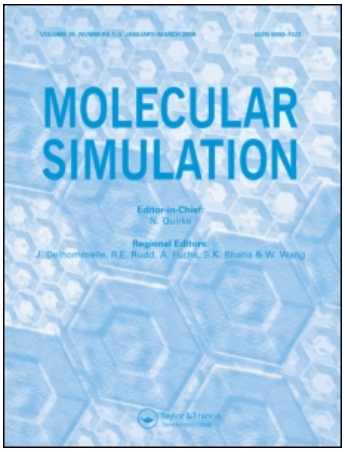

Molecular Simulation
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713644482

Molecular dynamics in arbitrary geometries: Parallel evaluation of pair forces

Graham B. Macpherson[a]; Jason M. Reese[a]
[a] Department of Mechanical Engineering, University of Strathclyde, Glasgow, UK

To cite this Article Macpherson, Graham B. and Reese, Jason M.(2008) 'Molecular dynamics in arbitrary geometries: Parallel evaluation of pair forces', Molecular Simulation, 34: 1, 97 — 115
To link to this Article: DOI: 10.1080/08927020801930554
URL: http://dx.doi.org/10.1080/08927020801930554

Molecular Simulation
Vol. 34, No. 1, January 2008, 97–115

Taylor & Francis
Taylor & Francis Group

Molecular dynamics in arbitrary geometries: Parallel evaluation of pair forces

Graham B. Macpherson* and Jason M. Reese

Department of Mechanical Engineering, University of Strathclyde, Glasgow G1 1XJ, UK

(*Received 14 August 2007; final version received 14 January 2008*)

A new algorithm for calculating intermolecular pair forces in molecular dynamics (MD) simulations on a distributed parallel computer is presented. The arbitrary interacting cells algorithm (AICA) is designed to operate on geometrical domains defined by an unstructured, arbitrary polyhedral mesh that has been spatially decomposed into irregular portions for parallelisation. It is intended for nano scale fluid mechanics simulation by MD in complex geometries, and to provide the MD component of a hybrid MD/continuum simulation. The spatial relationship of the cells of the mesh is calculated at the start of the simulation and only the molecules contained in cells that have part of their surface closer than the cut-off radius of the intermolecular pair potential are required to interact. AICA has been implemented in the open source C++ code OpenFOAM, and its accuracy has been indirectly verified against a published MD code. The same system simulated in serial and in parallel on 12 and 32 processors gives the same results. Performance tests show that there is an optimal number of cells in a mesh for maximum speed of calculating intermolecular forces, and that having a large number of empty cells in the mesh does not add a significant computational overhead.

Keywords: molecular dynamics; nano fluidics; hybrid simulation; intermolecular force calculation; parallel computing

PACS: 31.15.Qg; 47.11.Mn

1. Introduction and motivation

1.1 Molecular dynamics (MD) simulation in arbitrary geometries

Simulations of nano scale liquid systems can provide insight into many naturally occurring phenomena, such as the action of proteins that mediate water transport across biological cell membranes [1]. They may also facilitate the design of future nano devices and materials (e.g. high-throughput, highly selective filters or lab-on-a-chip components). The dynamics of these very small systems are dominated by surface interactions, due to their large surface area to volume ratios. However, these surface effects are often too complex and material-dependent to be treated by simple phenomenological parameters [2] or by adding 'equivalent' fluxes at the boundary [3]. Direct simulation of the fluid using MD presents an opportunity to model these phenomena with minimal simplifying assumptions.

MD fluid dynamics simulations have been reported [4–8], but MD is prohibitively computationally costly for simulations of systems beyond a few10s of nanometers in size. Fortunately, the molecular detail of the full flow-field that MD simulations provide is often unnecessary; in liquids, beyond 5–10 molecular diameters ($\lesssim 3$ nm for water) from a solid surface the continuum-fluid approximation is valid and the Navier–Stokes equations with bulk fluid properties may be used [2,9,10]. Hybrid simulations have been proposed [11–14] to simultaneously take advantage of the accuracy and detail provided by MD in the regions that require it, and the computational speed of continuum mechanics in the regions where it is applicable. An example application of this technique is shown schematically in Figure 1, where a complex molecule is being electrokinetically transported into a nanochannel for separation and identification [15]. Only the complex molecule, its immediately surrounding solvent molecules, and selected near-wall regions require an MD treatment; the remainder of the fluid (comprising the vast majority of the volume) may be simulated by continuum mechanics. A hybrid simulation would allow the effect of different complex molecules, solvent electrolyte composition, channel geometry, surface coatings and electric field strengths to be analysed at a realistic computational cost.

An alternative approach would be use one of several 'mesoscopic' simulation methods such as dissipative particle dynamics [16], lattice Boltzmann [17] or dynamic density functional theory [18]. These methods all require to be supplied with fluid–solid interaction boundary conditions and constitutive models, which it may not possible to accurately deduce in advance, and to which the behaviour of a nanoscale system is very sensitive.

*Corresponding author. Email: graham.macpherson@strath.ac.uk

ISSN 0892-7022 print/ISSN 1029-0435 online

DOI: 10.1080/08927020801930554
http://www.informaworld.com

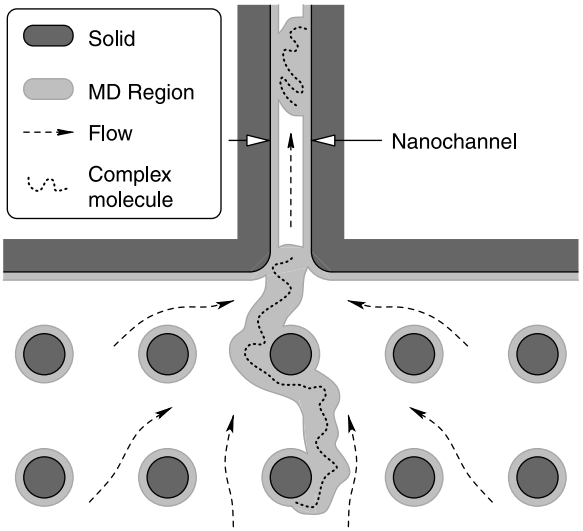

Figure 1. Schematic of an application of a hybrid MD/continuum simulation: complex molecules being transported into a nano channel. Only the complex molecules, regions near them and regions near solid surfaces need an MD treatment; the remaining volume can be simulated with continuum mechanics.

Providing that the intermolecular potentials used are accurate, MD avoids this problem by direct simulation.

In order to produce a useful, general simulation tool for hybrid simulations, the MD component must be able to model complex geometrical domains, for example see Figure 2. This capability does not exist in currently available MD codes: domains are simple shapes, usually with periodic boundaries. MD simulations are often performed to sample a system in a particular ensemble: a molecule's dynamics are not Newtonian and momentum is not necessarily conserved. For fluid mechanics the state of the molecules in the system should only be influenced at inlets, outlets and regions of solid walls not in contact with the fluid to impose the required boundary conditions via a physical body force (an electromagnetic field for example). The dynamics of the fluid will be purely Newtonian in the regions where results are gathered.

The most important, computationally demanding, and difficult aspect of any MD simulation is the calculation of intermolecular forces. This paper describes an algorithm that is capable of calculating pair force interactions in arbitrary, unstructured mesh geometries that have been parallelised for distributed computing by spatial decomposition. This work has been implemented in OpenFOAM [19], an open source C++ toolbox designed for computational fluid dynamics (CFD), and makes use of its particle tracking algorithm [20] for molecule dynamics and to efficiently maintain information about which mesh cell a molecule occupies.

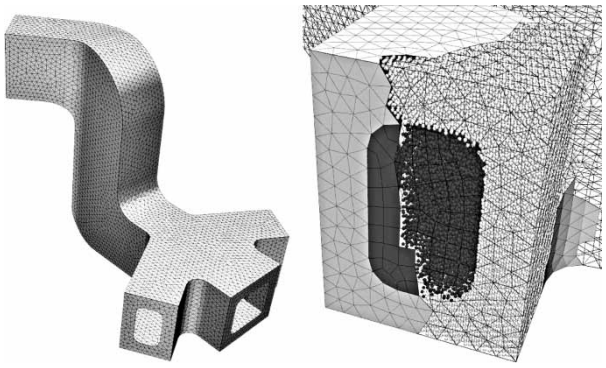

Figure 2. Left: A complex geometry test case. A three-inlet fluid mixing channel with a wall volume for explicit crystalline molecular walls. The overall height is 82 nm. The geometry was generated using Pro/ENGINEER® CAD software, exported to and meshed using GAMBIT®, then imported into OpenFOAM, filled with 2,167,110 molecules using the methods described in Ref. [21], and the MD simulation run using *gnemdFOAM* (see Section 3). Right: The mesh has been decomposed into 52 portions for parallel processing. The 48,148 molecules contained in one portion are shown. Two different types of molecule are used: wall molecules (light) are tethered in an FCC crystal and liquid molecules (dark) which are free to move.

1.2 Neighbour list limitations

The conventional method of MD force evaluation in distributed parallel computation is to use the cells algorithm to build neighbour lists for interacting pairs [22,23]. The *replicated molecule* method provides interactions across periodic boundaries and interprocessor boundaries, where the system has been spatially partitioned [24,25].

The spatial location of molecules in MD is dynamic, and hence not deducible from the data structure that contains them. A neighbour list defines which pairs of molecules are within a certain distance of each other and so need to interact via intermolecular forces. When considering systems where the geometry is defined by a mesh of unstructured, arbitrary polyhedral cells, that may have been divided into irregular and complex mesh segments for parallelisation, neighbour lists have two limitations.

(i) *Interprocessor molecule transfers*. A molecule may cross an interprocessor boundary at any point in time (even part of the way through a timestep), at which point it should be deleted from the processor it was on and an equivalent molecule created on the processor on the other side of the boundary. Given that neighbour lists are constructed as lists of array indices, references or pointers to the molecule's location in a data structure, deleting a molecule would invalidate this location and require searching to remove all mentions of it. Likewise, creating a molecule would require the appropriate new pair

interactions to be identified. Neither is practical due to the computational cost involved. It is conventional to allow molecules to stray outside of the domain controlled by a processor and carry out interprocessor transfers (deletions and creations) during the next neighbour list rebuild. This is straightforward when the spatial region associated with a processor can be simply defined by a function relating a position in space to a particular cell on a particular processor (i.e. a uniform, structured mesh, representing a simple domain). In a geometry where the space in question is defined by a collection of individual cells of arbitrary shape, this is more difficult. For example, the location the molecule has strayed to may be on the other side of a solid wall on the neighbour processor, or across another interprocessor boundary.

