

Vectorized molecular dynamics algorithms

D.C. Rapaport,
Physics Department,
Bar-Ilan University,
Ramat-Gan,
Israel

March 8, 1989

1 On vector computation

The vector supercomputer [1] represents a compromise between the computer designer's desire to achieve maximal computation rate (within specified cost constraints) and the user's demand for the fastest possible computations over a broad range of problems. While this compromise has proved to have considerable benefit to both parties in a great many kinds of computation in science and engineering, there is no shortage of situations where the performance potential of the supercomputer is far from realized. What distinguishes algorithms that are effectively mapped onto a vector computer is the manner in which data is accessed and the nature of the processing carried out. Peak gains are achieved when the data is retrieved sequentially from storage, when only certain combinations of the basic arithmetic operations (preferably excluding division) are carried out, and when the results are returned sequentially to storage. Any deviation from a general operational pattern of this kind results in sub-optimal performance. However, with the exception of certain kinds of matrix and vector computation which manage to follow this prescription precisely, this state of perfection is unattainable. The issue then is how to achieve the best performance given the preferred manner of operation of the hardware.

In addition to the requirement that the data be sequentially ordered for vector processing, the processor design imposes a fixed startup overhead associated with each vectorized operation; this overhead is independent of the number of data items processed in the course of the operation. An immediate consequence of this overhead is that, if the vectors are too short, the somewhat paradoxical situation where vector processing is actually slower than the corresponding set of scalar operations (on the same computer) can be achieved; this is certainly something to be avoided. The minimal vector length requirements vary, depending on both the type of operation and the machine itself. But it is clear that in addition to rearranging the data to oblige the processor, it will also be necessary to ensure that the resulting data are organized into vectors that are of adequate length to guarantee that the fixed overheads do not nullify the expected performance gains. There is no promise that this can be achieved in all cases; there are indeed calculations for which effective vectorization is not possible.

To help the user tailor the computations for the vector processor the machine instruction set generally features the capability for reorganizing data at a fairly rapid rate,

one that tends to be intermediate between vector and scalar processing speeds. There are different approaches to data rearrangement and selection, and not all approaches are to be found on all machines. Even when a particular scheme for rearranging data is implemented, the question of how fast such operations are carried out in comparison with the peak (vector) computation speed is something which must be taken into account.

The two principal schemes for selecting and reordering data are known as gather-scatter and compress-expand. The act of gathering data involves the use of a vector of indices to access some or all of the elements of a set of data items – in no particular order as far as the computer is concerned – and store them sequentially in another vector; the scatter operation is the converse, in which the index vector is used to store a sequential set of data items in some alternative order in another (possibly longer) vector (where some elements might be left unmodified). Compression involves selecting an ordered subset of data items from a vector and storing them as adjacent elements in another vector; expansion is just the reverse of this operation. Because the order of the data is preserved under the compress/expand operations, information concerning which of the elements are to be compressed, or where the elements resulting from an expansion are to be placed, can be represented in terms of a bit vector (where the ones and zeroes represent, respectively, the elements that are or are not involved), an extremely compact alternative to the more general vector of indices utilized by gather/scatter. The fraction of the total elements needed in operations that require ordered subsets of elements can determine whether gathering or compression is the more effective operation – provided the computer offers a choice; not all machines do.

Even if the computer has the flexibility for dealing with data that are arranged in a less than optimal manner, there is the issue of whether the compiler that translates the high-level source language program into the actual machine instructions is capable of recognizing the kinds of operations needed. Judging by the achievements to date, compiler efficiency is an even more complex issue than hardware efficiency, and the performance of different compilers exhibits a wide variation in this respect. Even if the compiler is very perceptive and competent at mapping the program onto the hardware, there are situations in which certain language constructs (for example the case of an algebraic statement which implies a recursive dependence on something that has only just been computed, which might be unvectorizable because of the manner in which pipeline processing restricts dependencies between individual data items involved) may prevent the compiler from dealing effectively with that segment of the program; in such situations further information – expressed in the form of directives that lie outside the programming language – may help the compiler perform its task (this might include information indicating that even though a recursive dependency has been detected it can still be safely vectorized). The capacity for aiding the compiler in this way also varies from one brand of computer to another.

