tuples containing at least one particle in $\mathcal{U}_p$ ("ETS": efficient tuple scan). It then suffices to take a particle q from $\mathcal{U}_p$ (instead of $\mathcal{W}_p$) and trying to find other particles in $\mathcal{W}_p$ which belong to a tuple with q. In general, an MD implementation will already contain a piece of code that does something like this, namely the evaluation of the pair-

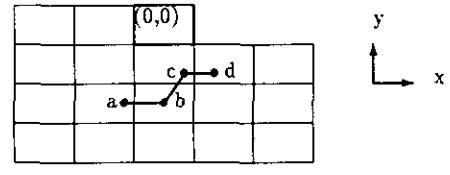

FIG. 4. Seven processors p_i with (a, b, c, d) in their $\mathcal{W}_{p_i}$.

potentials. This provides a perfect opportunity to maintain a list of possible candidates for multi-particle potentials. So, during the entire scan of $\mathscr{W}_p$ for particle $q \in \mathscr{U}_p$ in search of pair-potentials it is involved in, we build lists of tuples of multi-particle potentials it may also be involved in. After the scan of particle q, these tuples are checked to see if they are complete. If so, their corresponding potential can be evaluated. Figure 3 provides an example of an incomplete tuple, in which c will find a and b, but not d. Likewise, b will only find a, and a will find none. Only d will find a, b, and c.

We will now prove that if the processors use ETS, any tuple will be found by exactly one processor.

First we show that a tuple is not found more than once, by looking at the way two-particle potentials are searched. In this search, when particle i finds particle j and evaluates the pair potential, particle j will not find i. Hence, if particle i finds j, k, and l, with which it forms a potential tuple, none of these last three will find i.

Next, we prove that a tuple will be found at least once. This part of the proof is considerably more elaborate. We first describe a mesh containing all possible tuples. Recall that any two particles from a tuple have a distance of at most R_c. The size of a processor's cell is $B_x * B_y$, which means that any tuple t is contained in a submesh $\mathscr{M}$ of size $(S_x+1) * (S_y+1)$. See Fig. 5. So the minimum number of cells a tuple can be in is one, the maximum is $(S_x+1) * (S_y+1)$. By induction on a particular construction of $\mathscr{M}$, we can prove that the communication environment of Fig. 1 indeed guarantees that all tuples will be found by at least one processor, if all processors perform the search strategy ETS.

In the following we take $[x, y]$ to be shorthand for $\mathscr{U}_{(x,y)}$. The construction of $\mathscr{M}$ is straightforward. First take cell $[0, 0]$ and add cells in the x-direction (row 0) until the total number is S_x+1. Then add cell $[0, 1]$ and fill row 1, etc., until row S_y is filled. While constructing $\mathscr{M}$, we can build a communication environment $\mathscr{C}$. First, assume that a tuple lies completely in one cell $[0, 0]$. Then the necessary $\mathscr{C}$ for processor $(0, 0)$ can be empty. Next, assume that a tuple has one particle in $[1, 0]$ and the rest in $[0, 0]$ and $[1, 0]$. Processor $(1, 0)$ will then find this tuple if it has $[0, 0]$ in its $\mathscr{C}$. Repeating this process, if a tuple has one particle in $[S_x, 0]$ and the rest in $[0, 0] \cdots [S_x, 0]$, processor $(S_x, 0)$ will find this if it has $[0, 0] \cdots [S_x - 1, 0]$ in its $\mathscr{C}$. The next step in constructing $\mathscr{M}$ is to add $[0, 1]$. If a tuple has a particle in $[0, 1]$ and the rest of the tuple lies in $[0, 0] \cdots [S_x, 0]$ and $[0, 1]$, processor $(0, 1)$ will find this tuple if it has $[0, 0] \cdots [S_x, 0]$ in its $\mathscr{C}$. Filling row 1, if a tuple has a particle in $[S_x, 1]$ and the rest in $[0, 0] \cdots [S_x, 0]$ and $[0, 1] \cdots [S_x, 1]$, it will find this tuple if it has $[0, 0] \cdots [S_x, 0]$ and $[0, 1] \cdots [S_x, 1]$ in its $\mathscr{C}$. After building $\mathscr{M}$, $\mathscr{C}$ is found to be the one defined by Eq. (6).

This proves that if a mesh of processors performs ETS to search for potential tuples, each potential tuple will be found by exactly one processor.