(ii) *Spatially resolved flow properties*. MD simulations used for flow studies must be able to spatially resolve fluid mechanical and thermodynamical fields. This is achieved by accumulating and averaging measurements of the properties of molecules in individual cells of the same mesh that defines the geometry. If a molecule is allowed to stray outside of the domain controlled by a processor, as above, then it would not be unambiguous and automatic which cell's measurement the molecule should contribute to.

Both of these problems could be mitigated by communicating with the neighbouring processor to determine which cell a molecule outside the domain should be in and sending its information, or alternatively maintaining a local copy of the appropriate thickness layer of geometry of the neighbouring processors. In this work, however, it has been deemed that either of these options would result in an inflexible arrangement, with each additional simulation feature requiring special treatment; so neighbour lists have not been used.

Neighbour lists have some unfavourable features that limit their computational speed in large and non-equilibrium simulations. If one region of the fluid has a high temperature or high velocity, then the high molecular velocities will cause the neighbour list to be invalidated and rebuilt often, limiting the calculation speed in the whole domain. Increasing the size and temperature of a system will also reduce the number of timesteps a neighbour list is valid for, thereby increasing the computational cost. This relationship can be predicted, see Appendix A.

We have developed a new algorithm, which is designed to either replace the neighbour list technique outright, or be used to construct the neighbour list, for simulations operating on meshes of unstructured arbitrary polyhedral cells.

2. Arbitrary interacting cells algorithm (AICA)

2.1 Interacting cell identification

The new AICA we propose here is a generalisation of the conventional cells algorithm (CCA) [22,23]. In the CCA, a simple (usually cuboid) simulation domain is subdivided into equally sized cells. The minimum dimension of the CCA cells must be greater than r_{cut}, the cut-off radius of the intermolecular pair potential, so that all molecules in a particular cell interact with all other molecules in their own cell and with those in their nearest neighbour cells (i.e. those they share a face, edge or vertex with – 26 in 3D). Extensions to this that permit the use of smaller cells [26–28] have been proposed, but they still require the domain to comprise a structured mesh of uniform cubes or cuboids. The objective of any cell algorithm is to evaluate interactions between as few molecules as possible that are further apart than r_{cut} to maximise computational speed.

AICA uses a 3D mesh of unstructured polyhedra, as would be used in CFD to define the geometry of a region. There are no restrictions on cell size, shape or connectivity. A cell's position in the mesh structure does not imply anything about its physical location or connectivity. Therefore, the ability to handle arbitrary geometries comes from deducing locally which cells are in the interaction range. Each cell has a unique list of other cells that it is to interact with, this list is known as the direct interaction list (DIL) for the cell in question (CIQ). It is constructed by searching the mesh to create a set of cells that have at least one part of their surface within a distance of r_{cut} from the surface of the CIQ, see Figure 3. Where there are multiple molecular species present with different values of r_{cut}, the maximum value is used to determine cell interactions. Where substantially different cut-off radii are present, it would be possible to have separate DILs for different range interactions, all of which could be constructed simultaneously using a single evaluation of the mesh.

This work has been carried out on the basis that the mesh is static. It is possible to use the same methods for a system with a dynamic mesh, providing that the information about which cells interact can be efficiently and reliably kept up-to-date, or rebuilt without incurring an unreasonable computational cost.

The DILs are established prior to the start of simulation and are valid throughout because the spatial relationship of the cells is fixed, whereas the set of molecules they contain is dynamic. In a similar way to the CCA, at every timestep a molecule in a particular cell calculates its interactions with the other molecules in that cell and consults the cell's DIL to find which other cells contain molecules it should interact with. Information is required to be maintained at all times stating which cell a molecule is in – this has been implemented in a robust

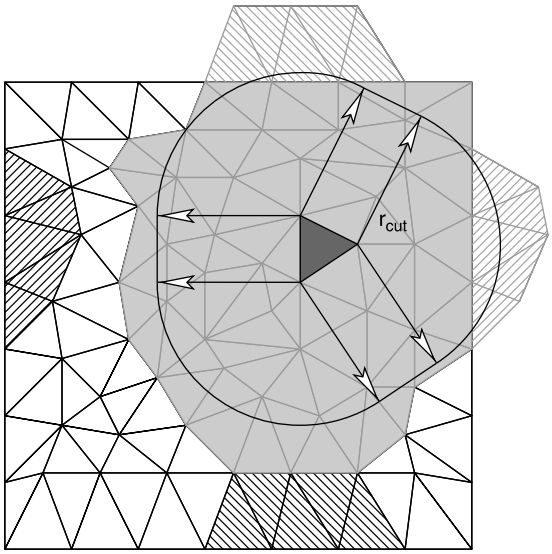

Figure 3. Interacting cell identification example using 2D unstructured triangular cells. The spatial domain (main square section comprising the real cells) is periodic top-bottom and left-right. Real cells within r_{cut} that interact with the CIQ (dark) are shaded in grey. The required referred cells are hatched in alternate directions according to which boundary they have been referred across. Realistic systems would be significantly larger compared to r_{cut} than shown here.

and efficient manner in OpenFOAM as part of the method for tracking particle motion [20].

2.2 Replicated molecule periodicity and parallelisation

When parallelising an MD simulation, the spatial domain is decomposed and each processor is given responsibility for a single region [22]. Molecules that cross the boundaries between these regions need to be communicated from one processor to the next. Processors also communicate when carrying out intermolecular force calculations, in which molecules close to processor boundaries need to be replicated on their neighbours to provide interactions. This process is illustrated in Figure 4. Periodic boundaries also require information about molecules that are not physically adjacent in the domain (see Figure 5); these required interactions can also be constructed by creating copies of molecules outside the boundary.

It is possible to handle processor and periodic boundaries in exactly the same way, because they have the same underlying objective: molecules near to the edge of a region need to be copied either between processors or to other locations on the same processor at every timestep to provide interactions. This is a useful feature because decomposing a mesh for parallelisation will often turn a periodic boundary into a processor

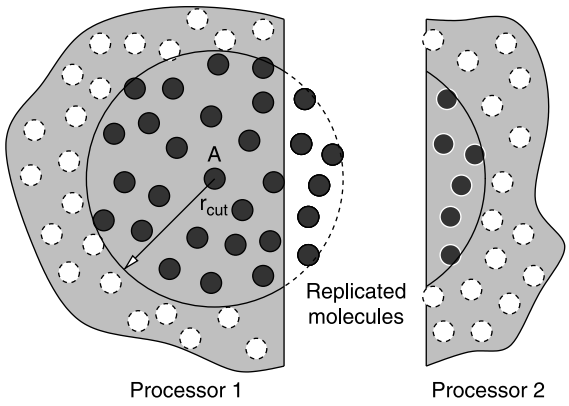

Figure 4. Spatial decomposition for parallel processing. Molecule A must calculate intermolecular forces with all other molecules within its cut-off radius, r_{cut}. When the domain has been decomposed, some of these molecules may lie on a different processor. In this case, copies of the appropriate molecules from processor 2 are made on processor 1.

boundary. The issue is: how to efficiently identify which molecules need to be copied, and to which location, because this set continually changes as the molecules move.

More general and flexible coupled boundaries can be implemented using the same framework, where molecules are replicated to either side of a spatially separated, non-parallel boundary. For example, a 'recycling' boundary condition or a rotationally symmetric simulation, see Figure 6, or coupling an inlet or outlet boundary or hybrid interface with any orientation to a static, external molecule reservoir.

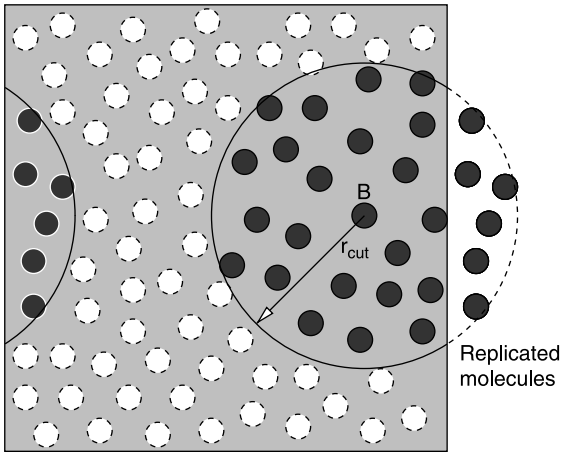

Figure 5. Periodic boundaries. Molecule B must calculate intermolecular forces with all other molecules within r_{cut}. Some of those molecules may be on a periodic image of the system, which, in computational terms, will reside on the other side of the domain. Serial calculations in simple geometries typically use the minimum image convention [22,23], but this is not suitable for parallelisation.

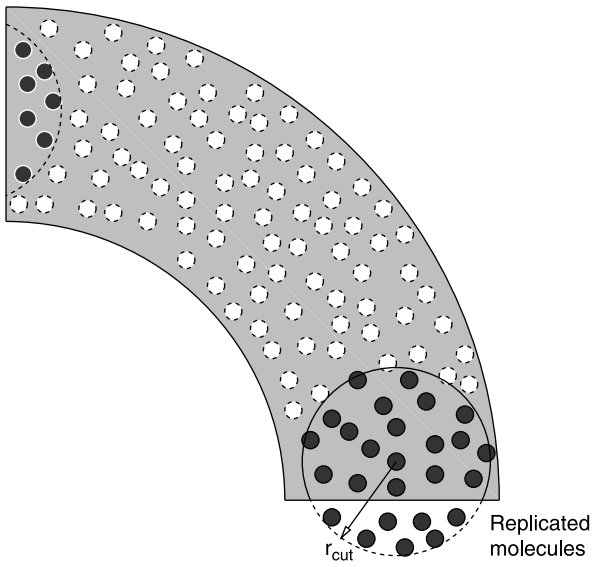

Figure 6. An example of a non-parallel, separated boundary. This could represent either a 90° bend in a channel where molecules are 'recycled' from inlet to outlet. It could also represent a cross-section through a hybrid simulation of flow in a pipe where the MD section is an outer annulus and rotational symmetry has been exploited to simulate only a quarter of the pipe – a typical CFD technique. In both cases, molecules near the separated boundaries must correctly exchange intermolecular forces.

2.2.1 *Referred molecules and cells*

Replicated molecule parallelisation and periodic boundaries are handled in the same way using referred cells (see Figure 3) and referred molecules.

