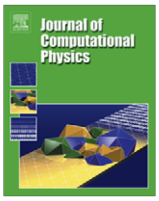
Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp



Using piecewise polynomials for faster potential function evaluation

Pedro Gonnet [*]

Department of Computer Science, ETH Zurich, 8092 Zürich, Switzerland

A R T I C L E   I N F O

Article history:
Received 19 January 2009
Received in revised form 23 September 2009
Accepted 23 September 2009
Available online 2 October 2009

MSC:
65D05
65D07
65D10
65D20
65Y99

PACS:
02.30.Mv
02.60.Ed
02.60.Gf
02.70.Ns
31.15.xv
83.10.Rs

Keywords:
Molecular dynamics
Potential function
Piecewise polynomial interpolation
Horner scheme
Chebyshev polynomials

A B S T R A C T

In many molecular dynamics simulation software packages and hardware implementations, piecewise polynomials are used to represent and compute pairwise potential functions efficiently. In this paper, we present three modifications applicable to most interpolations to increase their accuracy. The increased accuracy reduces the amount of data that needs to be stored for each interaction potential, making such interpolations more suitable for architectures with limited memory and/or cache or hardware implementations.

1. Introduction

In most molecular dynamics or Monte-Carlo simulations, the most expensive part of each time step is the evaluation of the non-bonded pairwise interactions [1–3]. Given a pair of particles p_i and p_j of the species A and B respectively, the *interaction energy*

$$e_{ij} = v_{AB}(r_{ij})$$

is computed from the *interaction potential* v_{AB} specific to the particle species A and B and the inter-particle distance r_{ij}. The resulting *interaction force* on the particles p_i and p_j is the gradient of the potential with respect to the particle coordinates

* Tel.: +41 446324552.
E-mail address: gonnetp@inf.ethz.ch.

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.09.028

Algorithm 1. Naive interaction of two particles p_i and p_j of species A and B respectively

```
1:  $\mathbf{r}_{ij} \leftarrow \mathbf{x}_i - \mathbf{x}_j$    (compute the inter-particle vector)
2:  $r_{ij} \leftarrow \|\mathbf{r}_{ij}\|_2$    (compute the inter-particle distance)
3:  $\mathbf{f}_{ij} \leftarrow \mathbf{0}, e_{ij} \leftarrow 0$    (initialize the force and energy)
4:  if $AB$ interact with a Lennard–Jones 12-6 potential then
5:      $e_{ij} \leftarrow e_{ij} + 4\varepsilon_{AB}\left[\left(\frac{\sigma_{AB}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{AB}}{r_{ij}}\right)^{6}\right]$
6:      $\mathbf{f}_{ij} \leftarrow \mathbf{f}_{ij} + \mathbf{r}_{ij}\frac{24\varepsilon_{AB}}{\sigma_{AB}^2}\left[-2\left(\frac{\sigma_{AB}}{r_{ij}}\right)^{14} + (\frac{\sigma_{AB}}{r_{ij}})^{8}\right]$
7:  else if $AB$ interact with a Morse potential then
8:      $e_{ij} \leftarrow e_{ij} + D_{AB}\left[1 - \exp\left(-\sqrt{k_{AB}/2D_{AB}}(r_{ij} - r_{AB})\right)\right]^2 + e_{AB}$
9:      $\mathbf{f}_{ij} \leftarrow \mathbf{f}_{ij} + \frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\partial}{\partial r_{ij}}D_{AB}\left[1 - \exp\left(-\sqrt{k_{AB}/2D_{AB}}(r_{ij} - r_{AB})\right)\right]^2$
10: else if $AB$ interact with a ... potential then
11:     $e_{ij} \leftarrow e_{ij} + \dots$
12:     $\mathbf{f}_{ij} \leftarrow \mathbf{f}_{ij} + \dots$
13: end if
14: if $AB$ interact with a Coulomb potential then
15:     $e_{ij} \leftarrow e_{ij} + \frac{q_1 q_2}{r_{ij}}$
16:     $\mathbf{f}_{ij} \leftarrow \mathbf{f}_{ij} + \mathbf{r}_{ij}\frac{q_1 q_2}{r_{ij}^3}$
17: else if $AB$ interact with an Ewald potential then
18:     $e_{ij} \leftarrow e_{ij} + \frac{q_1 q_2}{r_{ij}}\mathrm{erfc}(\kappa r_{ij})$
19:     $\mathbf{f}_{ij} \leftarrow \mathbf{f}_{ij} - \mathbf{r}_{ij}\frac{q_1 q_2}{r_{ij}^3}\left[\mathrm{erfc}(\kappa r_{ij}) - \frac{2\kappa r_{ij}}{\sqrt{\pi}}\exp(-\kappa^2 r_{ij}^2)\right]$
20: else if $AB$ interact with a ... potential then
21:     $e_{ij} \leftarrow e_{ij} + \dots$
22:     $\mathbf{f}_{ij} \leftarrow \mathbf{f}_{ij} + \dots$
23: end if
```

$$\mathbf{f}_{ij} = -\nabla_{\mathbf{x}_i} v_{AB}(r_{ij}) = \nabla_{\mathbf{x}_j} v_{AB}(r_{ij}).$$

The computation of the pairwise energy and force can be implemented naively as shown in Algorithm 1. This naive computation has some obvious drawbacks:

(i) the relatively expensive evaluation of arithmetic operations such as $\sqrt{\cdot}$ or $(\cdot)^{-1}$, e.g. when computing the inter-particle distance r_{ij} or within the potentials themselves,
(ii) the relatively expensive evaluation of transcendental functions such as $\mathrm{erfc}(\cdot)$ or $\exp(\cdot)$ in the computation of the more complicated potentials,
(iii) the cascading conditional statements (**if-then-else** statements) can cause stalls on processors with long instruction pipelines or no branch prediction[1] and make exploiting SIMD parallelism more difficult,
(iv) the size of the interaction computation can cause problems on computers with small instruction caches or on hardware implementations where die surface and complexity are critical.

Problem (iii) can, in some cases, be avoided by implementing a separate interaction loop for each interaction type. This would, however, require the list of interacting particle pairs to be traversed more than once. This inefficiency can easily offset whatever advantage was obtained by avoiding conditional branches in the first place.

It is for these and other[2] reasons that several authors have opted to compute not the exact potentials, as is done in Algorithm 1, but to compute, store and evaluate an approximation of the potential function:

$$g_{AB}(r_{ij}) \approx v_{AB}(r_{ij})$$

The approximation $g_{AB}(r_{ij})$ is usually a function of r_{ij}^2 to avoid evaluating the $\sqrt{\cdot}$ to compute r_{ij}:

$$g_{AB}(r_{ij}^2) \approx v_{AB}(r_{ij}).$$

The approximated potential is then usually represented as a set of n piece-wise polynomials between a set of nodes $x_i,\ i = 0 \dots n$:

[1] IBM's Cell Broadband Engine [4], for example, has no dynamic branch prediction capabilities, incurring a penalty of 18–19 cycles for each mis-predicted branch.