The alternative to total dependence on the capability of the compiler is to directly invoke the machine instructions. This can be done by programming in assembly language, but there may exist a preferable alternative which facilitates accessing machine instructions by subroutine calls from the higher-level language. The advantage of the latter approach is that it need only be resorted to when the compiler cannot deal with the problem, and it leaves the text of the program in more intelligible form; for most of the program the compiler should prove up to the task, and only certain critical sections may need to be handled in this way. This feature too is not necessarily provided on all

computers.

2 Reformulating molecular dynamics algorithms

The subject of molecular dynamics simulation – the techniques and the applications – have been described elsewhere [2, 3]. Here we deal with the question of reorganizing the molecular dynamics calculation into a form suitable for efficient vector processing. The most difficult instance of algorithm conversion is the one that deals with the case of a very large fluid system with only short-range interactions between particles. The discussion of this case will be deferred until last; the vectorization in other cases is much more readily carried out. We note that it is the details of the interaction computations on which the discussion is focused; the actual integration of the coupled equations of motion, as well as other relatively minor – from the point of view of computational effort – aspects of the simulations (such as establishing the initial conditions, or modifying particle coordinates to satisfy periodic boundary conditions) present no significant problems since these parts of the simulation constitute a relatively minor portion of the calculation that is readily (normally in a completely automatic manner) vectorized.

For relatively small systems, typically up to a few hundred particles, the quickest method for a vector processor is generally the simplest one, namely to consider all pairs of particles, even if they lie beyond the interaction range. The interaction computation can easily be altered to return a zero value if the cutoff separation is exceeded. If the pairing is carried out in a suitable manner that ensures that interacting pairs are organized as vectors associated with the particles concerned in an equitable manner [4] (since the interactions between pairs of particles can be regarded as a triangular matrix, the simplest scheme of processing a row or column at a time is not the most efficient way to approach the problem) the data to be processed can be organized into vectors of adequate length to avoid serious problems with the fixed startup overheads. The additional computations required to evaluate interactions between particles lying outside the interaction range are, in this case, not of sufficient quantity that the alternative techniques described below would yield any improvement in performance.

If the molecules of the fluid are complex, in the sense that there are several interactions sites at fixed relative positions within the molecule, then the amount of computation required normally precludes consideration of systems larger than about a thousand particles. In order to vectorize such a computation it is again worth considering the possibility of treating all pairs of molecules in the system (and subsequently discarding those pairs separated by more than the cutoff range – where the separation now refers to the centers of mass), but this time the calculation is broken into several sections (at the conceptual level, although the implementation itself may involve treating all these sections in a uniform manner as parts of a single 'outer' loop) each of which deals with a distinct pair of interaction sites in all the molecule pairs; for each pair of site types the innermost loop of the program evaluates the contributions from just those site pairs for all molecules of the system. This is the opposite of the usual approach in which all site pairs belonging to a particular pair of molecules are treated together; it also requires additional storage to hold interim data for the coordinates of the interaction sites on all of the molecules as well as the forces acting at each of the sites, but the rearrangement of the order of the computational loops results in an algorithm which vectorizes with no difficulty.

The computations associated with two other types of system are also readily vectorized. If the interaction is long-ranged, all pairs of particles interact and there is little opportunity for any shortcuts; the necessarily heavy computation will be carried out in fully vectorized fashion. Ewald sums [5] or particle-in-cell methods [6] may be applicable. Simulation of solids also presents an ideal situation for vector processing. The fact that the interacting pairs remain the same throughout the computation means that there is little problem in rearranging the computations in an appropriate way; this might be done by ordering the processing of interacting pairs according to the direction and range by which they are offset in the lattice.

This leaves the problem of a fluid with short-range interactions, where 'short' is defined relative to the length of the region occupied by the fluid. On a scalar computer there are two techniques in use, preferably both together, which reduce the size-dependence of the computation from quadratic to linear. The first is based on the introduction of a fictitious space-filling array of cells to which the particles are assigned. There is then no difficulty in determining which particle pairs interact using an amount of computation proportional to system size. If the linear dimensions of the cell are chosen so that they exceed the interaction range then only particle pairs that lie in the same or in neighboring cells are possible interaction candidates. The second is the construction of a list containing those neighboring pairs of particles that actually interact [7]. Since the earlier subdivision into cells overestimates the number of potentially interacting pairs that must still be examined to determine if they really do interact, there is clearly something to be gained from this approach, although there is a substantial penalty to be paid in terms of storage requirements for the extra information generated. Since the rearrangement of particles in the fluid is a gradual process it may not be necessary to update this list at every time step; the frequency of updates can be reduced if the list includes particle pairs whose separation exceeds the maximum range of interaction by a prescribed small amount – there is a tradeoff involved with the work needed for generating the neighbor list and the increased size of list if additional non-interacting pairs are included. By monitoring the maximum particle displacements at each time step it is possible to ensure that the neighbor list is recomputed as infrequently as possible, while at the same time guaranteeing that no interactions are missed.

