NOTE

Systolic Calculation of Pair Interactions Using the Cell Linked-Lists Method on Multi-processor Systems

Methods generally used in molecular dynamics (MD) simulations to speed up the calculation of forces by reducing the number of operations that have to be performed mostly fall into two classes: neighbour list [1] or cell linked-lists [2] algorithms. The relative merits of both methods have been discussed extensively in the literature (see, e.g., Ref. [3, 4]) and the discussion is not to be repeated here. Let us only recall the conclusion that the cell linked-lists method becomes competitive if the potential cutoff radius r_c used in the particle–particle interaction calculation is small compared to the system dimensions.

According to this method [5] the simulation box is geometrically divided into a regular lattice of cells. These cells are chosen so that the side of the cell l is greater than the cutoff distance r_c. For each cell it is established which particles it contains. Two arrays are created during the particle sorting process. The head-of-chain array HEAD which has one entry for each cell containing the identification number of one particle (if any) contained in that cell, and the linked-list array LIST (whose dimension is the number of particles) containing the rest of particles indices. By means of these arrays searching through the neighbours is a rapid process. With reference to the two-dimensional example of Fig. 1 the neighbours of any particle in cell 24 are to be found in cells 16, 17, 18, 23, 24, 25, 30, 31, 32 for each of which a different path exists from HEAD through LIST array which singles out all the particles contained in it.

The cell linked-lists method, as well as the neighbour list, is suited for massively parallel computers with distributed memory, but the exact way in which such parallelization can be achieved strongly depends on the parallelization scheme of the program in which those acceleration devices have to be used. The cell linked-lists method has already been implemented as part of a program which adopts a geometric parallelization scheme [6]. A far less obvious problem is how to adapt it to a systolic parallelization scheme [7, 8], where one cannot rely on any natural partitioning of cells as in the geometric case. The present note describes how to access the cell linked-lists method in a systolic parallel context.

We begin by giving a rapid outline of the systolic parallelization scheme, pointing out how the cell linked-list acceleration device has to be used during the force evaluation phases. For the sake of simplicity we will refer to two-dimensional examples, but what we are going to say holds true in both two and three dimensions. Consider a system of N molecules whose pairwise interactions have to be evaluated. Let P be the number of processors and n_i ($i = 1, ..., P$) be the number of molecules assigned to the ith processor, with $\sum_{i=1}^{P} n_i = N$. According to the typical systolic approach, [7, 8], P identical processes run concurrently on P processors connected by communication channels to form a ring so that each processor receives data from the neighbouring processor on one side and sends data to the neighbouring processor on the other side. It is convenient to think of data as forming P packets, each containing, besides the usual coordinates and force accumulators of the n_i particles assigned to each processor, the HEAD and LIST arrays. Packets move around the ring in a synchronized manner so that every processor simultaneously transmits a packet to its downstream neighbour and receives a new packet from its upstream neighbour.

The sequence of operations performed by every processor is the following (see Fig. 2): at each MD integration step each processor starts by filling the HEAD and LIST arrays with its home particles' indexes, performing the classical sorting. Then it makes a copy of the data packet relative to its particles. The original packet will remain fixed in the home processor while the copy will circulate systolically around the ring to return back to its home processor in P pulses. Initially, i.e., in the zeroth pulse, each processor evaluates the interactions between the n_i resident particles by looping over all cells [5]. Then, $P-1$ pulses follow, in each of which every processor evaluates interactions among home and circulating particles. However, now the generic processor i will not perform the loop over all cells. Indeed, during the first $P-i$ pulses it will loop over those cells whose index is even, while, in the remaining pulses, it will loop over odd cells only. To bring each circulating data packet back to its home processor a further pulse of pure communication is needed. In the home processor the fixed and circulating force accumulators are added to yield the total force acting on each particle. It is then possible to