[2] GROMACS, for example, uses an interpolated potential only for tabulated user-supplied potentials.

$$g_{AB}(x) = \begin{cases} g_0(x) & x_0 \leqslant x < x_1 \\ g_1(x) & x_1 \leqslant x < x_2 \\ \vdots & \\ g_{n-1}(x) & x_{n-1} \leqslant x < x_n \end{cases}$$

where the coefficients $b_k^{(i)}$ of each individual polynomial of degree m,

$$g_i(x) = \sum_{k=0}^{m} b_k^{(i)} x^k,$$

are tabulated. Algorithm 2 shows how this could be implemented naively. Note that only additions and multiplications are used in the evaluation of the potential and no conditional expressions are required.[3] This method, however, is not more efficient *per se*: the more accurately a potential needs to be modelled, the larger the number of nodes n or the higher the degree m of the polynomials will be necessary, which in turn leads to larger tables for the coefficients. If the resulting tables become too large, they may fail to fit comfortably in the processor cache resulting in overhead due to cache misses, slowing the computation down significantly. Large tables may also be unsuitable for on-chip hardware implementations or on systems using sub-cores (e.g. GPUs or IBM's Cell/BE processor) with small local memories and caches.[4]

Algorithm 2. Compute the interaction between two particles p_i and p_j of species A and B respectively using a piecewise polynomial approximation

```
 1: $\mathbf{r}_{ij} \leftarrow \mathbf{x}_i - \mathbf{x}_j$                    (compute the inter-particle vector)
 2: $r_{ij}^2 \leftarrow \|\mathbf{r}_{ij}\|_2^2$                             (compute the squared distance)
 3: load the nodes $x_i$ and coefficients $b_k$ for the species pair $AB$
 4: find $i$ such that $x_i \leqslant r_{ij}^2 < x_{i+1}$
 5: $e_{ij} \leftarrow b_m^{(i)} r_{ij}^2 + b_{m-1}^{(i)}$       (evaluate $e_{ij}$ and $f_{ij}$ using the Horner Scheme)
 6: $f_{ij} \leftarrow b_m^{(i)}$
 7: for $k = m-2 \ldots 0$ do
 8:   $f_{ij} \leftarrow f_{ij} r_{ij}^2 + e_{ij}$
 9:   $e_{ij} \leftarrow e_{ij} r_{ij}^2 + b_k^{(i)}$
10: end for
11: $\mathbf{f}_{ij} \leftarrow \mathbf{r}_{ij} f_{ij}/2$                       (construct the force vector)
```

In [5], Andrea et al. are the first to describe such an approach for simulations run on a PDP-11/70 using an FPS AP120-B array processor. They represent the potential as a piecewise fifth-degree Hermite interpolation polynomial matching the potential and its first two derivatives at the edges of each interval. The interpolation is computed as a function of r_{ij}^2 and the piecewise polynomials are evaluated using the Horner scheme, which requires only additions and multiplications. Since the interpolation is smooth and its derivatives approximate those of the original potential, the derivative of the interpolation can be used to compute the magnitude of the inter-particle force.

Allen and Tildesley [1] suggest storing a table of evaluations of the potential at fixed intervals x_i^2, $i = 0 \ldots n$. The kth interval containing the distance $x_k^2 \leqslant r_{ij}^2 < x_{k+1}^2$ is found and the potential is computed by evaluating the interpolation of the potential at the nodes $x_{k-1}^2, x_k^2, x_{k+1}^2$ and x_{k+2}^2 at the node r_{ij}^2 on the fly using Newton's forward difference formula. The resulting interpolation is not smooth over the nodes x_k^2 and, if continuity of the first derivative (i.e. the interaction force) is required, a separate table for the force must be stored and evaluated. The thus interpolated force, however, will not be the exact derivative of the interpolated energy, which can cause inconsistencies in the simulation.

The GROMACS simulation package [2] uses "cubic splines" to interpolate tabulated user-supplied potential function such that they match the potential and its *second* derivative at the interval edges [6]. Unfortunately, if these second derivatives are not chosen such as they are in "normal" cubic splines, the first derivative will not be continuous over the interval edges, which may cause problems if continuity of the interaction force is required. The interpolation is constructed over r_{ij} and not r_{ij}^2. As we will see later, this offers some advantages.

A somewhat different approach is taken by the MD-GRAPE project [7], which produces specialized hardware for MD simulations: fourth-order piecewise polynomials are generated for some basic functions such as

$$g_{LJ}(x) = x^{-4} - x^{-7} \quad \text{or} \quad g_C(x) = x^{-3/2}$$

[3] If we assume that the interval search can be implemented without conditional statements (e.g. if the x_i are uniformly distributed, which is usually the case) and that the loops in Lines 5–10 are unrolled.

[4] The Cell/BE processor's Synergistic Processing Elements (SPE) have a local store of 256 kB [4].

Table 1
Different types of piecewise interpolation polynomials.

Method	Variable	Degree	Constraints	Notes
Andrea et al.	$x = r_{ij}^2$	5	$g_i(x_i) = f(x_i), g_i(x_{i+1}) = f(x_{i+1})$ $g_i'(x_i) = f'(x_i), g_i'(x_{i+1}) = f'(x_{i+1}),$ $g_i''(x_i) = f''(x_i), g_i''(x_{i+1}) = f''(x_{i+1})$	The interpolation nodes x_i are chosen irregularly
Allen and Tildesley	$x = r_{ij}^2$	3	$g_i(x_i) = f(x_i), g_i(x_{i+1}) = f(x_{i+1}),$ $g_i(x_{i-1}) = f(x_{i-1}), g_i(x_{i+2}) = f(x_{i+2})$	$f(x)$ and $f'(x)$ must be interpolated separately and will not be consistent
GROMACS	$x = r_{ij}$	3	$g_i(x_i) = f(x_i), g_i(x_{i+1}) = f(x_{i+1}),$ $g_i'(x_{i+1}) = g_{i+1}'(x_{i+1})$ (for $i = 0 \ldots n-1$) $g_i''(x_{i+1}) = g_{i+1}''(x_{i+1})$ (for $i = 0 \ldots n-1$) $g_0''(x_0) = 0, g_{n-1}''(x_n) = 0$	We use the natural spline conditions such that the first derivative is continuous
MD-GRAPE	$x = r_{ij}^2$	4	$g_i(x_i) = f(x_i), g_i(x_{i+1}) = f(x_{i+1}),$ $g_i'(x_i) = f'(x_i), g_i'(x_{i+1}) = f'(x_{i+1}),$ $g_i''(x_{i+1}) = g_{i+1}''(x_{i+1})$ (for $i = 0 \ldots n-1$)	The second derivative smoothness condition is only assumed since the authors do not state how the interpolation is constructed
Bowers et al.	$x = ar_{ij}^2 + b$	3	$g_i(x_i) = f(x_i), g_i(x_{i+1}) = f(x_{i+1}),$ $g_i'(x_i) = f'(x_i), g_i'(x_{i+1}) = f'(x_{i+1})$	r_{ij}^2 is transformed to $x \in [0, 1]$
This paper	$x = ar_{ij} + b$	5	$g_i(x_i) = f(x_i), g_i(x_{i+1}) = f(x_{i+1}),$ $g_i'(x_i) = f'(x_i), g_i'(x_{i+1}) = f'(x_{i+1}),$ $\int_{x_i}^{x_{i+1}} [g_i(x) - f(x)]^2 \,d\lambda(x) = \min$	r_{ij} is transformed to $x \in [0, 1]$

