



ELSEVIER

Computer Physics Communications 120 (1999) 20–32

Computer Physics
Communications

www.elsevier.nl/locate/cpc

Tabulated potentials in molecular dynamics simulations

D. Wolff^{a,1}, W.G. Rudd^{b,2}

^a Department of Physics, Oregon State University, Corvallis, OR 97331, USA

^b Department of Computer Science, Oregon State University, Corvallis, OR 97331, USA

Received 1 March 1999

Abstract

The use of tables to determine inter-particle potentials and forces can speed up determinations of these functions in classical molecular dynamics simulations by as much as a factor of four for typical potential functions. Doing so results in a loss of accuracy in the resulting trajectories and requires increased use of main memory to store the tables. An analysis of these factors indicates when it is reasonable to use tables for various forms of the interaction potentials. © 1999 Elsevier Science B.V. All rights reserved.

1. Introduction

Molecular dynamics (MD) [1–3] is a proven means for simulating systems of atoms, molecules, and other particles. The method involves integrating the equations of motion derived from inter-particle potential functions. Statistical averages then yield mechanical, thermodynamic, and transport properties of the simulated systems.

In MD, computing the inter-particle potential and the forces between the particles is by far the most time-consuming aspect of the method for all but the simplest of potentials. There are informal reports that looking up potentials and forces in tables stored in memory can yield significant improvements in performance in MD calculations [4], but we know of no published work that focuses on the idea itself. In this report, we describe our study of the effects of replacing computations of the potentials and forces with table lookups

for these quantities³.

We focus on potentials that include terms that can be computed to any desired accuracy as part of the initiation before actual simulations begin, with emphasis on semi-empirical functions of relative positions of atoms, such as the Lennard-Jones (LJ) potential. The potentials we consider are thus response surfaces on which the simulations are based. These can be indexed by functions of atomic positions. The most common use of this in practice has been in tabulating the “Embedded Atom Method” or EAM potential. There are MD codes in use that use tabulated EAM potentials exclusively [5,6].

This is by no means the only way in which tables are used in MD calculations. For example, implementations of the MD-MC/CEM technique [7,8] use tables to store pre-computed integrals indexed by local jellium densities and to hold coefficient matrices used in numerical differentiations of the potentials. Tabu-

¹ E-mail: wolff@physics.orst.edu

² E-mail: rudd@cs.orst.edu

³ Henceforth, we will use “potential” to represent potential energy, and we will use “atoms” to represent the atoms, molecules, or other particles that comprise the systems being simulated.

lated potentials have also been used to accelerate the calculation of long range electrostatic forces in the Ewald sum [9].

Memory cycle times on common workstations are about 60 ns, which is approximately the time required for a single double-precision (DP) floating-point calculation. We therefore would expect that looking up values for potentials would be faster than computing them for all but the most trivial forms of the potential.

However, this is not always the case. Processors used in serious MD calculations include internal pipelined processors for floating-point arithmetic. Many offer vector processors as well. Both these processor enhancements require a steady flow of operands from memory. Efficient caching hardware and algorithms and optimizing compilers are tuned to prevent delays due to interruptions in the movement of data from memory. These techniques work best when the data access patterns are localized in memory and occur in a predictable order. On the other hand, using tabulated potentials is equivalent to performing random accesses into large arrays, so that accesses are neither localized nor predictable. We therefore expect and observe a trade-off between using tables to save arithmetic calculations and degradation in raw processor performance. Nevertheless, we expect to benefit from tabulating potentials for those that require relatively intense arithmetic computation and relatively little associated logic. This study supports that expectation.

Even when tabulating potentials offers better performance than computing them, there are still two other trade-offs to consider in deciding which method to use. The use of tables guarantees a loss of accuracy in computing the potentials. We use DP arithmetic in MD simulations to eliminate possible inaccuracies that could cloud the interpretation of results. It is not clear that we need that level of accuracy in many simulations. Often we do not know values of parameters in potential functions with any degree of certainty; in much of the work in materials science and other application areas we use semi-empirical potentials that involve constants that are known to only one or two decimal places in accuracy. These values frequently appear in power-law or exponential terms in potential functions. Therefore, inaccuracies in determining values of potentials under these circumstances are swamped by lack of precision in our knowledge of

the exact potential. Note that this comment does not apply to numerical techniques used to integrate the equations of motion; these techniques must be chosen carefully to ensure that they are sufficiently accurate. Otherwise, numerical instabilities can arise that lead to erroneous or divergent solutions.

The third trade-off between using tables and computing potentials and forces is the space-time trade-off, which is also linked with the accuracy issue. To make the computations faster, we must use memory to store tables that would otherwise be available for other purposes. This factor becomes particularly significant when we consider simulating systems with more than one species of atom. For two body potentials, the number of tables increases with the number of possible pair interactions,

$$N_{\text{tables}} = \binom{n}{2} + n, \quad (1)$$

where n is the number of atomic species. When the calculations are to be done on parallel or distributed processors, we must keep copies of the tables on each processor to avoid extra inter-processor communications. Memory is an important factor in limiting the sizes of simulation we can do. As main memory sizes continue to grow, the relative overhead from the tables becomes smaller. Our results described below indicate that under normal circumstances the use of memory for tables is not an important consideration.

In this report, we explore all of these issues. We begin with a discussion of how one might construct and access tables efficiently. We then present our results on the analysis of errors resulting from the use of tables and their effects on the results of simulations. Following a comparison of performance between the two methods for several different kinds of potentials, we present our conclusions and recommendations.

2. Implementing tables

Well-designed molecular dynamics codes [2,5,6,10–12] include single modules that compute potentials. To change a code to obtain potentials and forces from tables, we simply replace the code that computes these values with code that retrieves them from data structures in which they have been stored before the simulations are started. In order for this to

be worthwhile, an efficient method of indexing the table must first be found.