This represents the most effective approach to dealing with this kind of problem on a scalar computer; the question is how to achieve a similar reduction in computational effort in a vector environment. Quite obviously one would not want to completely abandon the approach which leads to a linear rather than quadratic size dependence since there is no way the gains resulting from vectorization can compensate for this loss. But to achieve this dependence requires representing the information embodied in the cell subdivision, and possibly the neighbor list as well, in a form that can be handled in vectorized fashion. As will become apparent, the solution is basically a simple one, although the implementation can become somewhat awkward.

On a scalar computer, the information concerning which particles belong to which cells is stored in set of linked lists, one such list per cell. A linked list [8] is a way of structuring data that is not intended to be accessed sequentially; with each data item there is associated a pointer to the next data item (if any – otherwise the pointer is given a special 'null' value). Starting from the head of the list, which is merely a pointer associated with the cell itself, the contents of the cell can be determined by following the chain of pointers. The reason for using a linked list is the fact that no a priori limits

are imposed on cell occupancy, as would be the case if a fixed amount of storage were reserved for each cell; the result is a considerable saving in storage – often as important a factor in software implementation as the speed of the computation. Unfortunately the linked list concept is totally incompatible with the vector approach and so must be sacrificed in the interest of overall performance, even if the storage requirements increase as a consequence. The question is the choice of an effective alternative.

The technique actually used – described in greater detail in [9] – requires that the identity of the cell occupied by each particle be determined as a first step. Then the data reorganization is carried out by placing the particle serial numbers in one of a set of 'layers'. There are enough such layers to accommodate the maximum possible cell occupancy, and each layer contains a single storage element for each cell. The first particle in a particular cell is assigned to the first layer (the order of particles within cells is of course arbitrary), the second particle in the cell – if present – to the second layer, and so on. Assigning a particle to a layer requires knowing how many particles in that cell have already been assigned. A simple scheme which scans the complete set of particles just once is obviously not vectorizable, since the number of layers that must be examined in order to assign each particle varies, depending on cell occupancy. The alternative, vectorizable scheme, involves filling the layers one at a time, even though it implies scanning the entire set of particles several times. This technique is not as bad as it might seem however, because once a particle has been assigned to a layer it can be eliminated from the set of particles yet to be assigned. The processing involved in the construction of each such layer is fully vectorized, and all but the final layers corresponding to the tail of the cell-occupancy distribution provide sufficiently long vectors to yield effective vector-processing performance (assuming a large enough system).

Once a layer has been populated, the data it contains can be compacted if storage is at a premium. A particularly effective method for those machines which support the operations of compression and expansion is to pack the layer data in an ordered manner and use a bit vector to indicate which of the cells are actually occupied. When the data is later required by the interaction computations the layer contents can be re-expanded. A further extension of this idea, that plays an important role in the computations for extremely large systems (a hundred thousand particles or more), is that the region of space occupied by the system can be divided into a number of subregions, and only the particular portion of the layer associated with the part of the system actually being considered need be expanded. In the case of a subdivided system, additional effort is required to ensure that interacting particles located in distinct subregions of the system are dealt with properly, but the extra bookkeeping is not unmanageable and does not add significantly to the workload.

The interaction computations that follow the layer assignment involve the treatment of all pairs of occupied layers; for each pair of layers, the pairs of particles at locations in the layers corresponding to the cells for which interactions are possible are considered – but only if the cell locations in both layers are occupied by particles; a particular case that requires special attention is that of a layer paired with itself – each pair of particles appears twice and so only half the pairs should be treated. This calculation also vectorizes readily, although effort is wasted when sparsely occupied layers are paired; it is quite possible to have no interaction terms emerging from the processing of such layer pairs. Despite the wasted effort, this scheme is probably the most efficient on machines

that support the compress/expand operations at the hardware level.