which are then used to compute arbitrarily parameterized potentials. The generalized Lennard–Jones potential, for instance, can be evaluated using

$$e_{ij} = 4\varepsilon_{AB}\left[\left(\frac{\sigma_{AB}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{AB}}{r_{ij}}\right)^{6}\right] = \frac{4}{\sigma_{AB}^2} r_{ij}^2 g_{LJ}\left(\frac{r_{ij}^2}{\sigma_{AB}^2}\right)$$
$$\mathbf{f}_{ij} = \mathbf{r}_{ij}\frac{24\varepsilon_{AB}}{\sigma_{AB}^2}\left[-2\left(\frac{\sigma_{AB}}{r_{ij}}\right)^{14} + \left(\frac{\sigma_{AB}}{r_{ij}}\right)^{8}\right] = \mathbf{r}_{ij}\frac{24\varepsilon_{AB}}{\sigma_{AB}^2 2^{4/3}} g_{LJ}\left(\frac{r_{ij}^2}{2^{1/3}\sigma_{AB}^2}\right)$$

Unfortunately, in the published literature surrounding the MD-GRAPE project, no information is given on how the interpolation is constructed other than its degree and the number of intervals (1024).

More recently, Bowers et al. have presented the simulation software *Desmond* [8] which uses uniformly distributed [9] piecewise cubic polynomials of the transformed variable $x = ar_{ij}^2 + b$ such that $x \in [0, 1]$ to compute the pairwise electrostatic interactions which would otherwise require the evaluation of the transcendental functions erfc$(\cdot)$ and exp$(\cdot)$. The polynomials interpolate the potential and its first derivative at the edges of each interval. The same approach, yet using variable-width[5] polynomial segments, is used in *Anton* [10], a special-purpose computer for molecular dynamics which implements the pairwise interactions in hardware, for both Lennard–Jones and electrostatic interactions in two separate pipelines.

The different interpolation schemes are summarized in Table 1. Since Taiji et al. (the authors of the MD-GRAPE project) give no specifications as to how they construct their fourth-degree interpolation polynomial, we will assume, for the sake of comparison, that they construct a Hermite polynomial interpolating the potential and its first derivative at the interval edges plus an additional continuity constraint on the second derivative. Since this results in $5n - 1$ constraints for $5n$ unknowns (the coefficients of the piecewise polynomials), we add the additional constraint $g_{n-1}''(x_n) = 0$. We will also assume, for continuity, that natural cubic splines are indeed used in GROMACS and we will choose the second derivative accordingly and not such as to match that of the potential.

The choice of which conditions (continuity of interpolatory) are required to be satisfied by the interpolations depends on the characteristics of the underlying simulation algorithm. In Monte-Carlo type simulations we could dispense with continuity constraints altogether since we only need to sample the physical space and not traverse it continuously. Molecular dynamics simulations using first-order time integration schemes or linear energy minimization algorithms, however, require continuous first derivatives. Higher-order time integration or energy minimization schemes may require the continuity of further higher-order derivatives. If the exact total potential energy of the interactions is not required, we can dispense with the conditions on the zeroth derivative, (e.g. the potential function itself) altogether and interpolate only its first and successive derivatives.

In our new approach, listed at the bottom of Table 1, we will differ from the other methods in three major points:

(i) Instead of increasing the degree of the interpolation polynomials by adding additional Hermite or smoothness constraints, we will require that the interpolation error with respect to some measure $\lambda(x)$ is minimal.

[5] Unfortunately, in the literature surrounding the Anton project, no information is given as to how the intervals are chosen.

(ii) We will not construct the interpolation on r_{ij}^2 but rather on r_{ij}, risking the cost of a $\sqrt{\cdot}$ operation.[6]
(iii) Instead of distributing the nodes x_i uniformly in the interval of interpolation, we will first map the interval to $[0,1]$ and select the interval using $i = \lfloor n(\alpha x + (1-\alpha)x^2)\rfloor$ as opposed to $i = \lfloor x/n \rfloor$.

We will assume that continuity of only the zeroth and first derivative are required, as is done by [8]. As we will see later, the techniques described herein can be easily adapted to satisfy any combination of continuity constraints imposed by the underlying simulation algorithm.

2. Error-minimizing Hermite interpolation polynomials

In all previously published methods, the degree of the interpolating polynomial is dictated by the number of prescribed matching derivatives or smoothness constraints. In the following, we will only require that our interpolation matches the zeroth and first derivatives of the potential at the interval boundaries. Additional degrees of freedom are then fitted by requiring that the resulting interpolation minimizes

$$\int_{x_i}^{x_{i+1}} [g_i(x) - f(x)]^2 \, \mathrm{d}\lambda(x) \tag{1}$$

in each interval, where $f(x)$ is the interpolated function and $\lambda(x)$ is a positive measure on $x \in [x_i, x_{i-1}]$.

For our interpolations, we will use the Chebyshev measure

$$\mathrm{d}\lambda(x) = \frac{\mathrm{d}x}{\sqrt{1-x^2}}, \quad x \in [-1,1]. \tag{2}$$

We chose this measure since the resulting polynomial minimizing Eq. (1) is near-optimal in the *min–max* sense for sufficiently smooth $f(x)$ [11], i.e. the resulting interpolation minimizes the maximum absolute error.

Accordingly, we will transform the interval $[x_i, x_{i+1}]$ to $[-1,1]$ and define the interpolant such that

$$g_i(x) \approx \hat{f}_i(x) = f\left(\frac{x_i + x_{i+1}}{2} + \frac{x_{i+1} - x_i}{2} x\right), \quad x \in [-1,1].$$

We will represent the $g_i(x)$ of degree m as linear combinations of the Chebyshev polynomials $T_k(x)$ of degree k:

$$g_i(x) = \sum_{k=0}^{m}{}' c_k^{(i)} T_k(x).$$

where Σ' denotes a sum in which the first term is halved.

We can compute the first four coefficients $c_k^{(i)}$, $k = 0 \ldots 3$ of an initial estimate $\tilde{g}_i(x)$ such that the zeroth and first derivatives at the interval edges match the interpoland by solving a Vandermonde-like system of linear equations

$$\begin{pmatrix} \frac{1}{2}T_0(-1) & T_1(-1) & T_2(-1) & T_3(-1) \\ \frac{1}{2}T_0(1) & T_1(1) & T_2(1) & T_3(1) \\ \frac{1}{2}T_0'(-1) & T_1'(-1) & T_2'(-1) & T_3'(-1) \\ \frac{1}{2}T_0'(1) & T_1'(1) & T_2'(1) & T_3'(1) \end{pmatrix} \begin{pmatrix} c_0^{(i)} \\ c_1^{(i)} \\ c_2^{(i)} \\ c_3^{(i)} \end{pmatrix} = \begin{pmatrix} \hat{f}_i(-1) \\ \hat{f}_i(1) \\ \hat{f}_i'(-1) \\ \hat{f}_i'(1) \end{pmatrix}. \tag{3}$$

We then define the *corrections*

$$\gamma_k(x) = \sum_{j=0}^{k}{}' \eta_j^{(k)} T_j(x)$$

for $k > 3$ as the polynomials whose kth coefficient $\eta_k^{(k)}$ is one and who's first four coefficients $\eta_j^{(k)}$, $j = 0 \ldots 3$ are chosen such that

$$\gamma_k(-1) = 0, \quad \gamma_k(1) = 0, \quad \gamma_k'(-1) = 0, \quad \gamma_k'(1) = 0. \tag{4}$$

The coefficients can be computed similarly to Eq. (3) (see Line 7 of Algorithm 3). The remaining coefficients $\eta_j^{(k)}, 3 < j < k$ are set to zero.