2.1. Hash functions

The objective is to determine the contributions to the force and potential acting on particle i by all the other particles j . We assume these forces depend only on separations r_{ij} between the particles. The potential/force routines in MD systems usually include checks to see if particles j are within a certain distance from particle i , as for example, when the potential function includes a short-range cutoff. These comparisons are done using r_{ij}^2 to avoid computing a square root. Therefore, it makes sense to use r_{ij}^2 as the measure of the distance used to determine which elements of the tabulated forces and potentials to use.

Since pointers and indexes for arrays are integers, we need to construct mappings between double precision measures of (squared) distances and pointers or indexes. We have investigated two kinds of mappings, casts of scaled values of the distance measure onto integers, and logical extraction of indexes from the DP distance measures.

Consider the pseudo-code below for the computation of the potential and force in a MD simulation:

```

for each  $r_{ij}^2$  do
   $e := \text{computeEnergy}(r_{ij}^2)$ 
   $f := \text{computeForce}(r_{ij}^2)$ 
   $\text{totalE} := \text{totalE} + e$ 
   $\mathbf{F} := \mathbf{F} + f\hat{\mathbf{r}}_{ij}$ 

```

Note that this is for a two-body potential. For a three-body potential the loop would be over triplets (r_{ij}^2, r_{ik}^2 , and θ_{ijk}). The computeEnergy and computeForce routines in practice are not separate routines, but since some of the computation is redundant, they are often just one function or are inlined directly into the loop.

The table version of the above code might look like the following:

```

for each  $r_{ij}^2$  do
   $e := \text{energyTable}[\text{hash}(r_{ij}^2)]$ 
   $f := \text{forceTable}[\text{hash}(r_{ij}^2)]$ 
   $\text{totalE} := \text{totalE} + e$ 
   $\mathbf{F} := \mathbf{F} + f\hat{\mathbf{r}}_{ij}$ 

```

Here, hash() is a function which maps pairwise dis-

tances to indexes in the tables. An example is a function that simply casts DP values to integers:

```

function hash(double t)
  return (int) ( (t) * 10000 )

```

This yields the integer part of the argument after shifting the decimal point four places to the right.

We could simulate a rounding operation with the following hash function:

```

function hash(double t)
  return (int) ( (t * 10000) + 0.5 )

```

The above two mappings are simple, but require converting a DP number to an integer, a process that is not inexpensive.

2.2. Double precision extraction

The other method we consider for determining indexes into potential arrays is the extraction of an index directly from a DP representation of some measure of distance between atoms. The advantage of this technique is that it eliminates the arithmetic, logic, and implicit function calls involved in converting DP values to integers.

For a given double precision exponent, we can use the mantissa, truncated at some precision, as an integer to access the table. The table size must be of size at least 2^{N_m} where N_m is the number of bits taken from the left-hand side of the mantissa.

However, in most cases we are interested in a range of numbers which include several different possible double precision exponents. In this case, we can use a few of the low order bits in the exponent as the first few high order bits in the integer index (see Fig. 1). Of course, care must be taken to make sure that enough bits are taken from the exponent to ensure that they are unique for all possible exponents that would be encountered in a simulation.

We will assume that a double precision number is 64 bits long. A function for extraction of the IN-DEX.BITS of Fig. 1 might be the following (the code is written using C syntax, and line numbers are inserted for future reference):

```

int extract(double t) {
  int *int_p;
  int index;

```

```

(1) int_p = (int *) &t;
(2) index = ( *int_p & MASK );
(3) index = index >> ( 20 - MANTISSA_BITS );
    return( index );
}

```

We have shown `extract()` as a function and, for clarity, we have written the computations in several steps. In practice, it should be in-lined within the code in order to avoid the overhead of the function call.

In the `extract()` function, `MANTISSA_BITS` is the number of bits of the mantissa which we would like to use. In other words, we are truncating the number at a precision of $2^{-\text{MANTISSA_BITS}} \times 2^x$ where x is the value of the exponent. The total length of an index to the table is `INDEX_BITS`, which is equal to `MANTISSA_BITS` plus the number of bits taken from the exponent (`EXP_BITS` in Fig. 1). The value of `INDEX_BITS` determines the sizes of the tables, which should be of length at least $2^{\text{INDEX_BITS}}$.

It is important to note that the table size is determined solely by the length of the integer used to access it. However, all of the table will not necessarily be used. The entire table will only be used when given a set of possible double precision inputs, `INDEX_BITS` includes all possible binary strings. Clearly, this depends on a number of factors including whether or not all possible binary strings are expected in the `EXP_BITS`, and whether or not the full range of mantissas for a given exponent is expected.

Here are what the numbered steps in `extract()` do:

- (1) We establish a pointer to an object of type `int` and set it to point to the `DP` argument. This allows us to ignore the type rules that ordinarily would restrict us to using `DP` operations on the argument. Since an integer is typically 32 bits and a double is usually 64, we are restricted to the leftmost 32 bits of the double. With an exponent of 11 bits, there are 20 bits of the mantissa to work with. One could use two integer pointers to point to the first and second half of the double in order to use more bits of the mantissa. However, this would require some extra instructions to combine the two.
- (2) `MASK` is used in an *and* operation (`&`) to remove the left-hand bits from the argument. `MASK` has the value

$$2^{\text{INDEX_BITS}+20-\text{MANTISSA_BITS}} - 1.$$

- (3) Shift the resulting integer to the right to produce an index of the required length.