Referred molecule. This is a copy of a real molecule that has been placed in a region outside a periodic or processor boundary in order to provide the correct intermolecular interaction with molecules inside the domain. A referred molecule holds only its own position and id (i.e. identification of which type of molecule it is for multi-species simulations). Referred molecules are created and discarded at each timestep, and do not report any information back to their source molecules. Therefore, if molecule j on processor 1 needs to interact with molecule k on processor 2, a separate referred molecule will be created on each processor.

Referred cell. Referred cells define a region of space and hold a collection of referred molecules. Each referred cell knows

- which real cell in the mesh (on which processor) is its source;
- the required transformation to refer the positions and orientations of the real molecules in the source cell to the referred location, see Appendix C. A general transformation of position and orientation is required for the cases mentioned in Section 2.2 and Figure 6, which require the replicated cells to have a different orientation to their source cell;
- the positions of all of its own vertices. These are the positions of the vertices of the source cell, which have been transformed by the same referring transform as the referred molecules it contains;
- the structure of all of the cell edges. Each edge is defined by a pair of indices in the referred vertex positions list indicating the vertices that comprise it;
- the structure of all of the cell faces. Each face is defined by a list of indices in the referred vertex positions list indicating the vertices that comprise it;
- which real cells are in range of this particular referred cell and hence require intermolecular interactions to be calculated. This is constructed once at the start of the simulation, in the same way as the DIL for real–real cell interactions.

2.3 *Interacting cell identification methods*

Building DILs and creating referred cells depends on identifying whether or not two cells are close enough such that the molecules they contain need to interact. Four methods for testing this were considered: PP, PPGR, CGAL and PFEE.

2.3.1 *Point–point (PP)*

The fastest and most straightforward method of determining whether cells are within r_{cut} is to use Point–Point (PP) searching. All of the N_{p} points in the mesh are compared to each other using a non-double-counting loop, and if any pair of points are within r_{cut} of each other, then all of the cells that these points form part of must all be in range of each other, see Algorithm 1. The information about which cells use a specific mesh point, and all subsequent information about relationship between the points, edges, faces and cells in the mesh exists in, and is readily accessible from, the mesh description in OpenFOAM. Note that a point is compared to itself ($p_j = p_i$ is used for the inner loop rather than $p_j = p_i + 1$) – this guarantees that neighbouring cells are

Algorithm 1. Point–Point cell searching.
for $p_i = 0$; $p_i < N_p$; $p_i + 1$ **do**
  for $p_j = p_i$; $p_j < N_p$; $p_j + 1$ **do**
    if the magnitude of the separation of the points referenced by p_i and $p_j \leq r_{\text{cut}}$ **then**
      all cells which p_j and p_i form part of must interact.
    end if
  end for
end for

identified for interaction if none of their non-shared vertices are within range of each other.

It is possible for PP to introduce errors, where molecules that should interact do not because their containing cells are not identified as being in range. Slices of cells may not be identified for interaction because

- a vertex may be within r_{cut} of a face of another cell, but not of any vertex. See Figure 7;
- the edges of two cells may be within r_{cut} of each other, but none of the vertices of either cell are within r_{cut} of a vertex, edge or face of the other. See Figure 8, this problem is only possible in 3D.

Both of these problems are most prevalent in meshes of tetrahedral cells, and cannot occur in regular meshes with cuboid or nearly cuboid cells.

2.3.2 *Point–point with guard radius (PPGR)*

The errors identified with the PP method can be reduced by adding a guard radius, r_G, to r_{cut}. PPGR works exactly as PP, except cells with vertices separated by $\leq r_G + r_{cut}$ now also interact. The guard radius will mean that many cells will have DILs that are larger than necessary, which will slow down the running of the simulation because more unnecessary molecule pairs will be evaluated at each timestep. It is possible to remove all errors from PP by adding a large enough guard radius, but this value is difficult to determine in advance, and in practice is relatively large, introducing a significant running performance penalty.

2.3.3 *Computational geometry algorithms library (CGAL)*

Whether cells need to interact can be found without either the errors of PP or the unnecessary additions to DILs of PPGR. The published CGAL library [29] can be used to calculate the closest distance between two convex hulls [30], constructed from the points of each cell. This method always finds all of the cells that need to interact, irrespective of their relative geometry, by using a single non-double-counting loop comparing all cells in the mesh to each other. When implemented, however, the computational cost proved thousands of times greater than any of the other three methods are described here. It is not suitable for meshes with a realistic number of cells, and will not be discussed further.

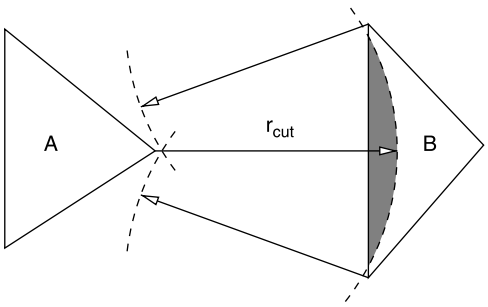

Figure 7. Errors caused by cut-off radius searching between vertices only. A sphere of radius r_{cut} drawn from the indicated vertex on cell A intersects a face of cell B, therefore, molecules near this vertex should interact with molecules in the shaded region. A sphere of radius r_{cut} drawn from either of the indicated vertices on cell B will not intersect any point of the surface of cell A, therefore; point-face searches are not reciprocal.

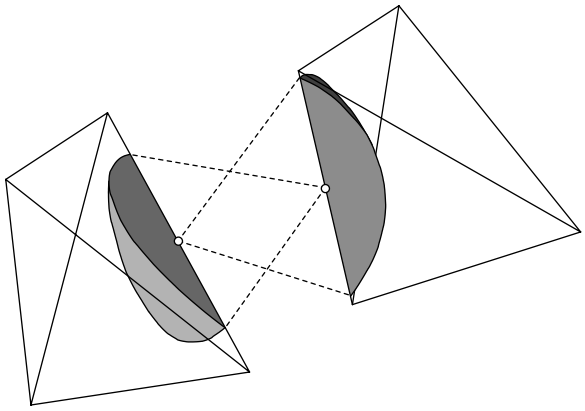

Figure 8. Two tetrahedral cells with edges within r_{cut} (dashed lines) of each other, but searching from vertices cannot establish this. The closest points between the two edges are marked, and the intersecting volumes on the other cell of a sphere of radius r_{cut} drawn from these points is shaded. Note that the straight edges of the shaded portions are perpendicular.

2.3.4 *Point–face and edge–edge (PFEE)*

All cells within r_{cut} of each other can be identified, without adding any unnecessary cells to a DIL, by conducting the mesh search on the basis of the two errors identified in PP: PFEE searching.

Point–face. The closest point on a face (which is a polygon with an arbitrary number of sides) to an arbitrary 3D point, **P**, can either be a face vertex, lie on an edge, or lie on the face itself. Figure 9 shows these three cases. The question of whether the point is within r_{cut} of the face can be determined by the following algorithm. The structure of this point-face algorithm is such that it only evaluates as much of the problem as is necessary to make a decision about whether the point is in range of the face. Once it is has this, it stops the evaluation of the current point-face pair and moves on to the next.

(i) Project **P** to the nearest point, $\mathbf{P_A}$ on the plane defined by the face centre, **C** and normal unit vector, **n**, see Figure 10, using

$$\mathbf{P_A} = \mathbf{P} - ((\mathbf{P} - \mathbf{C})\cdot\mathbf{n})\mathbf{n}. \quad (1)$$

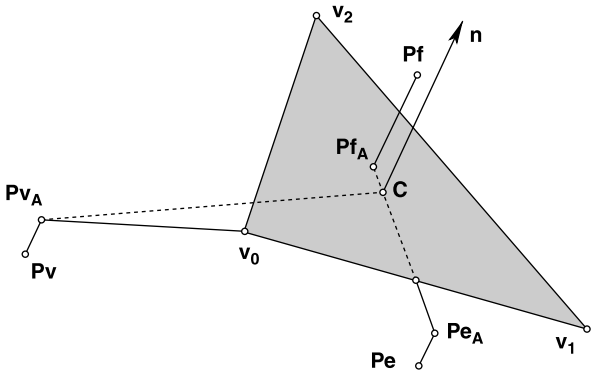

Figure 9. The three possibilities for the closest point on a face to an arbitrary point; the point is closest to a vertex, **Pv**, an edge, **Pe**, or the face itself, **Pf**. Points labelled with a subscript **A** are those projected onto the plane of the face.

If $|(\mathbf{P}-\mathbf{C})\cdot\mathbf{n}| > r_{\text{cut}}$ then it is not possible for the point to be in range of the face. The test actually performed is $|(\mathbf{P}-\mathbf{C})\cdot\mathbf{n}|^2 > r_{\text{cut}}^2$ to avoid the square root when determining the magnitude of the vector. All other vector magnitude comparisons are also performed in this way. No more calculation is necessary for this point-face pair.

(ii) Whether $\mathbf{P_A}$ lies inside or outside of the face must be determined. A line $\mathbf{P_AC}$ is drawn from $\mathbf{P_A}$ to $\mathbf{C}$, along the plane of the face and each edge of the face is tested to see if this line crosses it. To do this, all of the vertices, $\mathbf{v_0}, \mathbf{v_1} \ldots \mathbf{v_N}$, are projected onto a local 2D coordinate system that coincides with the plane of the face, with $\mathbf{P_A}$ as its origin, see Figure 11. This is necessary because $\mathbf{P_AC}$ and an edge may be skew if considered as 3D lines due to numerical round-off errors in the position of points of the face, or where a face is not perfectly flat [20]. The axis unit vectors of the coordinate system on the plane are

$$\mathbf{x}' = \frac{\mathbf{C}-\mathbf{P_A}}{|\mathbf{C}-\mathbf{P_A}|}, \tag{2}$$

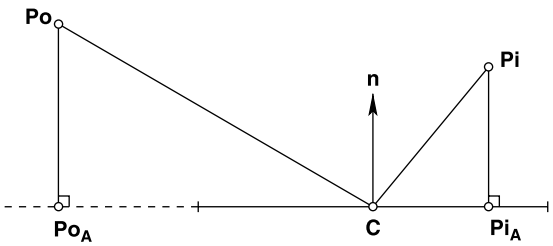

Figure 10. Projecting the point to be evaluated onto the plane defined by the face centre and normal unit vector. The projected point can either lie outside ($\mathbf{Po_A}$) or inside ($\mathbf{Pi_A}$) the face. The face is shown as a solid line and the extended plane as a dashed line.