For convenience we will orthogonalize the coefficients of each kth correction with the previous $j = 0 \ldots k-1$ corrections using Gram–Schmidt orthogonalization:

$$\eta_i^{(k)} \leftarrow \eta_i^{(k)} - \eta_i^{(j)} \frac{\sum_{l=0}^{j} \eta_l^{(k)} \eta_l^{(j)}}{\sum_{l=0}^{j} \left(\eta_l^{(j)}\right)^2}.$$

6 The interpolation used by GROMACS already incorporates this feature, but the interpolation is not used for performance reasons and the decision to use r_{ij} seems to have been more a matter of convenience.

If the coefficients are orthogonal, then so are the polynomials themselves with respect to Eq. (2). Note that since we are subtracting multiples of functions that satisfy Eq. (4), these conditions themselves are preserved.

Due to Eq. (4) we can add any multiple of $\gamma_k(x)$ to $\tilde{g}_i(x)$ without altering the Hermite interpolation at the interval edges (see Fig. 1). We then only need to choose the weights α_k such that

$$g_i(x) = \tilde{g}_i(x) + \alpha_4\gamma_4(x) + \alpha_5\gamma_5(x) + \dots \tag{5}$$

minimizes Eq. (1).

We will compute $g_i(x)$ incrementally, adding each $\alpha_k\gamma_k(x)$ one after the other. Since the individual $\gamma_k(x)$ are orthogonal with respect the measure in Eq. (2), it makes no difference in which order they are added.

In order to compute each weight α_k, we first re-formulate Eq. (1) using Eq. (5):

$$\int_{-1}^{1}\left[g_i(x) - \hat{f}_i(x)\right]^2 \mathrm{d}\lambda(x) = \int_{-1}^{1}\left[\tilde{g}_i(x) + \alpha_k\gamma_k(x) - \hat{f}_i(x)\right]^2 \mathrm{d}\lambda(x)$$

which we can minimize for α_k by setting its derivative with respect to the latter to zero

$$\int_{-1}^{1} 2\gamma_k(x)\left[\tilde{g}_i(x) + \alpha_k\gamma_k(x) - \hat{f}_i(x)\right] \mathrm{d}\lambda(x) = 0$$

and solving for α_k:

$$\alpha_k = -\frac{\int_{-1}^{1}\gamma_k(x)\left[\tilde{g}_i(x) - \hat{f}_i(x)\right]\mathrm{d}\lambda(x)}{\int_{-1}^{1}\gamma_k(x)^2\,\mathrm{d}\lambda(x)}. \tag{6}$$

Since we have represented $\tilde{g}_i(x)$ and $\gamma_k(x)$ using Chebyshev polynomials orthogonal with respect to the measure $\lambda(x)$, we can evaluate Eq. (6) as

$$\alpha_k = \frac{\sum_{j=0}^{k}\eta_j^{(k)}\left(c_j^{(i)} - \phi_j\right)}{\sum_{j=0}^{k}\left(\eta_j^{(k)}\right)^2}$$

where the coefficients ϕ_j are the Chebyshev coefficients of $\hat{f}_i(x)$ such that

$$\hat{f}_i(x) = \sum_{j=0}^{\infty}\phi_j T_j(x), \quad x \in [-1, 1].$$

These coefficients can be sufficiently well approximated using

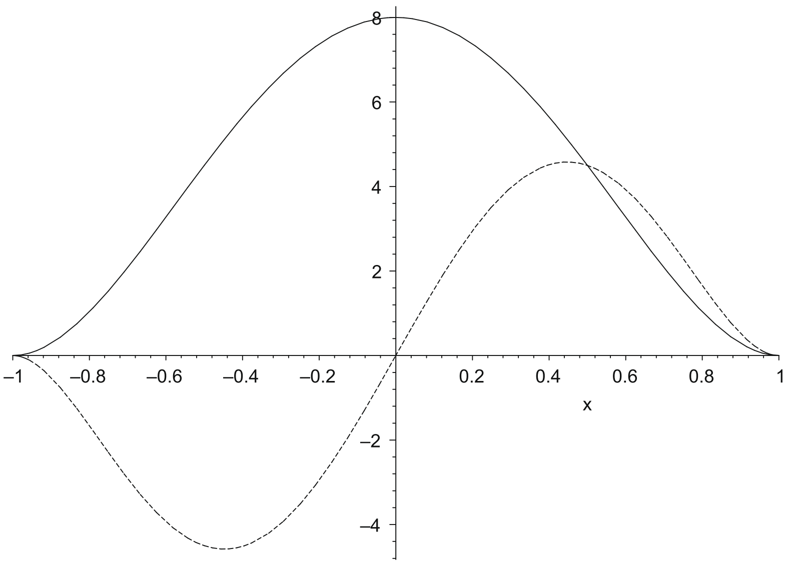


Fig. 1. The corrections $\gamma_4(x)$ (solid) and $\gamma_5(x)$ (dashed).

$$\phi_j = \frac{2}{N}\sum_{i=0}^{N}{}'' \hat{f}_i\left(\cos\frac{i\pi}{N}\right)\cos\frac{ij\pi}{N}$$

for some large N, where Σ'' denotes a sum in which the first and last terms are halved.