As an example case, consider indexing a table when our pairwise distances r_{ij}^2 range from 0.5 to 127.9 Å. This would be reasonable for a short range pair potential with a cutoff of about 11.3 Å. The exponents of the floating point representations range from -1 to 6. In bias 127 or bias 1023 notation these exponents are unique in their three lowest order bits. Not only are they unique, but for this range of values they complete all possible three bit strings. Hence, if we create a table using three as the value for `EXP_BITS` and say 17 for `MANTISSA_BITS`, we can create a table of size 2^{20} with nearly every entry filled. We would also be accurate in the value of r_{ij}^2 to a precision of about 2^{-17+6} in the worst case.

The key here is to make the mapping function as easy to compute as possible. While it appears that the `extract()` function is more complex than the `hash()` function, the fact is that the integer cast is an expensive operation. We have found the `hash()` function requires about 5 times as much time as the `extract()` function.

2.3. Multiple tables

MD codes often deal with more than one species of atom, making it necessary to use more than one table. This can be done quite readily for example by making an array of tables. The first two dimensions of the array might be indexed by a code for the species of the atom, and the third dimension would be the table itself. We would arrange the pointers in the top two levels of the array so that an interaction between atoms would access the same table, independent of the order in which the species are specified. An access to the table might be the following:

```
e = tables[type1][type2][hash( $r_{ij}^2$ )];
```

where `type1` and `type2` refer to the atomic species.

2.4. Interpolation

Finally, one could interpolate between successive points in the table for more accurate results for the forces and potentials. A linear interpolation is most often all that is required, although more complicated interpolations have been used [13]. The same effect

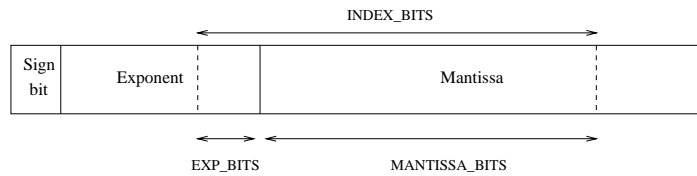


Fig. 1. Representation of double precision index extraction. INDEX_BITS are the bits that are actually extracted to index the table.

can often be achieved at little extra cost by increasing the size of the tables.

2.5. Constructing tables

Construction of the table is done before the simulation starts. We have found that for realistic problems, computing the tables takes a negligible amount of time and so we compute them with each run. It would make sense to store tables that are expensive to compute on disk.

Here is sample code to compute a table.

```

r := r_min
while r ≤ r_max do
  rsqr := r * r
  index := hash(rsqr)
  energyTable[index] := computeEnergy(rsqr)
  forceTable[index] := computeForce(rsqr)
  r := r + Δr

```

The routine above builds a table for $r_{\min} \leq r \leq r_{\max}$ where r_{\max} is the cutoff distance for the potential, and r_{\min} is some minimum distance where it is known that two atoms will never be less than r_{\min} apart. This, of course, depends on the details of the simulation.

When using the hash functions supplied above, some of this table would be left unused because the r^2 distances that map to the lower indices to the table $r < r_{\min}$ will never be needed. This is also often (although not necessarily) the case with the DP extraction routine. Depending on the size of the table that is needed the wasted space may not be a large concern. One could modify the hash function so that no space is wasted, but that would be at the cost of additional arithmetic when accessing the table.

It is possible to use hash functions along with chaining methods for collision resolution to help to eliminate the wasted space. We experimented with this approach and determined that the extra coding and computational effort required probably does not justify the

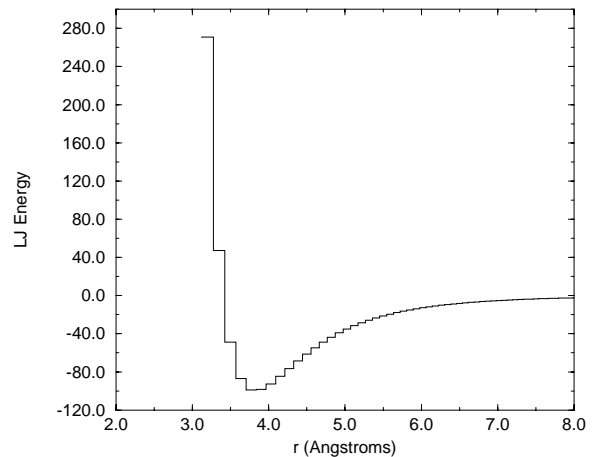


Fig. 2. The Lennard-Jones table-based potential. The spacing of the table values is much larger than would normally be used in practice.

small savings in memory.

3. Errors and error accumulation

The use of tables without interpolation changes the potentials and force functions to sequences of step functions (see Fig. 2). Again, at the cost of some arithmetic, we can improve the accuracy of tabulated potentials via interpolation. In our codes, we store both the potential and the force functions so that we need not differentiate the potential function numerically to obtain forces.

3.1. Estimation of error in the potential

In order to determine a more quantitative picture of the errors introduced by the use of a tabulated force field, we consider a general example. Most of the common potentials in use today have terms that depend either on an inverse power of the distance ($1/r^p$) or ex-

ponentially on distance (e^{-r}). We will consider both cases.

First we express the relative error for an inverse power term in a single table look-up as

$$\begin{aligned} \varepsilon_\phi &= \left| \frac{\phi(r) - \phi^{\text{table}}(r')}{\phi(r)} \right| = \left| \frac{1/r^p - 1/r'^p}{1/r^p} \right| \\ &= \left| 1 - \frac{r^p}{r'^p} \right|, \end{aligned} \quad (2)$$

where r is the actual position of the particle and r' is the truncated or rounded value used for a table lookup.