0021-9991/93 $5.00

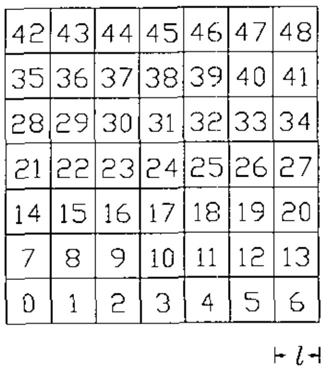

FIG. 1. The simulation box is divided into a regular lattice of cells. The side of the cell l is greater than the cutoff length r_c.

```
WHILE cycling
  SEQ
    ... sort 'home' particles into cells by filling HEAD and LIST arrays
    ... make the copy of data packet to be circulated systolically
    SEQ pulse = 0 FOR P
      IF
        pulse = 0
          PAR
            ... calculate interactions between 'home' particles
                by looping over all cells
            ... send outgoing data packet to downstream neighbour
            ... receive incoming data packet from upstream neighbour

        pulse > 0 AND pulse <= P - i
          SEQ
            PAR
              ... calculate interactions between 'home' particles
                  and particles received in the previous pulse
                  by looping over even cells only
              ... send outgoing data packet to downstream neighbour
              ... receive incoming data packet from upstream neighbour

            ... copy incoming data packet into outgoing data packet

        pulse > P - i
          SEQ
            PAR
              ... calculate interactions between 'home' particles
                  and particles received in the previous pulse
                  by looping over odd cells only
              ... send outgoing data packet to downstream neighbour
              ... receive incoming data packet from upstream neighbour

            ... copy incoming data packet into outgoing data packet

    ... last communication of data packets
    ... sum fixed and moving force accumulators
    ... integrate equations of motion to obtain new positions
        and velocities of 'home' particles
```

FIG. 2. Outline in pseudo-code (Occam-like [11]) of the sequence of operations performed by processor i at each iteration step. In Occam, processes are built out of more elementary processes, in a hierarchical manner, by the use of the constructors SEQ (sequential), PAR (parallel), WHILE (loop), IF (conditional), and others. Component processes, here represented by three dots followed by some text, are indicated by a two-space indentation relative to the constructor. Some constructors can be replicated; so e.g., `SEQ pulse = 0 FOR P` means that the component process is executed P times starting with the counter `pulse = 0`. Note the different usage of the cell data structure in various systolic pulses. For symbols explanation see text.

integrate the equations of motion according to some numerical method, to obtain the new positions and velocities.

To show how the method outlined above enables one to evaluate all pair interactions without any duplication, we will refer to Fig. 3. In this figure the cell situation is presented as seen by processor i in the generic systolic pulse p $(p \neq 0)$. White circles represent home molecules while black circles indicate molecules belonging to the circulating packet which, in that pulse, are visiting processor i. The highlighted cell is the one that processor is attending to. Continuous lines connecting particles of the same color in the same cell symbolize that such particles belong to the same linked-list formed by each processor with its home particles, before starting the interaction calculations. Dashed lines represent some of the interactions evaluated by processor i in the considered pulse. Note that dashed lines never join two particles of the same color since the evaluation of interactions between particles assigned to the same processor has been completed in the zeroth pulse.

Let us suppose that processor i is considering particle H. First it will compute the interactions of this particle with particles belonging to processor $q = \mathrm{MOD}(i + p, P)$ which in pulse p are its "guests" and that, being in the same cell, have their head-of-chain in the entry of the circulating HEAD array relative to the cell under examination (see Fig. 3A). It will do the same with all other particles in its own linked-list. Then processor i will consider interactions between particles in the current cell and those in neighbouring cells. This implies that it will take care, not only of the interactions between home particles in the current cell with all guest particles which are in neighbouring cells, but