An alternative approach is to use the layers to construct neighbor lists. The neighbor lists produced in this manner will have the property that in the list segment corresponding to a given layer no particle can appear more than once. Thus the interaction computations can again be vectorized, and the additional saving is that the neighbor list construction, together with the prerequisite layer assignment, need not be carried out at every time step. There is a heavy storage penalty associated with neighbor lists, especially if the interaction range allows many interacting neighbors per particle; it is unlikely that the approach would be useful for the extremely large systems that are now being studied – an example of the sacrifice in speed to economize on storage.

More of the bookkeeping mentioned earlier is needed to take periodic boundaries into account, as well as to deal with the interactions if the subdivision technique is employed. In particular, the conditional tests associated with periodic boundaries when computing interactions (the interparticle forces wrap around the 'edges' of the system, so that particles at opposite ends interact if both are sufficiently close to their respective boundaries) can be avoided by replicating the particles near the boundaries; these dummy particles are used only for computing the interactions and are then discarded.

Since layer assignment requires significantly less work than the interaction computations themselves, the assignment can be carried out afresh at each time step; thus there is no need, for example, to monitor particle displacements (as was done when using neighbor lists) to avoid the possibility that cell occupancies may no longer be correct. Finally, another shortcut that can be used if an efficient gather operation is available is to employ tabulated interactions and to perform a table lookup (possibly supplemented by interpolation) rather than evaluate the interactions from scratch; again the decision as to which technique is preferable depends on the machine.

3 Summary

It should be clear from the foregoing that there is no single technique that can be used for implementing all molecular dynamics simulations on all vector supercomputers. There are a broad variety of problem types, and an almost equally broad range of supercomputer architectures. By way of an exercise the reader should examine both the similarities and differences between well-known kinds of machine, typically the Cray X-MP and the ETA-10 (which in most respects is equivalent to the Cyber 205). A particular kind of computation which runs efficiently on one brand, or even model, of machine may not perform as well on another. The storage requirements can also differ significantly between machines, and there is often a need to determine an optimum balance between speed and memory needs. This is not the place to go into further details of the techniques which have appeared in print elsewhere (ref. [9], and to be published); suffice it to say that the methods have made possible production type simulations of 200,000 particle systems over a similar number of time steps, and are capable of dealing with even larger systems as suitable computers become available (tests on systems containing as many as 500,000 particles have been carried out). Such enormous (by usual standards) systems appear to be necessary for the study of classes of hydrodynamic instability at the microscopic level [10], and could well turn out to be necessary in other contexts as well.

Acknowledgements

This article is based on an invited talk given at the 1989 Workshop on Recent Developments in Computer Simulation in Condensed Matter Physics, organized by the Center for Simulational Physics at the University of Georgia (and will appear in expanded form in the proceedings to be published by Springer). Many of the developments described here were carried out during visits to the Center, and also during a visit to the Department of Physics at Mainz University; David Landau and Kurt Binder are thanked for their hospitality and support.

References

- [1] R.W. Hockney and C.R. Jesshope, *Parallel Computers*, 2nd edn., (Adam Hilger, Bristol, 1988).
- [2] *Molecular Dynamics Simulation of Statistical Mechanical Systems*, Proceedings of the International School of Physics “Enrico Fermi”, Course XCVII, Varenna, 1985, eds. G. Ciccotti and W.G. Hoover (North-Holland, Amsterdam, 1986).
- [3] M.P. Allen and D.J. Tildesley, *Computer Simulation of Liquids* (Oxford University Press, Oxford, 1987).
- [4] S. Brode and R. Alrichs, *Computer Phys. Comm.* **42**, 51 (1986); see also D. Brown, *CCP5 Newsletter* no. 24, p. 39 (1987).
- [5] R.W. Hockney and J.W. Eastwood, *Simulation Using Particles* (McGraw-Hill, New York, 1981).
- [6] J.-P. Hansen, in [2], p. 89.
- [7] L. Verlet, *Phys. Rev.* **159**, 98 (1967).
- [8] D.E. Knuth, *Fundamental Algorithms* (Addison-Wesley, Reading, 1968).
- [9] D.C. Rapaport, *Computer Phys. Repts.* **9**, 1 (1988).
- [10] D.C. Rapaport, *Phys. Rev. A* **36**, 3288 (1987), and to be published.