We can access the table by either rounding the value to the nearest table entry, or by simply truncating (rounding down). In the case of the rounding method of accessing the table, because of the increasing, negative slope of $1/r^p$, the maximum possible error occurs when $r < r'$ and $r' - r = \Delta r/2$ where Δr is the table spacing (see Fig. 3). Hence

$$\varepsilon_\phi \leq \left| 1 - \left(\frac{r' - \Delta r/2}{r'} \right)^p \right|. \quad (3)$$

Since p is positive and r' is always greater than $\Delta r/2$, we can write the upper bound as

$$\varepsilon_\phi \leq 1 - \left(1 - \frac{\Delta r}{2r'} \right)^p. \quad (4)$$

This gives an upper bound for the error in $1/r^p$ type potential. The maximum value that ε_ϕ can take on depends on the smallest possible value for r' for a given Δr . To determine this we must look to the particular substance being simulated. The pair distribution function $g(r)$ for a substance gives us information on the environment of a given atom. Since most atoms act like hard spheres for small values of r , the pair distribution function must go to zero at some small value of r , let us call it r_{\min} . Hence, we can expect that no atom will ever be within r_{\min} of another. This gives us a lower bound on r' in Eq. (4). Hence we can write the maximum possible error in the potential as

$$\varepsilon_\phi^{\max} = 1 - \left(1 - \frac{\Delta r}{2r_{\min}} \right)^p. \quad (5)$$

For our argon case study, the pair distribution function goes to zero at about 3.0 Å. So for a Δr of 0.001 Å and $p = 6$, $\varepsilon_\phi^{\max} = 0.001$. This is respectable when compared with the accuracy to which many potential

parameters are known. Also, it should be noted here that this may also be on the order of the accuracy given by the integration method used. Most integration methods conserve total energy to within the order of the timestep size. And perhaps more importantly, most table lookups will have substantially smaller error. In addition, for the rounding method, we can err on either side of the actual value for the potential (Fig. 3). That is, the tabulated value might be larger or smaller than the actual value. Hence, we can expect that some of the overall error will be canceled out by this effect.

A similar analysis can be done for an exponential potential,

$$\phi(r) = e^{-r}, \quad (6)$$

which yields

$$\varepsilon_\phi \leq \left| 1 - e^{-\Delta r/2} \right|. \quad (7)$$

3.2. Accumulation of errors

For a pair potential, the total potential energy for a single timestep is simply a sum of pairwise potential values,

$$\Phi = \sum_{i < j} \phi(r_{ij}), \quad (8)$$

where

$$r_{ij} = |\mathbf{r}_i - \mathbf{r}_j| \quad (9)$$

is the distance between atom i and atom j . For the case of the exponential potential, the relative particle-particle error is independent of the pairwise distance. Eq. (5) gives an expression for the $1/r^p$ potential that is also independent of the particle-particle distance.

Now, since the maximum relative error of each term in the sum is independent of the pairwise distance, the relative error in the sum is the same as the relative error of each term. Hence, Eqs. (5) and (7) also hold for the total potential,

$$\varepsilon_\Phi = \varepsilon_\phi. \quad (10)$$

Next, we consider the total force on a given atom,

$$\mathbf{F}_i = \sum_{j \neq i} f(r_{ij}) \hat{\mathbf{r}}, \quad (11)$$

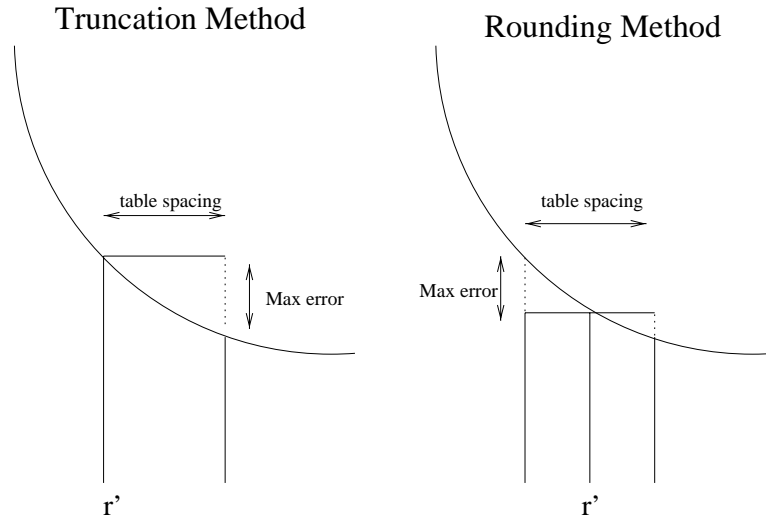


Fig. 3. Representation of a single table entry. Note that the error is largest when $r = r' + \Delta r$ for truncation, and when $r = r' - \Delta r/2$ for the rounding method.

where

$$f(r) = \frac{d}{dr}\phi(r). \quad (12)$$

It can be shown that the relative error in $f(r)$ for the $1/r^p$ potential type is the same as Eq. (4) except with an exponent of $p + 1$. Similarly, it is easy to show that the relative error in $f(r)$ for the exponential type potential is the same as Eq. (7).

Now, the force is a vector quantity, so the summation is a little different here as opposed to the total potential energy. However, one can still put an upper bound on the total error by assuming that all of the errors point in the same direction. Again the same argument applies as above. The relative error in the force for a single particle is

$$\varepsilon_{|F_i|} \leq 1 - \left(1 - \frac{\Delta r}{2r_{\min}}\right)^{p+1}, \quad (13)$$

for the $1/r^p$ potential. For the exponential potential a similar analysis reveals the same upper bound as Eq. (7).