2.4. *Different Communication Environments*

The way in which $\mathscr{M}$ is constructed determines the $\mathscr{C}$ that is derived. For instance, if $\mathscr{M}$ is built by randomly adding cells (instead of the row and column sequence mentioned in the previous subsection), the necessary communication environment for processor (i, j) is found to consist of almost $(2S_x+1) * (2S_y+1)$ cells. Also, there cannot be a smaller one than that derived above, apart from the cases in which the stretches are so large that two opposite corners have a distance exceeding R_c.

For pair potentials, it is not necessary for the communication environment to look like Fig. 1. The environment of Fig. 6 would also suffice. For multi-particle potentials, however, this communication environment does not permit the use of ETS. It is easy to construct a counterexample in this case for the statement that all tuples will be found (e.g., four particles in a row, maximum distance $\approx R_c$). Furthermore, the environment cannot be efficiently communicated with the technique described in the Appendix.

The smaller communication environment of Fig. 2 is also not fit for use with ETS. We have already seen that, for pair potentials, this environment needs not just a search for pairs with at least one particle in the processor's cell, but also an additional search for pairs with a particle in the same row and a particle in the same column as the processor's cell. For multi-particle potentials, something similar holds. Not only tuples with at least one particle in the processor's cell will have to be searched for, but also tuples with one particle in the same row and one particle in the same column. If these are found, an extra search for the remaining particles of the tuple in the communication environment is necessary.

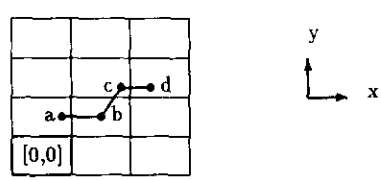

FIG. 5. The minimum submesh $\mathscr{M}$ containing every possible tuple.

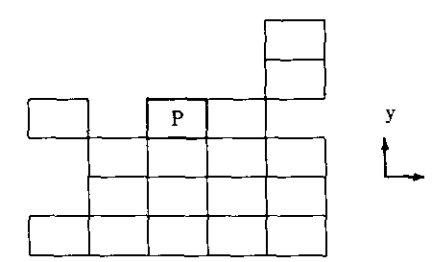

FIG. 6. An also-possible communication environment.

3. IMPLEMENTATION AND RESULTS

So far, we have assumed that there is one cell $\mathcal{U}_p$ per processor p. If the available processor network consists of few, but very powerful, processors (such as a network of workstations), this is not very efficient, for even a stretch of one would be too much. In such a case, one would like to have more cells per processor, in which each cell edge is roughly R_c in size, see Fig. 7.

As in the previous section, the environment of the processor consists of half the available cells. However, if good scale-up is to be guaranteed, the individual cells of a processor need their own communication environments, in order to implement the standard linked-list method. If the processor as a whole only communicates the environment of Fig. 7, the individual cells have different environments. Cell C_3, for instance, lacks two lower-right neighbors. This, however, can be made up for by letting C_0 and C_1 include the upper-left cells in their environments. The proof that ETS still finds all tuples is more elaborate in this case. It needs an $\mathcal{M}$ of size $(S_x + C_x) * (S_y + C_y)$, C_x being the number of cells in the x-direction on a processor. We will not present this proof here.

Another issue is the dimensions. Our implementation uses three dimensions. The third dimension is mapped such that a processor gets a column of cells. This mapping can be proven to be optimal with respect to communication (I). If $S_x = S_y = S_z = 1$ and $C_x = C_y = 1$, Fig. 8 shows the communication environment for one cell of the column.

Another property not mentioned so far is the use of (Verlet) particle lists to keep track of nearby particles for the Lennard–Jones potential. Also, lists are built to store pairs, triples, and quadruples, respectively, of particles for which the bonding, bending, and torsion potential has to be evaluated. The size of the cells is taken to be $R_c + R_s$, so that the built lists can be used several iterations before updating. Lists are built in Procedure **Verlet**.

We performed simulations on transputer networks of $1 * 1$ up to $20 * 20$ processors (T800). The results presented in this section are mainly intended to show that ETS is an efficient search technique, not so much that our implementation has good scale-up properties. Therefore, we restricted ourselves to three universes, all with linear chains. The first universe consisted of 70 chains of length 20, the second of

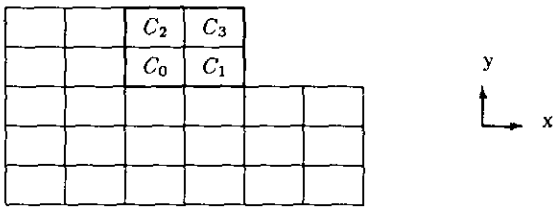

FIG. 7. Processor P with four cells and its environment.