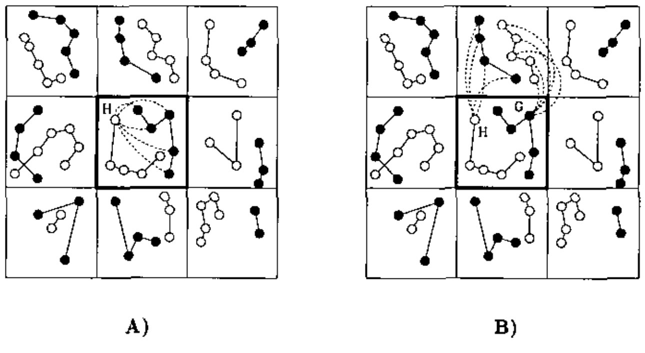

FIG. 3. (A) The processor which, in a certain systolic pulse, is considering the highlighted cell first will have to compute the interactions between its own particles (white circles) in that cell and "guest" particles (black circles) in the same cell. As an example, some of the interactions in which particle H is involved are represented by dashed lines. Continuous lines connect particles belonging to the same linked-list. (B) The same processor of (A), in the same systolic pulse, will also have to compute interactions of "guest" particles (e.g., particle G) in the highlighted cell with "home" particles in neighbouring cells, as well as interactions between "home" particles (e.g., particle H) in the highlighted cell and guest particles in neighbouring cells.

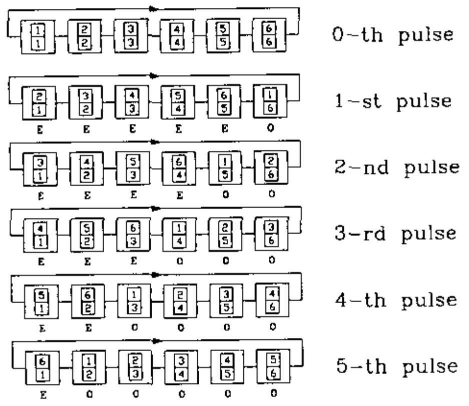

FIG. 4. Relative positions of fixed (lower) and circulating (upper) data packets inside each one of the six processors (big squares) of the systolic ring, in successive pulses. Capital letters denote the parity, even (E) or odd (O), of the cells a processor has to loop over in each pulse.

also of the interactions between guest particles which are in the current cell with all the home particles in neighbouring cells (see Fig. 3B). In doing so it will resort to the same trick, used on sequential machines to avoid interaction duplications, of considering only half of the neighbouring cells [5]. Thus, for example, a molecule in cell 24 (see Fig. 1) may interact with molecules in the eight neighbouring cells but processor i only checks chains in cells 31, 32, 25, and 18. Interactions of this molecule with molecules which are in cells of the same parity of cell 24, i.e., cells 16 and 30, are checked by the same processor in the same pulse, when cells 16 and 30 are, in their turn, the focus of attention. Instead, interactions with particles in cells of different parity, i.e., cells 17 and 23, cannot be checked by processor i as it is looping over even cells only. Nevertheless, those interactions will be (or they have already been) evaluated by processor q when, at a distance of $|P-|2(p-q)||$ pulses, it will take (or it has already taken) into account cells 17 and 23.

From Fig. 4 it can be seen directly how the proposed alternation in the parity of the cells to be considered leads to the correct interaction evaluation without any duplication. For example, the interactions between particles assigned to processor 3 and those assigned to processor 1 are evaluated partly by processor 1 in the second systolic pulse, looping over cells with even indexes only, and partly by processor 3 which, in the fourth pulse, will consider cells with odd indexes only.

A concurrent implementation of this algorithm has been carried out in Occam [11] for MD simulation of water molecules interacting via ST2 potential [12]. The physical system chosen as a test case for evaluating the performance of the algorithm consists of 4096 water molecules which are enclosed in a cubic box of 50.70476 Å side length. The standard periodic boundary conditions are imposed and a potential cutoff of 7.8 Å has been used in water–water interaction evaluation. An equilibrated configuration at a temperature of ca. 300° K has been chosen as initial conditions, and a time step of 5×10^{-14} s has been used for the numerical integration of the equations of motion.

To avoid unnecessary complications in the evaluation of the algorithm's performance, the simulated system has been chosen to be homogeneous and in a one-phase liquid region, so that the load balancing problem is optimally solved by the simple assignment of an equal (or nearly equal) number of particles to each processor at the beginning of the simulation. It is worth noting that more complicated situations, like simulations of systems undergoing phase transitions, can be efficiently tackled by easily integrating a newly designed dynamic load balancer [9, 10] with the present algorithm.