3.3. Errors in measured values

Now that we have an upper bound on the relative error for the force and potential for a given timestep, let us consider how those errors propagate into some

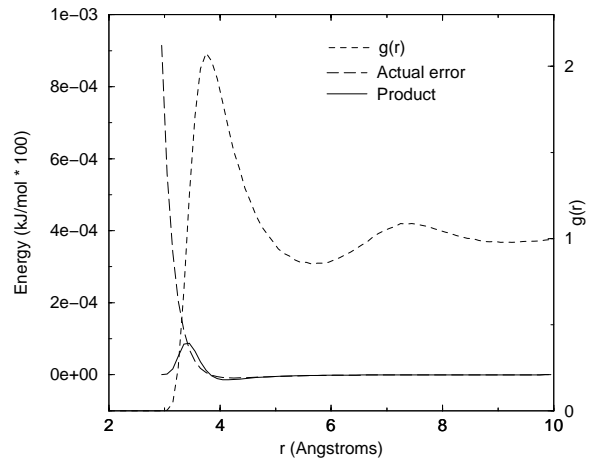


Fig. 4. The scaled, actual error $(\phi(r) - \phi(r + \Delta r))/\epsilon$ and pair distribution function taken from an MD simulation of LJ liquid argon, where ϵ is the depth of the potential well. The solid line is the product of the error and the pair distribution function.

of the quantities that are measured. The pressure, temperature, stress, and total energy all depend on various summations involving the force, positions, and velocities of the particles. We have already discovered how the errors in individual table lookups propagate to the total force and potential energy.

The total energy is given by

$$E = \sum_{i < j} \phi(r_{ij}) + \frac{1}{2} \sum_j m_j v_j^2. \quad (14)$$

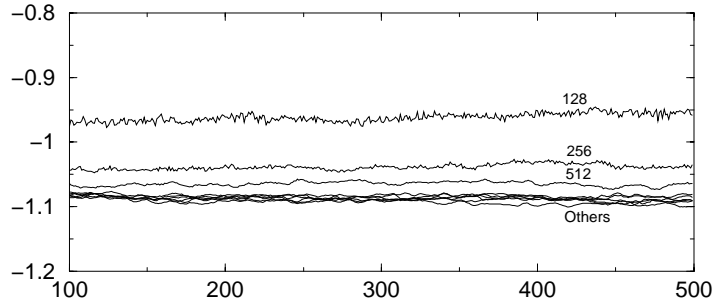


Fig. 5. Total energy per atom versus time for several sizes of potential tables.

We have already put an upper bound on the error in the first term, it remains to put an upper bound on the error in the second.

The velocity is obtained from the total acceleration by integrating over some short time interval. There are many methods of doing this [1], but for simplicity and in order to consider upper bounds, we will consider just a Taylor expansion of the velocity,

$$\mathbf{v}(t + \delta t) = \mathbf{v}(t) + \delta t \mathbf{a}(t) + \dots \quad (15)$$

For a given timestep there is some error in the force (acceleration) due to the use of a tabulated force field. This is equivalent to adding a term to the acceleration,

$$\mathbf{v}(t + \delta t) = \mathbf{v}(t) + \delta t (\mathbf{a}(t) + \varepsilon_a \mathbf{a}(t)), \quad (16)$$

where ε_a is the relative error bound in the acceleration which is equal to the relative error bound in the force. Now, this term is of order $\delta t \varepsilon_a a$ which is similar order as the error in the potential energy term. In the expression for the total energy, this term is squared, which makes the error of order $\delta t \varepsilon_a a \cdot v$. Regardless, we can consider the error in the total energy to be proportional to ε_ϕ . We use ε_ϕ instead of ε_a because the binomial is raised to a higher power in ε_a . In any case, as will be seen in the next section, we are most interested in how the error changes with respect to the table size, and for this purpose, the above argument is sufficient.

3.4. Error with respect to the table size

We expand the binomial in Eq. (4),

$$\varepsilon_\phi \leq p \frac{\Delta r}{2r} - \frac{p(p-1)}{2} \left(\frac{\Delta r}{2r} \right)^2 + \dots, \quad (17)$$

where we have dropped the prime on r . Since $\Delta r \ll 2r$ for all reasonable⁴ r , we know that

$$0 < \frac{\Delta r}{2r} \ll 1. \quad (18)$$

Now, we would like to truncate the series to first order, but we must first convince ourselves that the upper-bound still holds in that case. In other words, we must show

$$1 - (1-x)^p \leq px, \quad (19)$$

for $0 \leq x \leq 1$. Since $p \geq 1$, this is true for $x = 1$ and for $x = 0$. Consider the first derivative of both sides with respect to x ,

$$p(1-x)^{p-1} \leq p. \quad (20)$$

Since we are only concerned with the range $0 \leq x \leq 1$ and since $p \geq 1$, this is clearly true. Having established that the slope of the left-hand side is always smaller or equal to the slope of the right-hand side, and that the endpoints of our range of interest satisfy Eq. (19), we can say that Eq. (19) holds for all $0 \leq x \leq 1$.

Truncating Eq. (17) to first order in Δr , we have

$$\varepsilon_\phi \leq p \frac{\Delta r}{2r} \leq p \frac{k}{2rN}, \quad (21)$$

here $\Delta r = k/N$ where $k = r_{\max} - r_{\min}$ is the range of values of r in the table, and N is the number of table entries or the size of the table.

This indicates a linear relationship between the inverse table size and the relative error.

⁴ By reasonable r , we mean any value of r which will normally be encountered in a simulation. This is bounded at the low end by the point at which the pair distribution function $g(r)$ becomes negligible.

3.5. Errors in practice

In practice, the upper-bounds that we have derived in the previous sections are very high. In fact, the actual errors that are incurred overall will be of a much lower magnitude. This is because of several reasons.

First, the error in a table lookup can be on either side of the actual value. Hence, we can expect some “washing out” of the errors when multiple values from the table are summed.