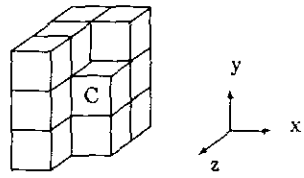

FIG. 8. One cell ("C") with 13 neighbors

150 of the same length, and the third of 200 chains of length 50. The parameters of our various potentials can be found in Ref. [11]. The density was set to $2.24\sigma^{-3}$. With this density, each particle has a Lennard–Jones interaction with ≈ 200 other particles; furthermore, it is involved in two bonding, three bending, and four torsion potentials (with the exception of the head and tail particles). Since the potential tuples are stored in lists, the cost per evaluation can be timed fairly precisely. It is found in Table I. (The timings include energy and virial calculation; they have not been optimized by using table lookup.)

Note that although the torsion is an expensive potential, it has a contribution for four particles and that it is evaluated by exactly one processor. The cost *per particle* is therefore roughly equal for all potentials. Also we see from this that the evaluation of the bending and torsion potential accounts for 5.8% of the calculation of the total force.

More important for this article is the cost of ETS. Therefore we turn attention towards procedure **Verlet**. It is important to realize that this procedure depends heavily on the subdivision of the universe into cells. For each of its particles q in all its cells, a processor scans the communication environment of the cell to look for other particles with which q has an interaction. This scanning is therefore a function of the number of cells E in the communication environment and the number of particles per cell. We have

$$E = 4S_xS_yS_z + 2(S_xS_y + S_xS_z + S_yS_z) + S_x + S_y + S_z. \tag{7}$$

The following equation yields the time (s) per particle that **Verlet** costs (potential evaluation and particle communication are excluded),

$$\mathcal{V} = C_0 N_c (E + 0.5) \tag{8}$$

TABLE I

Execution Times per Potential Evaluation

Potential	Time (μs)
Lennard-Jones	110
Bonding	125
Bending	306
Torsion	507

TABLE II

The Cost of ETS Compared to **Verlet**

No. particles	Bending/torsion	ETS/**Verlet**
1400	No	$4.84 \times 10^{-3} \pm 7.3 \times 10^{-4}$
1400	Yes	$9.70 \times 10^{-3} \pm 1.4 \times 10^{-3}$
3000	No	$3.38 \times 10^{-3} \pm 4.7 \times 10^{-4}$
3000	Yes	$6.88 \times 10^{-3} \pm 1.0 \times 10^{-3}$
10000	Yes	$4.8 \times 10^{-3} \pm 1.9 \times 10^{-3}$

in which N_c equals the number of particles per cell. Our simulations showed that

$$C_0 = 7.52 \times 10^{-5} \pm 2.5 \times 10^{-6}. \quad (9)$$

Each simulation was performed twice, once without and once with bending and torsion evaluation. Both types of simulation evaluated Lennard–Jones and bonding potentials. Within the accuracy, the value of C_0 *does not* depend on whether or not bending and torsion potentials are evaluated. This shows that **Verlet** is dominated vastly by the search for Lennard–Jones (and bonding) potentials.

Comparing the two types of simulation yields an opportunity to assess the time it takes (for ETS) to find potential tuples. We therefore timed the actual cost of the ETS part of **Verlet**. Table II shows the quotient of the execution times for ETS and **Verlet**. In the first two universes, the time it takes to find the bonding pairs is about equal to the time it takes to find the bending and torsion tuples. All three are negligibly small compared to the search time for Lennard–Jones potentials. From the simulations without bending and torsion we find that the time it takes to find a bonding pair equals $1.72 \times 10^{-4} \pm 4 \times 10^{-6}$, independent of N. We further find that the time it takes to find a bending or torsion tuple equals $9.35 \times 10^{-4} \pm 1.6 \times 10^{-5}$. This figure depends slightly on the structure of the chains.

In short, using ETS for finding bonding, bending, and torsion tuples requires less than 1% of the time needed to find the Lennard–Jones pairs for typical configurations.

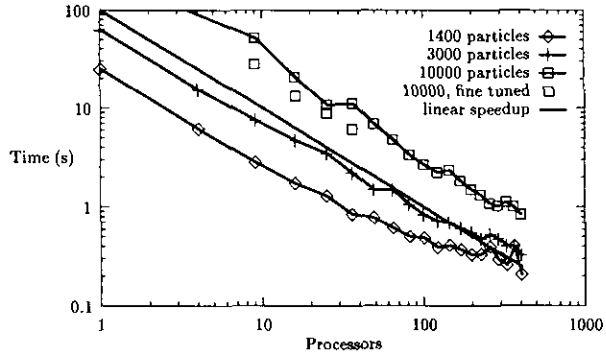

FIG. 9. Execution times for 1400, 3000, and 10,000 particles.