Algorithm 3. interp(f, x_i, x_{i+1}, m)

```
1:  $\hat{f}_i(x) \leftarrow f(\frac{x_i+x_{i+1}}{2} + \frac{x_{i+1}-x_i}{2}x)$
2:  $\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} \leftarrow \begin{pmatrix} \frac{1}{2}T_0(-1) & T_1(-1) & T_2(-1) & T_3(-1) \\ \frac{1}{2}T_0(1) & T_1(1) & T_2(1) & T_3(1) \\ \frac{1}{2}T'_0(-1) & T'_1(-1) & T'_2(-1) & T'_3(-1) \\ \frac{1}{2}T'_0(1) & T'_1(1) & T'_2(1) & T'_3(1) \end{pmatrix}^{-1} \begin{pmatrix} \hat{f}(-1) \\ \hat{f}(1) \\ \hat{f}'(-1) \\ \hat{f}'(1) \end{pmatrix}$
3:  for $j = 0 \ldots m$ do
4:      $\phi_j \leftarrow \frac{2}{N}\sum_{i=0}^{N}{}''\hat{f}(\cos\frac{i\pi}{N})\cos\frac{ij\pi}{N}$
5:  end for
6:  for $k = 4 \ldots m$ do
7:      $\begin{pmatrix} \eta_0^{(k)} \\ \eta_1^{(k)} \\ \eta_2^{(k)} \\ \eta_3^{(k)} \end{pmatrix} \leftarrow \begin{pmatrix} \frac{1}{2}T_0(-1) & T_1(-1) & T_2(-1) & T_3(-1) \\ \frac{1}{2}T_0(1) & T_1(1) & T_2(1) & T_3(1) \\ \frac{1}{2}T'_0(-1) & T'_1(-1) & T'_2(-1) & T'_3(-1) \\ \frac{1}{2}T'_0(1) & T'_1(1) & T'_2(1) & T'_3(1) \end{pmatrix}^{-1} \begin{pmatrix} T_k(-1) \\ T_k(1) \\ T'_k(-1) \\ T'_k(1) \end{pmatrix}$
8:      $\eta_k^{(k)} \leftarrow 1$
9:      for $j = 4 \ldots k-1$ do
10:         $w \leftarrow \sum_{l=0}^{j}\eta_l^{(k)}\eta_l^{(j)} / \sum_{l=0}^{j}\left(\eta_l^{(j)}\right)^2$
11:         for $l = 0 \ldots j$ do $\eta_l^{(k)} \leftarrow \eta_l^{(k)} - w\eta_l^{(j)}$ end for
12:     end for
13:     $\alpha_k \leftarrow \frac{\sum_{j=0}^{k}\eta_j^{(k)}\left(c_j^{(i)} - \phi_j\right)}{\sum_{j=0}^{k}\left(\eta_j^{(k)}\right)^2}$
14:     for $j = 0 \ldots k$ do
15:         $c_j \leftarrow c_j + \alpha_k\eta_j^{(k)}$
16:     end for
17: end for
```

The entire computation is shown in Algorithm 3 and Fig. 2. Note that for any different set of constraints, we only need to re-define Eqs. (3) and (4), without modifying any other part of the algorithm.

The resulting interpolants are represented using their Chebyshev coefficients $c_k^{(i)}, k = 0 \ldots m$. To evaluate this representation we could convert them to their monomials equivalents using the matrix M with

$$M_{ij} = i\text{th monomial coefficient of } T_j(x)$$

and computing $\underline{b}^{(k)} = M\underline{c}^{(k)}$. Alternatively, we could keep the coefficients $c_i^{(k)}$ and use the Clenshaw algorithm [12] instead of the Horner scheme [13] in Lines 5–10 of Algorithm 2. Note also that the thus constructed interpolations are for the interval $[-1, 1]$ as opposed to $[x_i, x_{i+1}]$. We therefore need to either transform $x \in [x_i, x_{i+1}]$ to the interval $[-1, 1]$ or shift the polynomials from $[-1, 1]$ to the required interval $[x_i, x_{i+1}]$. Although the former approach is numerically stable for any value of x, it would require the storage of the interval centers and widths to transform $x \in [x_i, x_{i+1}]$ to $[-1, 1]$.

Table 2 shows the number of intervals required for each of the methods shown in Table 1 to achieve an absolute error of at most 0.01 kJ/mol when applied to a Lennard–Jones potential with $\varepsilon = 1$ kJ/mol and $\sigma = 0.3$ nm for $r_{ij} \in [0.3, 1]$. The new interpolation requires significantly fewer intervals than that of Andrea et al. which uses polynomials of the same degree ($m = 5$), yet fitted to higher derivatives.

3. Interpolating on r_{ij} instead of on r_{ij}^2

In almost all interpolated interaction potentials, the potential energy is computed as a function of r_{ij}^2 as opposed to its definition over r_{ij}. This was originally motivated by the need to avoid expensive arithmetic operations such as the $\sqrt{\cdot}$ used to compute r_{ij} from $r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2$. However, on most modern architectures, the $\sqrt{\cdot}$ instruction is implemented efficiently and/or can be pipelined.[7] This relaxed cost leads us to reconsider the advantages of not computing r_{ij}.

[7] Several processors offer fast instructions which return initial estimates of the reciprocal square root, (e.g. the `frsqrte` instruction in the IBM PowerPC Architecture [14]) which can be used to approximate the exact value iteratively. These computations can be inlined and pipelined with other computations.

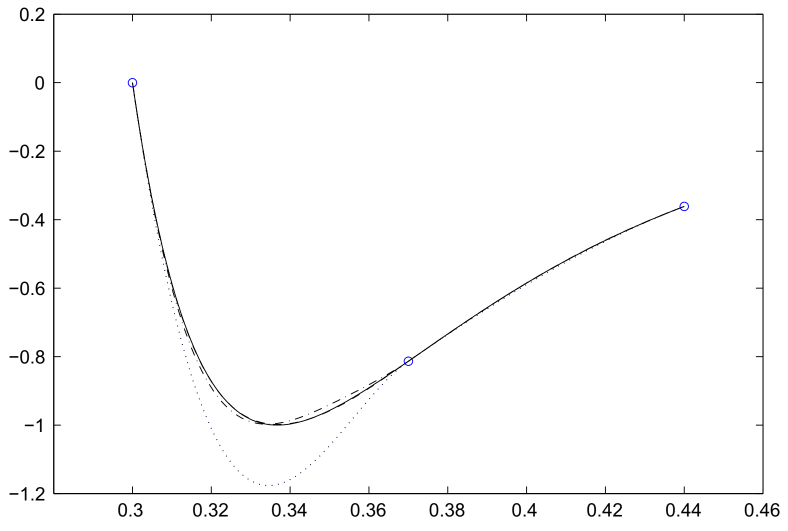


Fig. 2. Interpolation of a Lennard–Jones potential with $\varepsilon = 1$ kJ/mol and $\sigma = 0.3$ nm (—) using $\tilde{g}_i(x)$ $(\cdots)$, $\tilde{g}_i(x) + \alpha_4\gamma_4(x)$ (– · –) and $\tilde{g}_i(x) + \alpha_4\gamma_4(x) + \alpha_5\gamma_5(x)$ (– –, too close to the original function to be seen).

Table 2
Number of equidistant nodes needed to achieve a maximum absolute error of at most 0.01 kJ/mol for different interpolation methods applied to a Lennard–Jones potential with $\sigma_{AB=}$0.3 nm and $\varepsilon_{AB} = 1$ kJ/mol for $r_{ij} \in [\sigma_{AB}, 1]$ when interpolating over r_{ij} or r_{ij}^2.

Method	$x = r_{ij}^2$	$x = r_{ij}$
Andrea et al.	28	12
Allen and Tildesley	116	50
GROMACS	244	110
MD-GRAPE	40	18
Bowers et al.	61	27
This paper	17	8

The use of $x = ar_{ij} + b$ instead of $x = ar_{ij}^2 + b$ is motivated by the observation that if we represent a continuous and smooth function $f(x)$ by an interpolation of degree m, the interpolation error in each interval will be proportional in magnitude to the derivative $f^{(m+1)}(\xi)$ for some ξ inside the interval. In Fig. 3 we plot the fourth derivative of a Lennard–Jones potential with $\sigma_{AB} = 0.3$ nm and $\varepsilon_{AB} = 1$ kJ/mol interpolated in r_{ij} and in r_{ij}^2. The derivatives for the interpolation using r_{ij}^2 are larger and hence more (i.e. smaller) intervals will be necessary to achieve the same precision.