Second, at least for $1/r^p$ potentials, the error is reduced for atoms that are farther apart than r_{\min} . As was stated previously, r_{\min} is the minimum distance that two atoms might approach each other. This is indicated by the point at which the pair distribution function for the substance being simulated goes to zero.

Fig. 4 shows a graph of the pair distribution function for liquid argon overlaid with the maximum error as a function of r . The region of largest error in the force is where the potential is steepest. Also included in the graph is the product of the two functions. The pair distribution function $g(r)$ is proportional to the probability of finding an atom at a distance r around any given atom. The product of the two functions shows a maximum at approximately the location where the two curves cross, and goes to zero with $g(r)$. This demonstrates that distances between neighboring atoms are seldom in the region of largest error. Finally, for the force, it is very unlikely that all of the forces on an atom will point in the same direction. Hence, we should expect the absolute errors to sum in a “random walk” fashion.

For purposes of analysis we have made the simplification that the potential energy is made up of a single term that is always positive. In reality, the potential energy will be a sum of positive and negative contributions from these terms. In such cases we would need to be much more careful in using the relative error since the potential energy and the force may be zero. Nevertheless, we can still do a similar analysis on each term of the potential and then combine the errors after the summation.

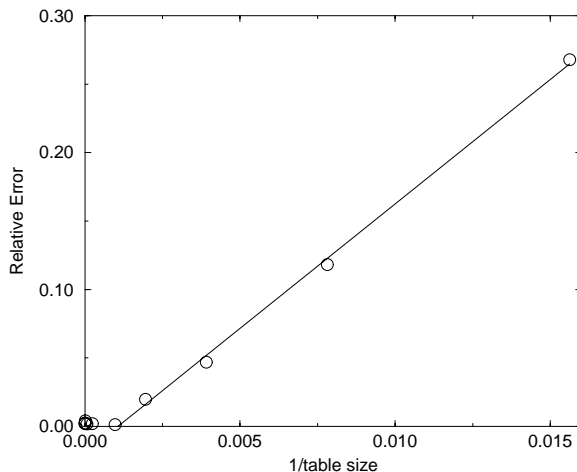


Fig. 6. Relative error in total energy versus inverse table size for the experiment in Fig. 5.

4. Experiments on sample systems

4.1. Variation of table size

Fig. 5 shows the results of tests in which we varied the sizes of the potential and force tables. Each curve is a 50000-step run for LJ liquid argon. The systems consisted of 10648 atoms. We used Rapaport’s code [2] and a leapfrog integrator with reduced time steps of 0.001.

The table sizes are determined by the parameter INDEX_BITS used in the extract() mapping described above. We see that results using tables of sizes 128, 256, and 512 elements for each table move toward the aggregate of results using larger tables, labeled “Others” in the graph, which are essentially indistinguishable. The larger tables contained 2^n elements, for $n = 10, 12, 14, 16, 18,$ and 20 .

We computed the average energy for the last 25000 timesteps for each table size in Fig. 5. Taking the average value of the three curves ($n = 16, 18,$ and 20) which are indistinguishable as the “exact” energy, we plotted relative error versus $1/N$ (where N is the table size) in Fig. 6. This demonstrates the linear relationship predicted in Eq. (21). Hence, we can see that the expression for single particle error contributes in a similar way to overall errors in the energy.

These results indicate that tables of size 2^{12} or larger are more than adequate (at least for the total energy); increasing the table size beyond that has no noticeable

effect on the results for the total energy. This also indicates that interpolation may not be necessary.

We obtained similar results in simulations of 216 silicon atoms in the liquid state using a table for the pair part of the Stillinger–Weber potential [14].

One would expect that so long as the tables are large enough to provide the same results as computed potentials, it makes no difference how large the tables are. However, to do so would be to ignore the cache effect. In the LJ argon case, for example, we found that the time per call to the routine that computes all the forces in a time step increased by 50% when n was increased from 14 to 16, which represent table sizes of 16K and 64K entries, respectively, on an ultrasparc processor with a 1 Mbyte cache. Storage requirements for these tables were 256K and 1M bytes, respectively. There was no further increase in execution time for larger tables up to 2^{20} entries.

4.2. Comparison of output

We now describe our comparisons of the results from simulations that use tabulated potentials and simulations in which the potentials and forces are computed with every step. Table 1 shows the results for LJ liquid argon using the Moldy [10] code. We note that, despite the relatively short times used for equilibration and for accumulating averages, the variations within energy are well within acceptable limits and the standard deviations in energy and temperature show the expected behaviors as the size of the system increases.

Fig. 7 shows the simulation time dependence of the total energy per particle for LJ liquid argon using the two methods. The simulation was for 50000 steps for 10648 atoms using the Rapaport code. While the fluctuations in energy between the two simulations are not synchronized after about 10000 steps, there is no significant difference between them. Fig. 8 shows that there is no significant difference in structure between the results produced by the two methods. The inset shows a small difference in the heights of the second peak. This is close to being within the noise of the function, but could be significant for detailed structural studies. We conclude that, at least for the equation of state and structure studies in which general structural information is needed, there is no important difference in results between the two methods.

5. Performance

We turn now to an analysis of the relative performance of the two methods.

We have focused on three forms for potentials: (1) polynomials in $1/r$, such as the LJ potential, (2) “generalized” potentials, which are combinations of polynomials and exponentials, and (3) three-body potentials, such as the Stillinger–Weber potential for silicon. The latter class involves considerable logic and arithmetic to compute the three-body contribution to the potential in addition to the polynomials and exponentials in the two-body contribution in the potential.

Table 2 shows the performance results for the LJ potential. The simulations were on a system of argon atoms. Kernel times refer to the amount of time spent in the subroutine that calculates the potential and force. The relative performance is reproducible across codes and machines. We used the Rapaport code and Moldy [10] on HP, Sun, and DEC workstations, with little variation in differences in relative performance.