With the above equation it is possible to calculate the actual ratios for any system.

For completeness, the performance of the implementation is given for three universes in Fig. 9. The scales are log/log, so a slope of -1 indicates a linear speedup. Note that the "bumps" in the graph are caused by the (discrete) changes in the stretch. The graph of 10,000 particles also shows four additional timings in which the number of boxes was tuned to yield an optimum result. The effect can be quite dramatic. Furthermore, the results are obtained on the $20 * 20$ network, in which the "unused" processors for the timings of networks of a smaller size merely pipe the data to complete the necessary torus. Therefore, the times in Fig. 9 are somewhat too high. As is to be expected, the performance of the larger networks is not too good for small universes with, say, 1400 particles. The efficiency (speedup/processors) decreases to 50%. The scaled speedup, however, is quite good.

This implementation is also used for the simulation reported in [12]. For 32,000 particles, the time per iteration was reduced to 0.75 s, mainly due to lower density ($0.7\ \sigma^{-3}$).

4. CONCLUDING REMARKS

We have shown that it is possible to implement multiparticle potentials on a network of processors when geometric parallelism is used. The implementation requires no more communication than for two-particle potentials and evaluates a particular potential by exactly one processor. The search for the potential tuples can be embedded efficiently in the search for two-particle potentials. Three tests served as an example.

APPENDIX: EFFICIENT COMMUNICATION

With the environment shown in Fig. 1 it is possible to implement the communication of the particles very efficiently, as described in Refs. [6, 10]. There it is explained for a stretch equal to one only (see Fig. 10). In order to broadcast information from P to a, b, c, and d, this information is first sent to b and d, which can be done at communication cost 1 (the processors are adjacent). Sending it to a and c would normally cost 2, since the information has to be routed through (say) b. But since the information is already there, b might send this to a and c directly.

This principle can be generalized. Even unattractive communication environments, such as that depicted in Fig. 11,

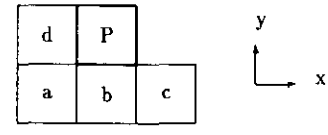

FIG. 10. Processor P and its environment (stretch = 1).

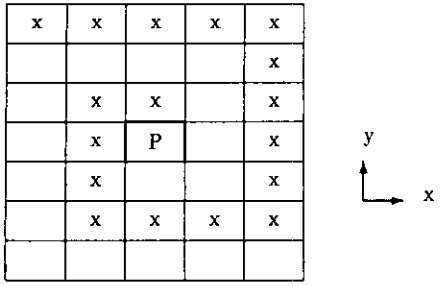

FIG. 11. An unattractive environment (marked "x") for processor P.

can benefit from this technique. The naive cost of broadcasting information from P to all 17 processors P_i marked x would be $\sum_{P_i} \text{dist}(P, P_i)$. But using the knowledge about where to find the information, the communication cost can be kept down to 17. In Fig. 6 the technique cannot yield the same cost.

This is an important possibility. It shows that there is no need for "more links" on each processor. With the environment in this article, the communication cost is of the order of the size of the environment. More links, to shorten communication paths, cannot improve this (so the torus connectivity is good enough). One can use this finding to prove that the plain "z-mapping" of the universe is the best one can do, given the state-of-the-art processor networks (I).

REFERENCES

1. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Oxford Sci., New York, 1987).
2. A. W. Appel, *SIAM J. Sci. Stat. Comput.* **6** (1), 85 (1985).
3. J. Barnes and P. Hut, *Nature* **324**, 446 (1986).
4. K. Esselink, *Inform. Process. Lett.* **41** (3), 141 (1992).
5. K. Esselink, B. Smit, and P. A. J. Hilbers, *J. Comput. Phys.* **105** (1) (1993).
6. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors, Vol. I* (Prentice-Hall, Englewood Cliffs, NJ, 1988).
7. L. Greengard and V. Rokhlin, *J. Comput. Phys.* **73**, 325 (1987).
8. R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles* (McGraw-Hill, New York, 1981).
9. H. G. Petersen and J. W. Perram, *Mol. Phys.* **67** (4), 849 (1989).
10. M. R. S. Pinches, D. J. Tildesley, and W. Smith, *Mol. Simul.* **6**, 51 (1991).
11. D. Rigby and R.-J. Roe, *J. Chem. Phys.* **87** (12), 7285 (1987).
12. B. Smit, P. A. J. Hilbers, K. Esselink, L. A. M. Rupert, N. M. van Os, and A. G. Schlijper, *J. Phys. Chem.* **95**, (1991).
13. L. Verlet, *Phys. Rev.* **159** (1), 98 (1967).