Table 2 shows the number of equidistant nodes needed to achieve a maximum absolute error of at most 0.01 kJ/mol for each of the interpolation methods discussed in Table 1 applied to the same Lennard–Jones potential as in the previous example when interpolating over r_{ij} or r_{ij}^2. We conclude that if we compute the interpolation over r_{ij} directly, we will need roughly half as many intervals for the same accuracy.

4. Distribution of the nodes x_i

For most interactions, the potential function tends to become less and less interesting with increasing r_{ij}, i.e. the potential flattens out, and less nodes would be needed for its accurate interpolation. This was already noted by Hunt and Cohen in [15] who presented different interval distribution schemes, yet for a piecewise constant potential, and by Berendsen, who presents an *ad-hoc* non-uniform distribution in [16, p. 542] in which the interval widths decrease for small values of r_{ij}.

The interval distribution should allow for a fast identification of the index i such that $x_i \leqslant x < x_{i+1}$ (Line 4 in Algorithm 2). Andrea et al. use an unspecified variable interval spacing in which the index i could be found using a binary search. This, however, would not be especially advantageous if we are trying to avoid conditional expressions. The other methods use, where specified, a uniform node distribution. If we assume that $x = ar_{ij} + b$ is in the range $[0, 1]$, we can compute the nodes as

$$x_i = i/n, \quad i = 0 \ldots n$$

The index i can be easily evaluated for any x as

$$i = \lfloor x/n \rfloor$$

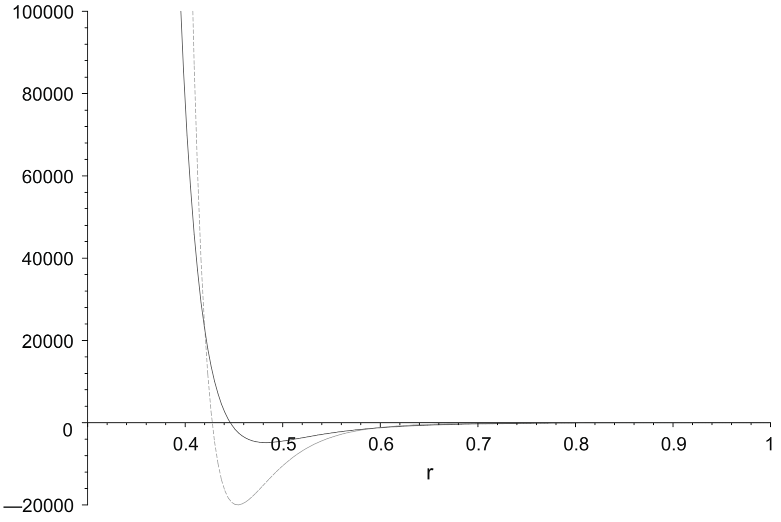


Fig. 3. Fourth derivative of the Lennard–Jones potential with $\sigma_{AB} = 0.3$ nm and $\varepsilon_{AB} = 1$ kJ/mol computed over r_{ij} (solid) and r_{ij}^2 (dashed).

We could, however, use any function of x monotonous for $x \in [0,1]$ to generate the indices. In the following, we will use the form

$$i = \lfloor n(\alpha x + (1-\alpha)x^2) \rfloor \tag{7}$$

which generates indices in the range [0,n] for $x \in [0,1]$ and is monotonic for $\alpha \in [0,2]$ (see Fig. 4). Although any monotonic function would work, we chose this form since it is both parameterizable and relatively simple to evaluate. Note that for $\alpha = 1$ we recover the uniform node distribution and for $\alpha = 2$ we recover the node distribution suggested by Berendsen.

The interpolation nodes can be pre-computed as

$$x_i = \frac{\alpha - \sqrt{\alpha^2 + 4(i/n)(1-\alpha)}}{2(\alpha - 1)}.$$

We now want to compute the optimal value for the parameter α. Given the interval distribution in Eq. (7), the width $h(x)$ of the interval enclosing any value of x is proportional to the inverse of the first derivative of Eq. (7) with respect to x:

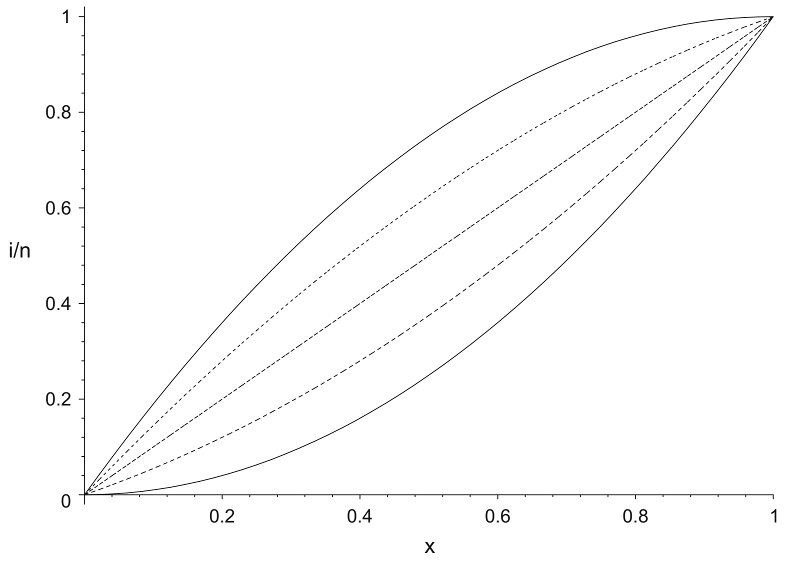


Fig. 4. Different node distributions for different values of α in Eq. (7). From bottom to top, $\alpha = 0, 0.5, 1, 1.5$ and 2.

Table 3
Interactions for the SPC/E water model using a particle-mesh Ewald scheme to compute the long-range electrostatics, where $q_O = -08476$ e, $q_H = +0.4238$ e, $\varepsilon_{OO} = 0.6502$ kJ/mol, $\sigma_{OO} = 0.3166$ nm and $\kappa = 3.0$.

Interaction	Range	$v_{AB}(r_{ij})$
OO	$[0.3, 1]$	$\frac{q_O^2}{r_{ij}}\operatorname{erfc}(\kappa r_{ij}) + 4\varepsilon_{OO}\left(\left(\frac{\sigma_{OO}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{OO}}{r_{ij}}\right)^{6}\right)$
OH	$[0.15, 1]$	$\frac{q_O q_H}{r_{ij}}\operatorname{erfc}(\kappa r_{ij})$
HH	$[0.17, 1]$	$\frac{q_H^2}{r_{ij}}\operatorname{erfc}(\kappa r_{ij})$

$$h(x) = [n(\alpha + 2(1-\alpha)x)]^{-1} \tag{8}$$

For a piecewise interpolation of degree m, we know that the interpolation error in each interval is proportional to

$$f(x) - g_i(x) \sim h(x)^{m+1} f^{(m+1)}(\xi_i), \quad x, \xi \in [x_i, x_{i+1}]$$

For $n \to \infty$ and thus $x_{i+1} - x_i \to 0$, this becomes

$$f(x) - g_i(x) \sim h(x)^{m+1} f^{(m+1)}(x).$$

In order to minimize the maximum interpolation error over all intervals, we choose $\alpha \in [0, 2]$ such that

$$\max_{x\in[0,1]} \left| h(x)^{m+1} f^{(m+1)}(x) \right| \tag{9}$$

is minimal. Since Eq. (9) is not easy to compute analytically, we compute the optimal $\alpha = \alpha_m$ numerically using Brent's golden section algorithm [17]. Note that the value α_m^* is optimal for the asymptotic limit $n \to \infty$ and that for small n, while still a good estimate, it may not be optimal.