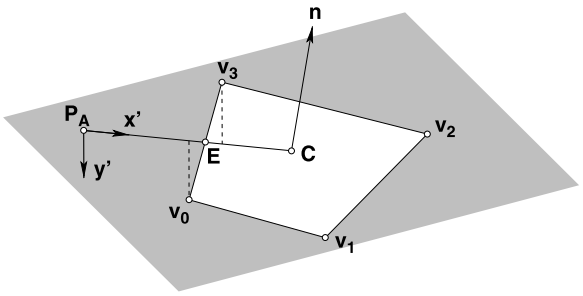

Figure 11. Defining a coordinate system local to the plane of the face to calculate intersections with face edges.

$$\mathbf{y}' = \frac{(\mathbf{C}-\mathbf{P_A})\times\mathbf{n}}{|(\mathbf{C}-\mathbf{P_A})\times\mathbf{n}|}, \tag{3}$$

and the position vector of a vertex, $\mathbf{v}_\alpha$, in plane coordinates is

$$\mathbf{v}'_\alpha = ((\mathbf{v}_\alpha-\mathbf{P_A})\cdot\mathbf{x}')\mathbf{x}' + ((\mathbf{v}_\alpha-\mathbf{P_A})\cdot\mathbf{y}')\mathbf{y}'. \tag{4}$$

Given that $\mathbf{C}$ lies on the line of $\mathbf{x}'$, by definition

$$\mathbf{C}' = |\mathbf{C}-\mathbf{P_A}|\mathbf{x}'. \tag{5}$$

When determining the local coordinates of the vertices, if $|\mathbf{P}-\mathbf{v}_\alpha| \le r_{\text{cut}}$ then the face must be in range of the point. No more calculation is necessary for this point-face pair.

Finding $\mathbf{E}$, the intersection point between $\mathbf{P_AC}$ and the edge defined by vertices $\mathbf{v}'_\alpha$ and $\mathbf{v}'_\beta$. The equations for $\mathbf{E}'$ (the intersection in plane coordinates) of the lines $\mathbf{P}'_\mathbf{A}\mathbf{C}'$ and $\mathbf{v}'_\alpha\mathbf{v}'_\beta$ are

$$\mathbf{E}' = \lambda_A\mathbf{C}', \tag{6}$$

$$\mathbf{E}' = \mathbf{v}'_\alpha + \lambda_v(\mathbf{v}'_\beta - \mathbf{v}'_\alpha). \tag{7}$$

Expanding Equations (7) and (8) in $\mathbf{x}'$ and $\mathbf{y}'$ components

$$\begin{aligned} E'_{x'} &= \lambda_A C'_{x'}, \quad E'_{y'} = \lambda_A C'_{y'}, \\ E'_{x'} &= v'_{\alpha x'} + \lambda_v(v'_{\beta x'} - v'_{\alpha x'}), \\ E'_{y'} &= v'_{\alpha y'} + \lambda_v(v'_{\beta y'} - v'_{\alpha y'}), \end{aligned}$$

and solving for λ_A and λ_v, given that $C'_{y'} = 0$ because $\mathbf{E}'$ lies on the line of $\mathbf{x}'$

$$\lambda_A = \frac{1}{C'_{x'}}\left(v'_{\alpha x'} - v'_{\alpha y'}\frac{(v'_{\beta x'} - v'_{\alpha x'})}{(v'_{\beta y'} - v'_{\alpha y'})}\right), \tag{8}$$

$$\lambda_v = -\frac{v'_{\alpha y'}}{v'_{\beta y'} - v'_{\alpha y'}}. \tag{9}$$

If $0 \le \lambda_A \le 1$ and $0 \le \lambda_v \le 1$, then the edge in question is crossed between $\mathbf{P_A}$ and $\mathbf{C}$. $\mathbf{P_A}$ must have

been outside of the face, and **E** is the closest point on the face to **P**,

$$\mathbf{E} = \mathbf{P_A} + \lambda_A(\mathbf{C} - \mathbf{P_A}). \tag{10}$$

If $|\mathbf{P} - \mathbf{E}| \leq r_{\text{cut}}$ then the face is in range of the point. If no edge is crossed by $\mathbf{P_AC}$, then $\mathbf{P_A}$ must have been on the face, so if $|\mathbf{P} - \mathbf{P_A}| \leq r_{\text{cut}}$ then the face is in range of the point, and the cells that the face forms part of must interact with the cells that the point forms part of.

The non-reciprocal nature of the point-face search must be borne in mind when searching for cells using it. When constructing DILs all faces must be tested with all points; no reduction in cost via a non-double-counting procedure is possible. Real cell points must search for referred cell faces and referred cell points must search for real cell faces. This, and the relatively complex algorithm for calculating the point-face distance, results in PFEE being computationally expensive.

Edge–edge. This uses the derivation of the closest distance between two skew lines [31] to determine whether two edges are within range of each other. The edges to be compared (see Figure 12) lie on lines that extend to infinity, the equations for the closest points on these lines are

$$\mathbf{c_1} = \mathbf{v}_\alpha + \lambda_1(\mathbf{v}_\beta - \mathbf{v}_\alpha), \tag{11}$$

$$\mathbf{c_2} = \mathbf{v}_\gamma + \lambda_2(\mathbf{v}_\delta - \mathbf{v}_\gamma). \tag{12}$$

Defining [32],

$$\mathbf{a} = \mathbf{v}_\beta - \mathbf{v}_\alpha, \quad \mathbf{b} = \mathbf{v}_\delta - \mathbf{v}_\gamma, \quad \mathbf{c} = \mathbf{v}_\gamma - \mathbf{v}_\alpha,$$

and solving Equations (12) and (13) for λ_1 and λ_2 gives

$$\lambda_1 = \frac{(\mathbf{c}\times\mathbf{b})\cdot(\mathbf{a}\times\mathbf{b})}{|\mathbf{a}\times\mathbf{b}|^2}, \tag{13}$$

$$\lambda_2 = \frac{(\mathbf{c}\times\mathbf{a})\cdot(\mathbf{a}\times\mathbf{b})}{|\mathbf{a}\times\mathbf{b}|^2}. \tag{14}$$

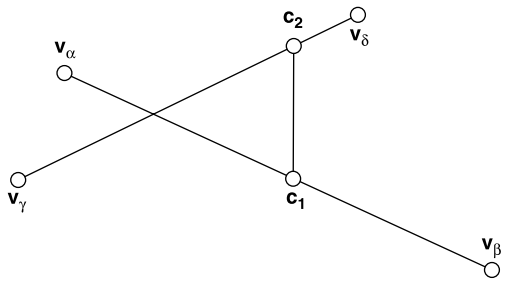

Figure 12. The closest distance between two edges.

If $|\mathbf{a}\times\mathbf{b}| = 0$, then the lines are parallel and this algorithm is invalid. A point-face search will, however, identify parallel edges as being in range if necessary. No more calculation is necessary for this edge pair.

If $0 \leq \lambda_1 \leq 1$ and $0 \leq \lambda_2 \leq 1$, then the closest points between the lines lie somewhere along both edges. In this case, calculate $\mathbf{c_1}$ and $\mathbf{c_2}$ using Equations (12) and (13) and test $|\mathbf{c_1} - \mathbf{c_2}| \leq r_{\text{cut}}$ to determine if the edges, hence the cells they form part of, are in range. Edge–edge searching is reciprocal and can be performed under a non-double-counting loop.

2.4 Intermolecular force calculation procedure

2.4.1 Build cell interactions

At the start of the simulation, the DIL for each real cell and appropriate referred cells are created, and the referred cells determine which real cells they must supply interactions to. Details of these processes can be found in Appendix B. Referred cells need not be sourced only from processors sharing a boundary, i.e. AICA is able to identify cell interactions between non-neighbouring processors. However, when constructing the referred cells, processors need only communicate across inter-processor faces, i.e. with neighbours only.

2.4.2 Build cell occupancy and refer molecules

At each timestep, before calculating any forces, a list for each cell is built stating which molecules it contains. This is a computationally cheap operation because it only involves querying each molecule for which cell it is in, and adding a reference or pointer to that molecule to the list for the appropriate cell. Each molecule holds information about which cell it occupies by virtue of the tracking mechanism [20].

At each timestep, all real cells which are the source cell of one or more referred cells send the position and id of all of the molecules they contain to the appropriate referred cell on the appropriate processor. The destination referred cells perform the appropriate position and orientation transformation when the molecules are received. The referred molecules are discarded and re-created from the source molecules at each timestep.

2.4.3 Force calculation

The total intermolecular force acting on a molecule is calculated at each timestep by considering two types of interactions. **Real–Real** interactions occur between a molecule and others on the same processor, according to Algorithm 2. **Real–Referred** interactions occur between real molecules and referred molecules arising from the

other side of a processor or periodic boundary, according to Algorithm 3. Referred molecules do not need to calculate interactions between themselves, because every referred molecule is a copy of a real molecule elsewhere, and as such will receive all of its real–real and real–referred interactions *in situ*.

Algorithm 2. Real–Real Molecule Force Calculation.

```
for all real cells, c_i do
  for all real molecules in c_i, m_i do
    for all real cells in c_i's DIL, c_j do
      for all real molecules in c_j, m_j do
        calculate intermolecular force f_ij between m_i and m_j
        add f_ij to f_i (total force vector for m_i)
        add f_ji=−f_ij to f_j
      end for
    end for
    for all real molecules in c_i with an index greater than*
        m_i, m_i' do
      calculate intermolecular force f_ii' between m_i and m_i'
      add f_ii' to f_i
      add −f_ii' to f_i'
      end for
    end for
  end for
```

*Comparison of the index of molecules in the same cell, $m_i' > m_i$, means that interactions between molecules in the same cell are not double-counted and a molecule does not calculate an interaction with itself. The order of comparison is not important, as long as all pairs are covered, so $m_i' < m_i$ would work equally well.

Algorithm 3. Real–Referred Molecule Force Calculation.

```
for all referred cells, c_q do
  for all referred molecules in c_q, m_q do
    for all real cells that c_q interacts with, c_p do
      for all real molecules in c_p, m_p do
        calculate intermolecular force f_pq between m_p and m_q
        add f_pq to f_p
      end for
    end for
  end for
end for
```

3. Implementation and performance

3.1 *gnemdFOAM*

AICA has been implemented in OpenFOAM [19], which is an open source C++ library intended for continuum mechanics simulation of user-defined physics (primarily used for CFD) in arbitrary, unstructured geometries. The AICA MD code has been built using OpenFOAM's lagrangian particle tracking library [20] and is called *gnemdFOAM*. All features of OpenFOAM have a common infrastructure for distributed memory parallel processing using MPI.