These results might seem disappointing at first, since it would seem that the LJ potential would require a moderate amount of arithmetic. However, a well-coded LJ calculator requires only three multiplications and a subtraction, if r^2 is already available, and a good compiler can rearrange the loop through the atoms and manage the cache to make sure that the pipeline is always full. Furthermore, on modern processors floating-point arithmetic is often faster than the integer arithmetic and random logic our `extract()` routine executes. Hence, one would expect to see very little or no improvement with tabulated potentials for a LJ potential.

Table 3 shows the results for a generalized potential with parameters for liquid argon. The generalized potential has the following form:

$$\phi(r) = A e^{-Br} + \frac{C}{r^{12}} - \frac{D}{r^4} - \frac{E}{r^6} - \frac{F}{r^8}. \quad (22)$$

This experiment was done by inserting a table into a previously established MD code. The code that we used was called “Moldy” [10]. The timings for the kernel routine show that the table method is about a factor of four faster (in kernel time) than direct computation. Our tests on one of the machines on which these calculations were done, a 170 MHz ultrasparc, showed that floating-point multiplications,

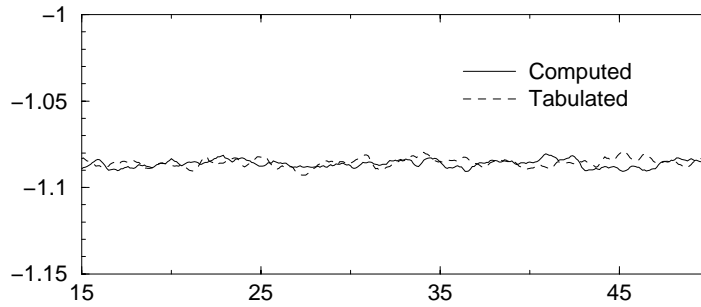


Fig. 7. Total energy per atom using computed and tabulated potentials and forces for a LJ argon system.

Table 1

Results for table and non-table based runs of liquid argon (density 0.8 g/cm^3 and initial temperature 120 K) with the Lennard-Jones potential. Averages are over 45000 timesteps after 5000 timesteps of equilibration with a timestep size of 0.01 ps. No scaling of the temperature is performed

Number of atoms	Energy (kJ/mol)	Std. Dev.	Temperature (K)	Std. Dev.	Type
100	-152.64	0.27	140.53	5.53	Computed
100	-152.75	0.28	140.67	5.52	Tabulated
200	-275.85	0.42	150.38	4.06	Computed
200	-275.97	0.41	150.52	4.06	Tabulated
500	-785.90	0.74	140.28	2.57	Computed
500	-785.73	0.72	140.09	2.48	Tabulated
1000	-1715.21	1.11	132.87	1.73	Computed
1000	-1715.05	1.08	132.85	1.78	Tabulated
5000	-8447.27	2.43	134.13	0.80	Computed
5000	-8448.92	2.55	134.22	0.82	Tabulated
10000	-15930.54	3.38	139.34	0.54	Computed
10000	-15931.17	3.17	139.33	0.57	Tabulated

Table 2

Kernel time and time per call to the force routine for the LJ liquid argon experiments

Experiment	Time in Kernel (s)	Time per force call (ms)
Table, 2^7 elements	877	14.9
Table, 2^{14} elements	946	16.3
Table, 2^{20} elements	1619	29.5
Computed	998	17.3

square roots, and transcendental function evaluations require 0.006, 0.21, and $1.8 \mu\text{s}$, respectively. The exponential functions make the difference.

The speedup is considerably less when one considers total execution time. This is because the computation of the potential is only one part of the computations that need to be done in a given timestep. Integration of the force, measurement of different pa-

rameters, movement of atoms between cells, and occasional neighbor list calculation make up the rest of a single timestep. For example, consider the speedup of a given timestep when the speedup in the kernel is 4,

$$\text{Speedup} = \frac{t_{\text{other}} + t_{\text{kernel}}}{t_{\text{other}} + \frac{1}{4}t_{\text{kernel}}} \quad (23)$$

If the other calculations in a timestep are equally expensive as the kernel computations, then we have

$$\text{Speedup} = \frac{t_{\text{kernel}} + t_{\text{kernel}}}{t_{\text{kernel}} + \frac{1}{4}t_{\text{kernel}}} = \frac{8}{5} \quad (24)$$

So, overall we would see a speedup of only 1.6 as opposed to 4.

Table 4 shows times for simulations of two species of atoms, which requires 3 tables. Here we used a potential of the 6-exp form,

Table 3

Timings for table and non-table based runs of liquid argon using a generalized potential. The kernel refers to the subroutine that calculates the potential and force for a list of pairwise distances. All times are in seconds

Number of Atoms	Time in kernel	Total time	Time per timestep	Type
10000	2969.28	24759.33	4.95	Tabulated
10000	10944.76	33467.90	6.69	Computed
5000	2204.53	17783.34	3.56	Tabulated
5000	7775.09	23188.80	4.64	Computed
1000	370.91	3310.07	0.66	Tabulated
1000	1617.31	4436.66	0.89	Computed
500	203.49	1463.60	0.29	Tabulated
500	689.00	1869.68	0.37	Computed

$$\phi(r) = -\frac{A}{r^6} + B e^{-Cr}, \quad (25)$$

which is similar to the generalized potential. When one compares the timings for this potential to the timings for the Lennard-Jones potential the significance of the exponential term with respect to speedup becomes clear. We observe that the results at first glance do not indicate a speedup as large as expected, since with one or two more multiplications we would have the generalized potential for which we measured a factor of 4 speedup. This is because there is now more than one table, and extra logic is included in the table lookup stage to determine which table to use. The reason this is necessary is due to specifics of the Moldy code. Hence, some optimizations by the compiler for the innermost loop are not implemented, not to mention the additional instructions needed to execute conditional statements. Nevertheless, the speedup demonstrates the significant contribution of the exponential.