The multi-processor system on which our tests have been performed is made up of a variable number (up to 30) of 20-MHz T800 transputers (designated by W in Fig. 5) with 1-Mbyte external RAM each, connected as a ring, plus a 20-MHz T800 transputer (designated by I/O in Fig. 5) with 8-Mbyte RAM. The latter is interfaced to the host PC/386 and provides I/O capabilities to the transputer network.

In Table I results are reported concerning the execution time values per single MD step that have been obtained by using various numbers of processors. Values are also reported of the system efficiency $E=(T_1/P)/T_P$, where T_1 and T_P are the execution times for one-[1] and P-transputer systems, respectively.

The algorithm is coded so as to take full advantage of the ability of the transputer to simultaneously calculate and

TABLE I

Execution Times (in seconds) per Molecular Dynamics Time Step and Efficiency for a System Consisting of 4096 Water Molecules Interacting via ST2 Potential Simulated on Multi-Transputer System with Various Numbers of Processors

Number of processors	Execution time (s)	Efficiency
1	243.82	1
5	49.57	0.98
10	25.69	0.94
20	13.29	0.92
30	9.24	0.88

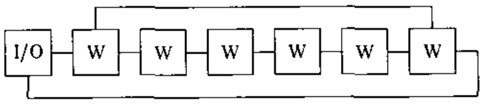

FIG. 5. Typical transputer network consisting of one I/O processor (I/O) and six worker processors (W).

[1] On one-transputer system, a purely sequential, communication-free program is executed.

communicate, as previously suggested [7] (see Fig. 2). By exploiting this feature, the time taken for data movements is no more additional to that taken for the force evaluation. Indeed, by overlapping computation and communication, data movement is almost completely free as long as packet transmission (at a rate of 10 Mbits s^{-1}) takes less time than the evaluation of forces during one pulse, a condition which is verified for every number of processors we have considered in our tests. This situation is to be expected due to the complexity of the potential used to model water molecules, and it should hold true up to ~ 10 particles per processor [7]).

So, the main source of the slight inefficiency has to be found in the increasing number (P, one per pulse) of loops over all cells that each processor has to perform as the processor number increases. As the total number of in-range interactions does not vary, an increasing amount of time is wasted in testing empty cells.

To further increase the number of processors without loss of efficiency, a mixed systolic and geometric parallelization approach should be adopted. Indeed, a gross geometric partition of physical space among groups of processors, which share particles belonging to their subspace in a systolic manner, could reduce the number of cells to be tested and maintain the high efficiency. Work is in progress to test the effectiveness of this solution.

ACKNOWLEDGMENTS

The author thanks Professor S. L. Fornili for helpful discussions. General indirect support from Italian MPI—60%, INFM, and CRRNSM funding is also acknowledged.

REFERENCES

1. L. Verlet, *Phys. Rev.* **159**, 98 (1967).
2. R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles* (McGraw–Hill, New York, 1981).
3. J. J. Morales, L. F. Rull, and S. Toxvaerd, *Comput. Phys. Commun.* **56**, 129 (1989).
4. W. F. van Gunsteren, H. J. C. Berendsen, F. Colonna, D. Perahia, J. P. Hollenberg, and D. Lellouch, *J. Comput. Chem.* **5**, 272 (1984).
5. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Oxford Univ. Press, Oxford, 1987).
6. F. Brugè and S. Fornili, *Comput. Phys. Commun.* **60**, 31 (1990).
7. A. R. C. Raine, D. Fincham, and W. Smith, *Comput. Phys. Commun.* **55**, 13 (1989).
8. F. Brugè and S. Fornili, *J. Comput. Phys.* **96**, 224 (1991).
9. F. Brugè and S. Fornili, *Comput. Phys. Commun.* **60**, 39 (1990).
10. J. E. Boillat, F. Brugè, and P. G. Kropf, *J. Comput. Phys.* **96**, 1 (1991).
11. INMOS, *OCCAM 2 Reference Manual* (Prentice–Hall, Englewood Cliffs, NJ, 1988).
12. F. H. Stillinger and A. Rahman, *J. Chem. Phys.* **60**, 1545 (1974).

Received October 19, 1990; revised October 11, 1991

F. BRUGÈ

Department of Physics,
University of Palermo and CNR-IAIF
Via Archirafi 36, I - 90123 Palermo, Italy