3.2 *Accuracy: Single timestep average pair potential energy*

The total potential energy of all pair interactions in a test system was calculated by an in-house C++ code, *gnemd*, which is based on, and is validated against, the code supplied in reference [22]. The positions of all of the molecules from the *gnemd* test case were imported into *gnemdFOAM*, transferring with a precision of 18 significant figures. The same system (same bounding box geometry and same intermolecular potential) was simulated by *gnemdFOAM* using a regular cubic mesh and 21 different tetrahedral meshes generated by commercial CFD meshing software GAMBIT®. The cubic mesh in *gnemdFOAM* comprised $28^3 = 21952$ cubic cells. The tetrahedral meshes were generated by specifying the number of cell edges to be placed on the edges of the bounding geometry, and allowing GAMBIT® to automatically generate a tetrahedral mesh. Edge numbers of 10–30 were used and the number of cells generated in each case shown in Table 1. An example mesh (edge number 15) is shown in Figure 13. Periodic boundaries in each direction were applied.

All MD values in these tests will be expressed in reduced units scaled using the Argon Lennard–Jones (LJ) potential length, energy and mass values.

Table 1. Number of cells in test tetrahedral meshes generated by GAMBIT®.

Edge no.	N_{cells}
10	7520
11	9748
12	11,101
13	19,344
14	22,289
15	22,338
16	28,113
17	36,646
18	43,420
19	50,877
20	56,171
21	67,520
22	73,801
23	88,334
24	97,488
25	105,385
26	129,761
27	145,016
28	145,196
29	185,016
30	196,847

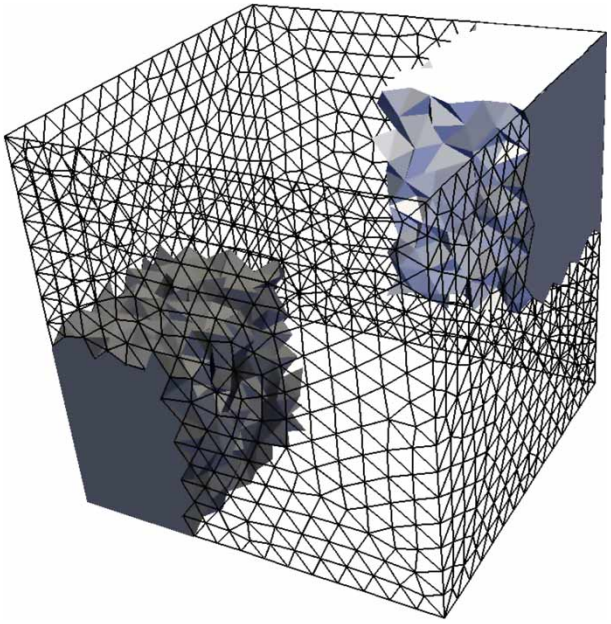

Figure 13. Tetrahedral mesh with edge number 15. The outline of cells on four of six faces of the bounding cube are shown and 2 of 12 portions of cells are shown from the mesh decomposed for parallelisation.

The shifted-force [23] LJ intermolecular potential for Argon was used in all simulations:

$$u(r_{ij}) = \begin{cases} 4\epsilon\left[\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right) - \left(\left(\frac{\sigma}{r_{\text{cut}}}\right)^{12} - \left(\frac{\sigma}{r_{\text{cut}}}\right)^{6}\right) + \frac{12 r_{\text{cut}}\left(r_{ij} - r_{\text{cut}}\right)}{\sigma^2}\left(\left(\frac{\sigma}{r_{\text{cut}}}\right)^{14} - \frac{1}{2}\left(\frac{\sigma}{r_{\text{cut}}}\right)^{8}\right)\right], & r_{ij} \le r_{\text{cut}} \\ 0, & r_{ij} > r_{\text{cut}}, \end{cases} \tag{15}$$

with $\epsilon = 120k_{\text{B}}$, where k_{B} is the Boltzmann constant, $\sigma = 0.34\,\text{nm}$ and $r_{\text{cut}} = 2.5\sigma$.

A cubic system domain of exactly 50 U side length was chosen to avoid any round-off errors when generating the mesh geometry, and a fluid number density of $(50/46)^{-3} = 0.778688$ chosen to give a perfect simple cubic lattice of $46^3 = 97336$ molecules. The total potential energy of all pair interactions for the molecules in their initial configuration, divided by the number of molecules, will be used to compare the results. This average potential per molecule, $\langle PE \rangle$, was calculated using *gnemd*, $\langle PE \rangle_g$, and by *gnemdFOAM* for 22 meshes (21 tetrahedral and cubic) using the PP, PPGR (with $r_{\text{G}} = 0.125$, 0.25, 0.5 and 1.0) and PFEE cell interaction identification methods. The cubic mesh in *gnemdFOAM* was not sensitive to any of the mesh searching errors of PP identified in Section 2.3.1 and produced exactly the same result, $\langle PE \rangle_{gF}$, with each of the cell interaction identification methods:

$$\begin{aligned} \langle PE \rangle_g &= -4.22656275761469, \\ \langle PE \rangle_{gF} &= -4.22656275759921, \\ \left|\frac{\langle PE \rangle_g - \langle PE \rangle_{gF}}{\langle PE \rangle_{gF}}\right| &= 3.64 \times 10^{-12}. \end{aligned} \tag{16}$$

The difference between $\langle PE \rangle_g$ and $\langle PE \rangle_{gF}$, Equation (17), most probably arises from the accumulation of rounding errors in the calculation of all pair potentials and any round-off error in the transfer of molecule positions between the two codes. The magnitude of this difference is not considered to indicate any algorithm or implementation errors. The relative error between $\langle PE \rangle$ for the tetrahedral meshes using the various cell interaction methods and $\langle PE \rangle_{gF}$ is calculated in the same manner as Equation (17) and shown in Figure 14. The errors in the PP or PPGR results are due to cells containing molecules that should interact not being identified by point–point searching of the mesh. The errors in PP and PPGR simulations become smaller in meshes with more cells because a fixed $r_{\text{cut}} + r_{\text{G}}$ length becomes larger relative to the cell size, giving a greater chance of establishing two points of a cell being in range.

Note that PFEE always produces a result that appears to be on the numerical 'noise floor' of the simulation –

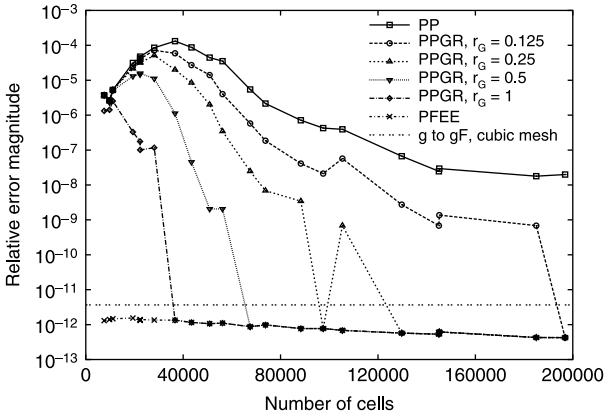

Figure 14. Relative error between $\langle PE \rangle$ for each tetrahedral mesh and $\langle PE \rangle_{gF}$ with different cell interaction build methods.

where the cumulative effect of numerical rounding errors becomes important. This is supported by the PPGR results dropping onto this line when r_G is large enough, indicating that all errors have been removed. This line has a smaller magnitude than the value from Equation (17), which is plotted as 'g to gF, cubic mesh'. This demonstrates that PFEE identifies the correct cells to provide every molecular interaction in the system, irrespective of mesh topology.

3.2.1 *Parallel accuracy*

The accuracy of PFEE when the simulation is parallelised is demonstrated in Figure 15. The tetrahedral meshes were decomposed into 12 portions (as shown in Figure 13) by a simple $x : y : z = 2 : 3 : 2$ split and into 32 portions using METIS [33], then simulated on a distributed memory parallel cluster. The 'parallel to serial error' data represents the relative difference between $\langle PE \rangle$ calculated for each mesh and $\langle PE \rangle_{gF}$. The 'parallel self-error' data represents the difference between $\langle PE \rangle$ calculated for each mesh and $\langle PE \rangle$ calculated for the cubic mesh parallelised in the same way as the tetrahedral meshes. It is interesting to note that the parallel self-errors are substantially lower than the parallel to serial errors. This is likely to be due to referred molecule positions being truncated from 64 to 32 bit precision by OpenFOAM to speed up inter-processor transfer. This would explain why the cubic mesh simulated in parallel gives a result substantially closer to the parallel tetrahedral meshes than the serial result. The parallel to serial errors are larger than those for PFEE in Figure 14, although still acceptably small. This demonstrates that the referred cells, which provide the intermolecular force connection between processors, are correctly created and used.

3.2.2 *Distribution of errors*

The values of $\langle PE \rangle$ for PP and PPGR do not give an indication of how the intermolecular force errors due to missed interactions are distributed. If a large number of molecules experience small errors, then this could be tolerable. If, however, a small number of molecules receive substantial errors, this will may cause unacceptable perturbations to the dynamics. The latter case is observed in practice.

The system simulated above is initialised in the cubic mesh with molecule velocities drawn from a Maxwellian distribution at a temperature of 2.5. The system is allowed to evolve in *gnemdFOAM* using the leapfrog [22] integration scheme for 1000 timesteps of $\Delta t = 0.005$ to create a reference system. This was done to let the initial simple cubic lattice relax into a more realistic configuration, because the net force on each molecule in the perfect crystal is zero, which is not useful for calculating relative errors. The molecule positions from the reference system are used in the tetrahedral meshes and the acceleration vector for each molecule calculated. The absolute and relative error magnitude between the acceleration for each molecule in the tetrahedral mesh and the acceleration for the corresponding molecule in the reference system is calculated:

$$\text{absolute}_i = |\mathbf{a}_{\mathbf{tet}_i} - \mathbf{a}_{\mathbf{ref}_i}|, \tag{17}$$

$$\text{relative}_i = \frac{|\mathbf{a}_{\mathbf{tet}_i} - \mathbf{a}_{\mathbf{ref}_i}|}{|\mathbf{a}_{\mathbf{ref}_i}|}. \tag{18}$$

These errors were added to histograms to assess their distribution. Examples of the histograms are shown in Figures 16 and 17 for the edge number 14 mesh, using

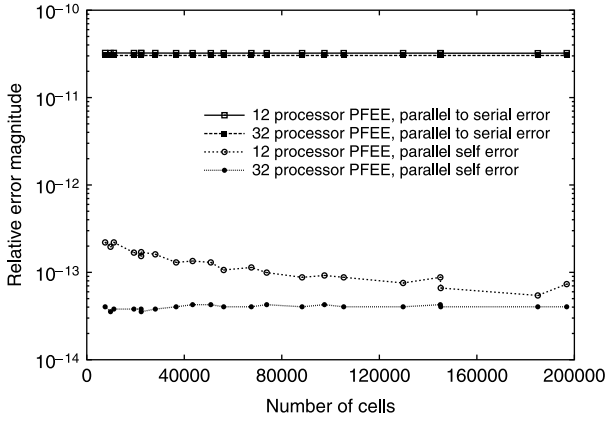

Figure 15. Relative error in $\langle PE \rangle$ for each tetrahedral mesh simulated in parallel using 12 and 32 processors. Cell interactions built with PFEE.