We compute α_m^* for the interactions in Table 3 used for the SPC/E model [18] for liquid water simulations, using a particle-mesh Ewald scheme to compute the long-range electrostatics. A cutoff of 1 nm and a screening of $\kappa = 3.0$ were used. The number of intervals n required to achieve an error of at most 0.001 kJ/mol for the different interpolation methods are shown in Table 4. As can be seen from the results, the interpolation over the node distribution in Eq. (7) using the optimal α_m^* requires slightly more than half the number of nodes as the interpolation over the uniformly distributed nodes for the same accuracy.

Table 4
Number of equidistant nodes needed to achieve a maximum absolute error of at most 0.001 kJ/mol for different interpolation methods applied to the interactions in Table 3.

Method	$v_{OO}(r)$			$v_{OH}(r)$			$v_{HH}(r)$		
	$\alpha = 1$	$\alpha = \alpha_m^*$	α_m^*	$\alpha = 1$	$\alpha = \alpha_m^*$	α_m^*	$\alpha = 1$	$\alpha = \alpha_m^*$	α_m^*
Andrea et al.	20	11	1.9171	8	5	1.8061	6	4	1.7790
Allen and Tildesley	97	47	1.9537	32	16	1.8979	22	12	1.8817
GROMACS	410	210	1.9537	88	47	1.8979	51	27	1.8817
MD-GRAPE	36	19	1.9341	11	7	1.8458	8	5	1.8230
Bowers et al.	54	28	1.9537	17	9	1.8979	12	7	1.8817
This paper	13	7	1.9171	5	3	1.8061	4	2	1.7790

Table 5
Lookup table sizes in bytes and average number of CPU clock ticks required to evaluate the potential function $v_{OO}(r_{ij})$ using the interpolation schemes of Bowers et al., Andrea et al. and the error-minimizing polynomials described in Section 2, with and without the improvements suggested in Sections 3 and 4.

Method	$x = ar_{ij}^2 + b$				$x = ar_{ij} + b$			
	$\alpha = 1$		$\alpha = \alpha^*$		$\alpha = 1$		$\alpha = \alpha^*$	
Bowers et al.	2032 B	97.1 clk	1024 B	91.9 clk	864 B	84.3 clk	448 B	89.0 clk
Andrea et al.	1152 B	97.6 clk	600 B	100.2 clk	480 B	97.2 clk	264 B	102.6 clk
This paper	744 B	93.0 clk	384 B	98.4 clk	312 B	94.7 clk	168 B	101.0 clk

5. Numerical results

To assess the effects of the modification proposed in Sections 3 and 4, the construction and evaluation of the schemes of Bowers et al., Andrea et al. and the error minimizing polynomials described in Section 2 were implemented as per Algorithm 2 and tested.[8]

The test program generates the interpolation of the potential $v_{OO}(r_{ij})$ from Table 3 and evaluates it using double-precision arithmetic 10,000,000,000 times for random values of $r_{ij} \in [0.3, 1]$. The table sizes were selected such that the maximum absolute error of each interpolation scheme was below 0.001 kJ/mol, analogously to the results in Table 4. To simulate the effect of having several different atomic species, and therefore several different interaction potential tables, 50 tables (corresponding to roughly 10 distinct species) of the same potential $v_{OO}(r_{ij})$ were generated. For each evaluation one of these tables was chosen at random. To simulate the effect of competing memory access on the CPU cache, as it is present in any MD simulation, a buffer of 4,000,000 double-precision values was allocated. Between each evaluation of the potential, 20 random values were accessed from the buffer, corresponding to the loading and updating of particle data such as positions, forces and other particle attributes. Every 10,000,000 evaluations the entire buffer was traversed to re-set the cache, as would happen between two simulation time steps when all particle positions, velocities and forces are updated.

The test program was compiled using the Intel C++ Compiler 11.1 [19] and the individual timings (number of CPU clock ticks, clk) were collected using the Intel VTune Performance Analyzer 9.1 [20] on an Intel Core 2 Duo running at 2493 MHz and a 6144 kB L2-cache. Note that different architectures with different instruction latencies, cache sizes, cache speeds or simulations with different memory access patterns may lead to significantly different results.

The results of this comparison are shown in Table 5. As a point of reference, the direct computation of the interaction potential required on average 512.0 clk (of which 416.7 clk were required to evaluate erfc$(\cdot)$), almost five times the number of cycles required by the worst result using interpolations. As can be seen in the first column ($x = ar_{ij}^2 + b,\ \alpha = 1$) and between the last two rows, the smaller table sizes cause less cache misses, resulting in a faster interpolation evaluation, even despite the overhead caused by using higher-degree polynomials (requiring two additional iterations of the loop in Lines 7 and 10 of Algorithm 2) in the two bottom rows. This edge, however, is successively lost in the other columns as the tables become smaller and therefore exponentially less susceptible to be overwritten in the cache (note that even the largest tables fit comfortably in the cache for these tests).

The optimal n on-uniform intervals (columns 2 and 4 of Table 5) requires an additional multiplication and addition over using a uniform interval spacing.[9] On the architecture on which the tests were run, these costs were only amortized for the larger tables required by Bowers et al. using $x = ar_{ij}^2 + b$. Note, however, that on different architectures, i.e. different cache sizes,[10] memory speeds and memory access patterns, the tradeoff may be vary drastically.

The same tradeoff can be seen for the interpolations over $x = ar_{ij} + b$ as opposed to $x = ar_{ij}^2 + b$: The cost of evaluating $\sqrt{\cdot}$ is best amortized in Bowers et al.'s interpolation due to the large table size. For the other two interpolation schemes the tables are already sufficiently small for this setup, resulting in a slightly lower performance. Note that the tradeoff for this improvement relies heavily on how well the $\sqrt{\cdot}$ operation can be pipelined with other operations. In our tests the program did little other than evaluate the potential, leaving little room for such an optimization.

If table size is the sole criteria to be optimized, then the three improvements presented in Sections 2–4 can reduce the table size by a factor of 12 compared to Bowers et al. and 6.8 compared to Andrea et al. while increasing the computational cost by 4.0% and 3.5% respectively. On hardware implementations, where computing $\sqrt{\cdot}$ separately is not a promising option, the table sizes can be reduced by a factor of 5.3 at the cost of three additional multiplications and additions as compared to Bowers et al., or by a factor of 3.3 at the cost of one additional multiplication and addition as compared to Andrea et al..