Table 5 shows the performance of the methods on silicon, 4096 atoms for 50000 time steps using the Rapaport code. In generating the code for the force calculations, we were able to tabulate only the two-body terms. The three-body terms, as usually written, require the calculation of the cosine of the angle between each triplet of atoms that are within a cutoff distance of each other. The cosine is a dot product between two of the vectors linking the atoms. There is little point in constructing tables for these because they would require a table indexed by r_{12} , r_{13} , and $\cos(\theta)$. Hence, the table would have to be three-dimensional which would increase the storage requirements by about a power of three. Also, very little arithmetic is saved since much of the work is in doing the dot product.

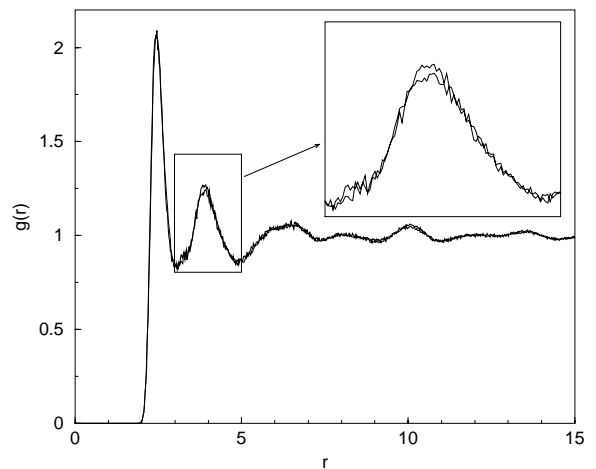


Fig. 8. Pair-correlation function from computed and tabulated potentials and forces for a liquid silicon system.

Nakano et al. [15] have derived a means for separating the three-body force into sets of two-body interactions. It would be interesting to see the gains that can be made through using their technique with a tabulated potential.

6. Conclusions

In Table 6 we provide approximate ratios of times in force calculations for computed and tabulated potentials. One message is clear. One should certainly use tables for potentials when the potential involves transcendental function calls and other intensive computations with relatively little logic. The case for more complex potentials, such as three-body potentials and problems that require multipole expansions is still

Table 4

Timings for table and non-table based runs of a SiO₂ system with 192 Silicon atoms and 384 Oxygen atoms. We see an improvement in speed of about a factor of 1.18 in execution time and 1.72 in kernel time. (Evaluation of the Coulomb interaction was omitted for the purpose of this study.)

Time in kernel	Total time	Seconds per timestep	Type
370.80	1529.64	0.77	Tabulated
639.15	1810.45	0.91	Computed

Table 5

Kernel time and time per call to the force routine for the silicon experiments

Experiment	Milliseconds per force call per atom
Table, 216 atoms, 5000 steps	0.79
Table, 4096 atoms, 50000 steps	0.75
Computed, 216 atoms, 5000 steps	0.98

Table 6

Approximate ratios for computing potentials versus table lookups for three classes of potential

Potential	Compute/Table time
Lennard-Jones	1/1
6-exp	7/4
Generalized	4/1
StillingerWeber	5/4

cloudy.

We have shown that the use of tabulated potentials can be effective in speeding up MD simulations. We have outlined different tradeoffs that need to be taken into account before implementing a table into a particular code. The number of different species to be simulated, complexity of the potential, and cache size are all important considerations. We have also investigated the errors introduced by tabulated potentials. Surprisingly small tables of a few thousand elements produce results that are indistinguishable from those obtained using double precision arithmetic.

References

- [1] M.P. Allen, J.P. Tildesley, *Computer Simulation of Liquids* (Oxford Science Publications, Oxford, 1987).
- [2] D.C. Rapaport, *The Art of Molecular Dynamics Simulation* (Cambridge Univ. Press, Cambridge, 1995).
- [3] R.W. Hockney, J.W. Eastwood, *Computer Simulation Using Particles* (Adam Hilger, Bristol, 1988).
- [4] Private communications with several investigators.
- [5] P. Clapp, J. Rifkin. Personal communication; see <http://www.ims.uconn.edu/centers/simul/>.
- [6] D. Turner, J. Morris, see http://cmp.ameslab.gov/cmp/CMP_Theory/cmd/alcmd_source.html.
- [7] D.E. Sanders M.S. Stave, L.S. Perkins, A.E. DePristo, *Comput. Phys. Commun.* 70 (1992) 579.
- [8] M.S. Stave D.E. Sanders, T.J. Raeker, A.E. DePristo, *J. Chem. Phys.* 93 (1990) 4413.
- [9] A.Y. Toukmaji, J. John A. Board, *Comput. Phys. Commun.* 95 (1996) 73.
- [10] K. Refson, Moldy Users Manual, see <http://www.earth.ox.ac.uk/~keith/moldy.html>.
- [11] M. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L. Kale, R.D. Skeel, K. Schulten, *J. Supercomput. Appl. & High Performance Computing* (1996).
- [12] D.M. Beazley, P.S. Lomdahl, *Parall. Comput.* 20 (1994) 173–195.
- [13] T.A. Andrea, W.C. Swope, H.C. Andersen, *J. Chem. Phys.* 79 (1983) 4576.
- [14] F.H. Stillinger, T.A. Weber, *Phys. Rev. B* 31 (1985) 5262.
- [15] A. Nakano, P. Vashishta, R.K. Kalia, *Comput. Phys. Commun.* 77 (1993) 303.