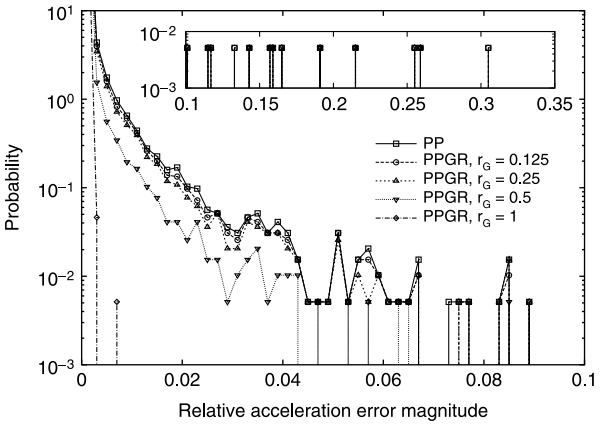

Figure 16. Relative error magnitude distribution for tetrahedral mesh, edge number 14.

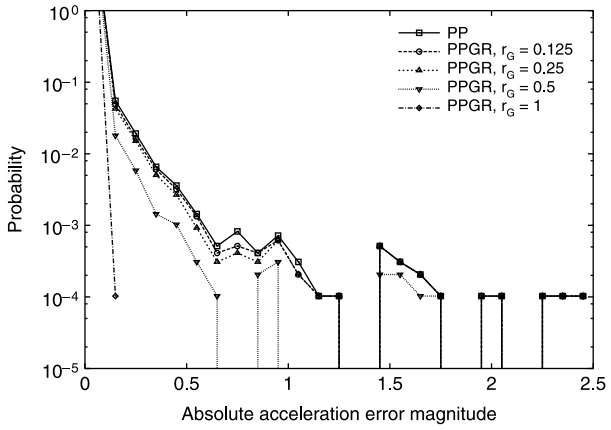

Figure 17. Absolute error magnitude distribution for tetrahedral mesh, edge number 14.

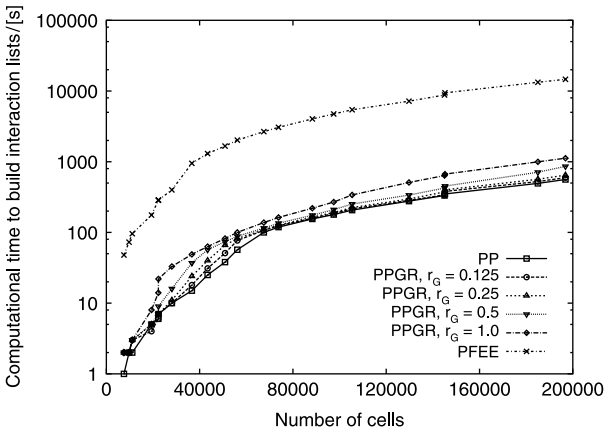

Figure 18. Time taken to build interaction lists at the start of the simulation.

PP and PPGR ($r_G = 0.125$, 0.25, 0.5 and 1.0). Both graphs show that the vast majority of molecules receive very small errors. A small number of molecules receive a substantial absolute error – up to 2.5, where acceleration magnitudes of up to 100–150 are typical in this simulation. Large relative errors of 30% (see the inset portion of Figure 16) are also seen, although these may be caused by the reference acceleration magnitude being very small. The addition of a guard radius does improve matters, but not substantially until, in this example, $r_G > 0.5$ is used.

3.3 Computational speed

The computational speed of establishing cell interactions and calculating intermolecular forces in *gnemdFOAM* was assessed for each of the tetrahedral meshes above, filled with 97336 LJ molecules. The time taken to build the interaction lists (create DILs, create referred cells and find real cells in range of referred cells) was recorded, and an intermolecular force calculation for all molecules performed 400 times and timed. The molecules were not moved during the simulation. All timing tests were performed on a PC with a 2.8 GHz, AMD Athlon™ FX-62 processor.

The time taken to build the interaction lists is shown in Figure 18 (note the logarithmic time axis). For the PP and PPGR data, an increase with increasing r_G is seen because more referred cells will be created. It is clear that PFEE takes substantially longer than PP or PPGR to build the interaction lists. This is understandable given the number of comparisons and calculations required by PFEE. Building the interaction lists is $O(N^2)$, for a mesh of N cells, and as such benefits greatly from parallelisation because each portion of the mesh only needs to search itself. The longest PFEE builds, taking several thousands of seconds, would typically not be practical and the mesh would be decomposed and the simulation parallelised. The interaction lists are established only once, at the start of the simulation, so for a very long simulation this may not represent a substantial portion of the total time. If the same mesh with the same r_{cut} is to be used repeatedly, then the cell interaction information can be calculated once then written to, and read from disk.

The average time taken per timestep to calculate all intermolecular forces is shown in Figure 19. All of the results follow a similar trend: in meshes with few cells, each cell contains a large number of molecules, and many pairs identified to interact will be further apart than r_{cut}. As the cells become smaller, the number of unnecessary interactions reduces until a point where the additional overhead of administrating more cells becomes more costly than the benefit derived, producing a 'dip'. All of the results essentially level off for meshes with a large number of cells (>80,000). This shows that the presence of empty cells (there are 97,336 molecules in the system) does not present a significant overhead;

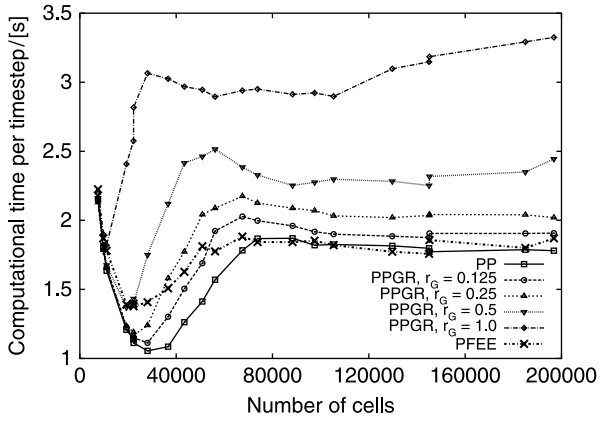

Figure 19. Time taken to calculate intermolecular pair forces between all molecules.

a favourable characteristic. The only exception is for PPGR, $r_G = 1.0$, where the large number of additional interacting cells seems to cause a noticeable overhead increase in meshes containing many cells.

The PP and PPGR results produce a family of curves, with increasing r_G the computational time increases and the region of the 'dip' narrows. An increased guard radius will produce DILs with unnecessary cells on them and unnecessary real–referred cell interactions, increasing the number of intermolecular evaluations between pairs of molecules that are further apart than r_{cut}.

The PFEE result has a similar form to PP and PPGR but does not fit into the family. In meshes with fewer cells, PFEE is more expensive than PP and most of the PPGR results because it is correctly identifying cells to interact that PP and PPGR miss. Comparing Figures 14–19 it can be seen that the 'dip' (up to approximately 40,000 cells) in computational time coincides with the region of maximum errors for PP and PPGR – missed cell interactions emphasise the 'dip' by reducing the number of calculations required by each molecule. The fact that PFEE does not take significantly longer to calculate forces than PP for finer meshes demonstrates that it does not identify any more cells to interact than are absolutely necessary.

There is an opportunity to trade-off the time taken to build the interaction lists using PFEE with a quick list build and slower timesteps using PPGR and a large enough r_G to remove all errors. The required r_G length is difficult to determine in advance, however, and most realistic simulations involve very many timesteps, which would soon make the PPGR simulation slower overall.

Further simulations to characterise the performance of the code, exploring the impact of molecule number density, system size, mesh cell size and morphology (for example, tetrahedral vs. hexahedral, skew and aspect ratio) number of processors and decomposition method are required.

4. Discussion and conclusions

The AICA algorithm is able to identify cells which are within a distance r_{cut} of each other in a mesh of unstructured arbitrary polyhedra. It can be used for building neighbour lists, or for intermolecular force calculations directly. In order to guarantee that every cell interaction is identified, particularly in tetrahedral meshes, a significant amount of calculation is required to identify when pairs of points and faces in the mesh are within range of each other, and when pairs of edges are within range. This is point-face, edge–edge (PFEE) searching of the mesh.

The accuracy of using PFEE mesh searching to determine which cells in the mesh should interact has been demonstrated by simulating a cubic MD system represented by 21 different tetrahedral meshes, ranging from 7520 to 196,847 cells. The results are the same, to within numerical round off error, to those produced by another verified MD code, *gnemd*. The computational cost of intermolecular force calculation with AICA depends on the mesh used but scales favourably with increasing refinement of the mesh. In particular, having a significant number of empty cells does not add much computational overhead. The accuracy of exchanging intermolecular force data across periodic and interprocessor boundaries using referred molecules and cells has been demonstrated, with the 21 tetrahedral meshes producing the same results when simulated in serial as on 12 and 32 processors.

4.1 Choosing a mesh

The performance of the algorithm is highly dependent on the mesh it is applied to. When choosing a mesh to use for a simulation the following points must be considered, and in some cases numerically tested and traded-off against one another:

- The time required to build cell interaction lists depends on the intermolecular potential cut-off radius, r_{cut}, the number of cells and their shape, and the form of the mesh: a tetrahedral mesh will have a difference balance of points:edges:cells compared to a hexahedral mesh, which will impact on the time taken for a PFEE search.
- Optimising the size of the mesh to minimise the time taken to run the simulation is complex and is case-specific, as demonstrated in Section 3.3.
- The mesh is not only used to define the shape of the domian and to help identify interacting molecules, thermodynamical and fluid mechanical fields are spatially resolved onto it and the mesh resolution may be dictated by this.
- The mesh does not need to be uniform, it can have locally high resolution for making measurements in regions of interest but be coarser to improve computational speed if necessary in other regions. The 'tricks of the trade' used in CFD can also be applied, for example the mesh can be adaptively refined to track a region of interest.
- The results in Section 3 showed that PP and PPGR are generally not good choices for building cell interactions unless either the mesh in question comprises regular hexahedra or some errors can be tolerated.