6. Conclusions

In the previous sections we have presented three concepts — error-minimizing interpolation, interpolating over r_{ij} instead of r_{ij}^2 and the non-uniform distribution of the interpolation nodes — which improve the accuracy of piecewise polynomial interpolations of potential functions, thus reducing the size of the coefficient table required for its interpolation.

All three concepts can be applied to the previous methods mentioned herein and all three concepts improve on their accuracy, (e.g. less nodes required for the same accuracy). The corrections $\gamma_k(x)$ described in Eq. (4) can be constructed such as to preserve any of the constraints in the fourth column of Table 1 and can thus be applied to any of the interpolation therein. Furthermore, the construction of these interpolations depends neither on the interpolation variable (r_{ij} vs. r_{ij}^2) nor on the distribution of the intervals (uniform vs. non-uniform).

[8] The interpolations of Allen and Tildesley and GROMACS were not tested since they are of the same degree as Bowers et al.'s yet require more intervals for the same accuracy. The interpolation scheme of MD-GRAPE was not implemented since it is not sufficiently well described in the literature surrounding the project.

[9] In both cases the mapping from $r_{ij} \in [0.3, 1]$ or $r_{ij}^2 \in [0.09, 1]$ to $i \in [0, n]$ can be computed by evaluating a polynomial of degree two or three for the uniform and non-uniform case respectively.

[10] While the test system had a rather large L2-cache of 6144 kB, other systems may have significantly less: The CPUs used in the IBM Roadrunner [21], the PowerXCell 8i Processor and the AMD Opteron 2210, have 512 kB of L2-cache on the Cell/BE Power Processor Element and 256 kB local storage on the Cell/BE Synergistic Processing Elements and 2 MB of cache on each Opteron 2210 core.

As would be expected, the improved accuracy comes at a cost:

(i) the error-minimizing interpolation constructs piecewise polynomials of higher degree which are more costly to store and to evaluate;
(ii) the interpolation over r_{ij} requires the evaluation of a $\sqrt{\cdot}$ operation to compute r_{ij} and a reciprocal[11] to compute the force components $\mathbf{f}_{ij} = \frac{\mathbf{r}_{ij}}{r_{ij}} f_{ij}$;
(iii) the non-uniform distribution of the interpolation nodes requires additional computations to compute the interval index (Eq. (7));

which, depending on the actual implementation and the underlying hardware, could upset the cost/benefit ratio. The three concepts are, however, mutually orthogonal in the sense that they do not rely on each other and that their effects are cumulative. If the implementation of any of the single concepts does not result in an improvement, it can be left out without impacting the benefits of the others.

References

[1] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids, Oxford Science Publications, Clarendon Press, Oxford, 1987.
[2] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A.E. Mark, H.J.C. Berendsen, GROMACS: fast, flexible and free, Journal of Computational Chemistry 26 (2005) 1701–1718.
[3] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, K. Schulten, NAMD2: greater scalability for parallel molecular dynamics, Journal of Computational Physics 151 (1999) 283–312.
[4] IBM Systems and Technology Group, Cell Broadband Engine Programming Handbook, Including the PowerXCell 8i Processor, IBM Corporation, 2008.
[5] T.A. Andrea, W.C. Swope, H.C. Andersen, The role of long ranged forces in determining the structure and properties of liquid water, Journal of Chemical Physics 79 (9) (1983) 4576–4584.
[6] D. van der Spoel, E. Lindahl, B. Hess, A.R. van Buuren, E. Apol, P.J. Meulenhoff, D.P. Tieleman, A.L.T.M. Sijbers, K.A. Feenstra, R. van Drunen, H.J.C. Berendsen, GROMACS User Manual version 3.3, www.gromacs.org, 2005.
[7] M. Tiji, N. Tetsu, Y. Ohno, N. Futatsugi, A. Suenaga, N. Takada, A. Konagaya, Protein explorer: a petaflops special-purpose computer system for molecular dynamics simulations, in: Proceedings of the ACM/IEEE SC2003 Conference (SC'03), 2003, p. 15.
[8] K.J. Bowers, E. Chow, H. Xu, R.O. Dror, M.P. Eastwood, B.A. Gregersen, J.L. Klepeis, I. Kolossváry, M.A. Moraes, F.D. Sacerdoti, J.K. Salmon, Y. Shan, D.E. Shaw, Scalable algorithms for molecular dynamics simulations on commodity clusters, in: Proceedings of the ACM/IEEE Conference on Supercomputing (SC06), Tampa, Florida, 2006, p. 84.
[9] D.E. Shaw Research, Desmond User's Guide, September 1987.
[10] R.H. Larson, J.K. Salmon, R.O. Dror, M.M. Deneroff, C. Young, J. Grossman, Y. Shan, J.L. Klepeis, D.E. Shaw, High-throughput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation, in: Proceedings of the 14th Annual International Symposium on High-Performance Computer Architecture (HPCA '08), Salt Lake City, Utah, 2008, pp. 331–342.
[11] L. Fox, I.B. Parker, Chebyshev Polynomials in Numerical Analysis, Oxford University Press, London, 1968.
[12] C.W. Clenshaw, A note on the summation of Chebyshev series, Mathematical Tables and Other Aids to Computation 9 (51) (1955) 118–120.
[13] W.G. Horner, A new method of solving numerical equations of all orders, by continuous approximation, Philosophical Transactions of the Royal Society of London (1819) 308–335.
[14] J. Wetzel, E. Shiha, C. May, B. Frey, J. Furukawa, G. Frazier, PowerPC User Instruction Set Architecture, Book I, IBM, January 2005.
[15] N.G. Hunt, F.E. Cohen, Fast lookup tables for interatomic interactions, Journal of Computational Chemistry 17 (16) (1996) 1857–1862.
[16] H.J.C. Berendsen, Simulating the Physical World, Cambridge University Press, Cambridge, UK, 2007.
[17] R.P. Brent, Algorithms for Minimization Without Derivatives, Prentice-Hall, LinkEnglewood Cliffs, N.J., 1973.
[18] H.J.C. Berendsen, J.R. Grigera, T.P. Straatsma, The missing term in effective pair potentials, Journal of Physical Chemistry 91 (1987) 6269–6271.
[19] Intel C++ Compiler Professional Edition 11.1 for Linux, http://software.intel.com/en-us/intel-compilers/.
[20] Intel VTune Performance Analyzer 9.1 for Linux, http://software.intel.com/en-us/intel-vtune/.
[21] K.J. Barker, K. Davis, A. Hoisie, D.J. Kerbyson, M. Lang, S. Pakin, J.C. Sancho, Entering the petaflop era: the architecture and performance of roadrunner, in: SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE Press, Piscataway, NJ, USA, 2008, pp. 1–11.

[11] Note that on many architectures the $\sqrt{\cdot}$ operator actually computes the inverse square root. Both r_{ij} and r_{ij}^{-1} can thus be computed using only one such operation and an additional multiplication as $r_{ij}^{-1} \leftarrow 1/\sqrt{r_{ij}^2},\ r_{ij} \leftarrow r_{ij}^2 r_{ij}^{-1}$.