4.2 Future developments

Currently this algorithm deals only with short-ranged pair forces. Future developments will incorporate rigid and flexible polyatomic molecules, and long-range

electrostatic forces. A hybrid MD/continuum implementation in OpenFOAM is currently under development. OpenFOAM is ideal for incorporating polar fluids and electrokinetic actuation, which feature in envisioned applications, because it is able to simulate electromagnetic, magnetohydrodynamic and fluid mechanic continuum fields on the same mesh that AICA operates on.

Acknowledgements

The authors would like to thank Chris Greenshields and Matthew Borg of Strathclyde University, and Henry Weller and Mattijs Janssens of OpenCFD Ltd. for useful discussions. This work is funded in the UK by Strathclyde University, the Miller Foundation and the James Weir Foundation, and through a Philip Leverhulme Prize for JMR from the Leverhulme Trust.

References

[1] P. Agre, *The aquaporin water channels*, Proc. Am. Thorac. Soc. 3(1) (2006), pp. 5–13.
[2] J. Koplik and J.R. Banavar, *Continuum deductions from molecular hydrodynamics*, Annu. Rev. Fluid Mech. 27 (1995), pp. 257–292.
[3] H. Brenner and V. Ganesan, *Molecular wall effects: Are conditions at a boundary "boundary conditions"?* Phys. Rev. E 61(6) (2000), pp. 6879–6897.
[4] D. Hirshfeld and D.C. Rapaport, *Molecular dynamics simulation of Taylor–Couette vortex formation*, Phys. Rev. Lett. 80(24) (1998), pp. 5337–5340.
[5] D.C. Rapaport and E. Clementi, *Eddy formation in obstructed fluid flow: A molecular-dynamics study*, Phys. Rev. Lett. 57(6) (1986), pp. 695–698.
[6] K.P. Travis, B.D. Todd, and D.J. Evans, *Poiseuille flow of molecular fluids*, Physica A 240(1–2) (1997), pp. 315–327.
[7] Evans J., *Departure from Navier–Stokes hydrodynamics in confined liquids*, Phys. Rev. E 55(4) (1997), pp. 4288–4295.
[8] T. Qian and X.-P. Wang, *Driven cavity flow: From molecular dynamics to continuum hydrodynamics*, Multiscale Model. Simul. 3(4) (2005), pp. 749–763.
[9] T. Becker and F. Mugele, *Nanofluidics: Viscous dissipation in layered liquid films*, Phys. Rev. Lett. 91(16) (2003), p. 166104.
[10] H. Okumura and D.M. Heyes, *Comparisons between molecular dynamics and hydrodynamics treatment of nonstationary thermal processes in a liquid*, Phys. Rev. E 70(6) (2004), p. 061206.
[11] R. Delgado-Buscalioni and P.V. Coveney, *Hybrid molecular-continuum fluid dynamics*, Philos. Trans. R Soc. Lond. A 362 (1821) (2004), pp. 1639–1654.
[12] G. Wagner and E.G. Flekkøy, *Hybrid computations with flux exchange*, Philos. Trans. R Soc. Lond. A 362(1821) (2004), pp. 1655–1665.
[13] X.B. Nie et al., *A continuum and molecular dynamics hybrid method for micro- and nano-fluid flow*, J. Fluid Mech. 500 (2004), pp. 55–64.
[14] T. Werder, J.H. Walther, and P. Koumoutsakos, *Hybrid atomistic-continuum method for the simulation of dense fluid flows*, J. Comput. Phys. 205(1) (2005), pp. 373–390.
[15] S. Pennathur and J.G. Santiago, *Electrokinetic transport in nanochannels. 2. Experiments*, Anal. Chem. 77(21) (2005), pp. 6782–6789.
[16] P. Español and P. Warren, *Statistical mechanics of dissipative particle dynamics*, Europhys. Lett. 30(4) (1995), pp. 191–196.
[17] J. Horbach and S. Succi, *Lattice Boltzmann versus molecular dynamics simulation of nanoscale hydrodynamic flows*, Phys. Rev. Lett. 96(22) (2006), p. 224503.
[18] A.J. Archer, *Dynamical density functional theory for dense atomic liquids*, J. Phys. Condens. Matter 18(24) (2006), pp. 5617–5628.
[19] OpenFOAM: The Open Source CFD Toolbox. http://www.openfoam.org
[20] G.B. Macpherson, N. Nordin, and H.G. Weller, *Particle tracking in unstructured, arbitrary polyhedral meshes for use in CFD and molecular dynamics*, Under Rev. Commun. Num. Meth. Eng. in press (2008).
[21] G.B. Macpherson, M.K. Borg, and J.M. Reese, *Parallel generation of molecular dynamics initial configurations in arbitrary geometries*, Mol. Simul. 33(15) (2007), pp. 1199–1212.
[22] D.C. Rapaport, *The Art of Molecular Dynamics Simulation*, 2nd ed., Cambridge University Press, Cambridge, 2004.
[23] M.P. Allen and D.J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press, Oxford, 1987.
[24] W. Smith, *Molecular dynamics on hypercube parallel computers*, Comput. Phys. Commun. 62(2–3) (1991), pp. 229–248.
[25] D.C. Rapaport, *Multi-million particle molecular dynamics. II. Design considerations for distributed processing*, Comput. Phys. Commun. 62(2–3) (1991), pp. 217–228.
[26] T.N. Heinz and P.H. Hunenberger, *A fast pairlist-construction algorithm for molecular simulations under periodic boundary conditions*, J. Comput. Chem. 25(12) (2004), pp. 1474–1486.
[27] W. Mattson and B.M. Rice, *Near-neighbor calculations using a modified cell-linked list method*, Comput. Phys. Commun. 119(2–3) (1999), pp. 135–148.
[28] P. Gonnet, *A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations*, J. Comput. Chem. 28(2) (2007), pp. 570–573.
[29] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org
[30] K. Fischer et al., and the CGAL Editorial Board. Optimal distances, *CGAL User and Reference Manual*, 3.3 ed., 2007.
[31] W. Gellert et al., (eds.), *VNR Concise Encyclopedia of Mathematics*, 2nd ed., Van Nostrand Reinhold, New York, 1989.
[32] E.W. Weisstein, Line-line distance. From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/Line-LineDistance.html.
[33] G. Karypis and V. Kumar, METIS. A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Version 4.0. University of Minnesota, http://glaros.dtc.umn.edu/gkhome/views/metis.

Appendix A: Neighbour list lifetime prediction

The computational cost of the neighbour list algorithm depends on the number of timesteps a neighbour list remains valid for – a quantity whose dependence on simulation properties can be predicted. For a given number of molecules, N, of mass m, in equilibrium at a temperature T, there is a probability of $1/N$ that a molecule in the system will be travelling with a velocity v_N. It is therefore likely that at every timestep there will be one molecule travelling at, or close to this velocity. Given that a neighbour list is invalidated when the cumulative sum of *maximum* displacements in the system exceeds a fixed threshold (usually $0.5\Delta R$, half the shell thickness [22]), finding the number of timesteps required for a molecule travelling at v_N to cover this distance gives an estimate of the lifetime of the list. Estimating v_N by equating the Maxwellian velocity distribution to $1/N$, where T, m and v_N are in reduced MD units [23] (see Figure A1):

$$4\pi\left(\frac{m}{2\pi T}\right)^{\frac{3}{2}} v_N^2 \mathrm{e}^{-mv_N^2/2T} = \frac{1}{N}. \tag{A1}$$

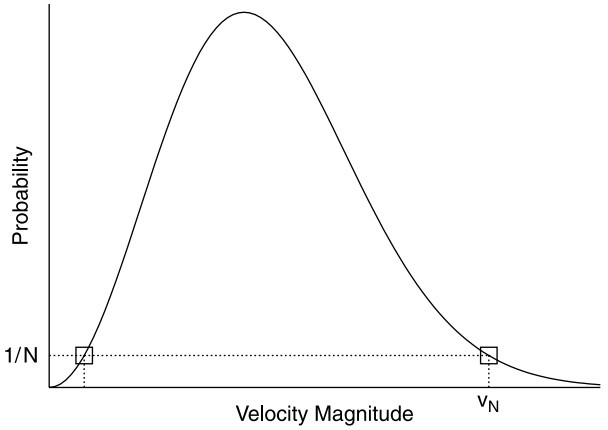

Figure A1. Maxwellian velocity distribution. Finding velocities corresponding to a probability of $1/N$, where N is the number of molecules in the system. There are two real, positive solutions – the high speed one is v_N, the quantity of interest.

The high speed solution is,

$$v_N(T,m,N) = \sqrt{-\frac{2T}{m} W_{-1}\left(-\frac{1}{4N}\sqrt{\frac{2\pi T}{m}}\right)}, \qquad \text{(A2)}$$

where W_{-1} is the secondary real branch of the Lambert W function.

The accuracy of v_N as a prediction of the maximum velocity in the system has been tested experimentally. Two systems containing 1728 and 13,824 LJ molecules, $m = 1$, at a number density of 0.8 were equilibrated to two different temperatures, $T = 1.0$ and $T = 2.5$, then simulated for 44,000 timesteps of $\Delta t = 0.005$. At each timestep the maximum velocity in the system was found and added to a histogram with a velocity bin width of 0.05. The histograms were then normalised to produce probability functions, their mean calculated, and the data was fitted to a curve of the form,

$$f_{v_N}(x) = p(x-s)^q \mathrm{e}^{-(x-s)/r}. \qquad \text{(A3)}$$

Figure A2 shows the maximum velocity probability functions with their mean and the corresponding value of $v_N(T,m,N)$. Table A1 shows the pertinent data and the curve fitting parameters for each result. The values of v_N and the distribution mean are close in all four cases, with v_N consistently higher (2.5–5.5% in these examples) but lying comfortably within the distribution. Using v_N will probably result in a slightly pessimistic estimate of the lifetime of a neighbour list.

The number of timesteps (of length Δt) a neighbour list is valid for, L, is given by

$$L = \left\lceil \frac{0.5\Delta R}{\Delta t v_N} \right\rceil. \qquad \text{(A4)}$$

This lifetime reduces with increasing temperature and number of molecules, as shown in Figure A3, resulting in a higher computational cost when using neighbour lists. At higher temperatures v_N is higher because the velocity distribution covers a higher molecular speed range. In systems comprising more molecules, it is more probable that at least one molecule will be travelling at a given speed in the upper region of the distribution, therefore v_N is also higher. The cost of an algorithm such as AICA, based only on cell interactions, is not sensitive to either of these parameters, therefore becomes increasingly attractive in large systems ($> 10^6$ molecules) where neighbour list lifetimes are lower. The use of the ceiling function in Equation (A4) reduces the impact of the difference between v_N and the measured mean of the maximum velocity probability function, resulting only in an alteration of the position of the 'steps' in the result.

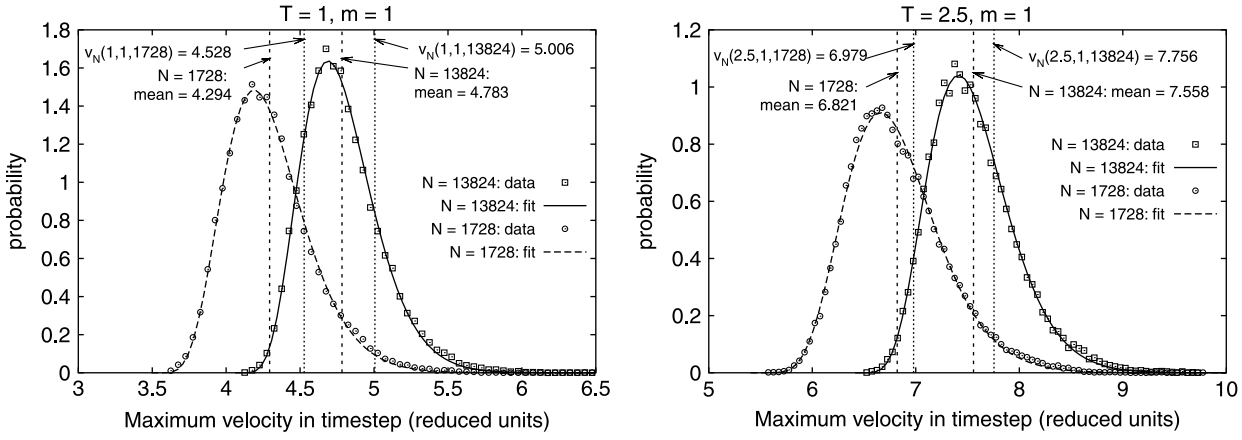

Figure A2. Probability of maximum velocity in simulation at $T = 1.0$ (left) and $T = 2.5$, (right) for 1728 and 13,824 Lennard–Jones molecules of mass $m = 1$ compared to $v_N(T,m,N)$. The data was collected from 44,000 timesteps of $\Delta t = 0.005$. The mean of the distributions is shown for comparison. Temperature, velocity and mass are expressed in reduced units.

Table A1. Experimental data and curve fitting parameters for $f_{\nu_N}(x)$.

T	N	ν_N	mean	$(\nu_N - \text{mean})/\text{mean}$	p	q	r	s
1	1728	4.528	4.294	5.45%	17310	6.794	0.1008	3.504
1	13824	5.006	4.783	4.66%	47060	7.027	0.08975	4.058
2.5	1728	6.979	6.821	2.31%	610.7	8.334	0.1494	5.416
2.5	13824	7.756	7.558	2.62%	1276	7.157	0.1406	6.406

Appendix B: Build cell interactions

Building DILs, creating referred cells and determining which real cells are in range of them is allowed to be computationally expensive (within reason) because it will only happen once at the beginning of the MD simulation, if the mesh is static. Accessing the information, however, must be as fast as possible because it will happen at every timestep. In each of these steps either the PP, PPGR or PFEE method can be used to test whether or not two cells are within r_{cut} of each other.

B.1 Building DILs

The construction of the DILs and accessing molecules is done in such a way as to not double-count interactions. For example, if cells A and B need to interact, cell B will be on cell A's DIL, but cell A will not be on cell B's DIL. When a molecule in cell A interacts with one in cell B the molecule in cell B will receive the inverse of the force vector calculated to be added to the molecule in cell A, because the pair forces are reciprocal, i.e. $\mathbf{f_{ab}} = -\mathbf{f_{ba}}$.

When using PP or PPGR, all points in the mesh are compared, as per Algorithm 1. When two points are in range, this produces a set of cells attached to one point and another set attached to the other point. When using PFEE, all points are compared to all faces. When this pair are in range the set of cells that are attached to the point and the set of cells that the face forms part of are produced. All edges in the mesh are compared in a non-double-counting loop, and when they are in range two sets of cells are again produced – those attached to one edge and those attached to the other.

For each step in each method, two sets of cells are returned. The index of each cell in one set is compared to the index of each cell in the other set, and the cell with the higher index is added to the DIL of the cell with the lower index. When the indices are equal, cells are not added to their own DIL.

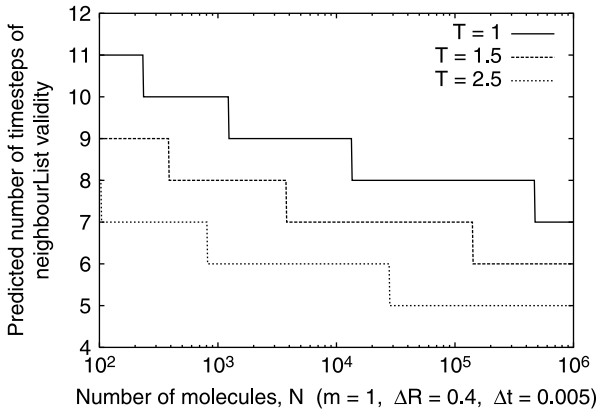

Figure A3. Variation of estimated neighbour list lifetime with typical simulation parameters for a Lennard–Jones fluid [22]. $T, m, \Delta R$ and Δt are in reduced MD units.

B.2 Creating referred cells

Coupled patches are the basis of periodic and interprocessor communication for creating referred cells. Patches, in general, are an OpenFOAM class comprising a collection of cell faces representing a mesh boundary of some description – they may provide solid surfaces, inlets, outlets, symmetry planes, periodic planes, or interprocessor connections. Coupled patches provide two surfaces; that which exits one enters the other. Two types are used in AICA:

- *Periodic patches* on a single processor are arranged into two halves, each half representing one of the coupled periodic surfaces. When a molecule crosses a face on one surface, it is wrapped around to the corresponding position on the corresponding face on the other surface.
- *Processor patches* provide links between portions of the mesh on different processors, one half of the patch is on each processor. When a molecule crosses a face on one surface, it is moved to the corresponding position on the corresponding face on the other surface, on the other processor. Decomposing a mesh for parallelisation will often require a periodic patch to be changed to a processor patch.

The surfaces of coupled patches may have any relative orientation, and may be spatially separated as long as the face pairs on each surface correspond to each other.

B.2.1 Creating patch segments

In the decomposed portion of the mesh on each processor, each processor and periodic patch should be split into segments, such that:

- faces on a processor patch that were internal to the mesh prior to decomposition end up on a segment;
- faces on a processor patch that were on separate periodic patches on the undecomposed mesh end up on different segments. These segments are further split such that faces that were on different halves of the periodic patch on the

undecomposed mesh end up on different segments;
- faces on different halves of a periodic patch end up on different segments.

Each segment must produce a single vector/tensor transformation pair (see Appendix C) which will be applied to all cells referred across it.

B.2.2 Cell referring iterations

For each patch segment:

(i) Find all real and existing referred cells with a portion of their surface in range of the faces comprising the patch segment.

(ii) Refer or re-refer this set of cells across the boundary defined by the patch segment using its transformation, see Appendix C. In the case of a segment of periodic boundary, this creates new referred cells on the same processor. For a segment of a processor patch, these cells are communicated to, and created on the neighbouring processor. Before creating any new referred cell a check is carried out to ensure that it is not

- a duplicate referred cell, one that has been created already by being referred across a different segment;
- a referred cell trying to be duplicated on top of a real cell, i.e. a cell being referred back on top of itself.

If the proposed cell for referral would create a duplicate of an existing referred cell, or end up on top of a real cell, then it is not created. To be a duplicate, the source processor, source cell and the vector part of the transformation (see Appendix C) must be the same for the two cells (note, the vector part of the transformation for a real cell is zero by definition).

A single run of these steps will usually not produce all of the required referred cells. They are repeated until no processor adds a referred cell in a complete evaluation of all segments, meaning all possible interactions are accounted for. In iterations after the first, in step (i) it is enough to search only for referred cells in range of the faces on the patch segment, because the real cells will not have changed and would all be duplicates. The final configuration of referred cells does not depend on the order of patch segment evaluation.

B.3 Determining real cells in range of referred cells

Once all of the referred cells have been created, each referred cell searches the mesh to determine which real cells have part of their surface within r_{cut} range of the surface of the referred cell, therefore need to be supplied with referred molecule interactions. The referred cell stores which real cells it needs to interact with. This process is accelerated by only searching a subset of the real cells in the mesh. These are the real cells that were identified as being in range of any processor or periodic patch when creating the referred cells. This set is guaranteed to contain all cells that could be in range of a referred cell.

B.4. Example construction of referred cells

Figures B1 and B2 show the start and end points of the cell referring operation on a mesh that has been decomposed to run in parallel. The example is in 2D for clarity but the process is exactly the same in 3D. In this example, the number of referred cells created exceeds the number of real cells, which would lead to much costly interprocessor communication. This is because the mesh has been made deliberately small relative to r_{cut} to demonstrate as many features of the algorithm as possible and to be practical to understand. In realistic systems, the mesh portions would be significantly bigger than r_{cut} and the referred cells would form a relatively thin halo around each portion. Decomposition of the mesh should preferably be carried out to minimise the number of cells that need to be referred and to ensure that the vast majority of the intermolecular interactions happen between real–real molecule pairs; in this way the communication cost is minimised.

Appendix C: Cell referring transformations

A spatial transformation is required to refer a cell across a periodic or processor boundary. Figure C1 shows the most general case of a cell being referred across a separated, non-parallel boundary, where,

α^0 = Cell with a face on one side of the boundary,
β^0 = Cell with a face on the other side of the boundary, coupled to α^0,
α^1 = Cell α^0 referred across the boundary,
$\mathbf{C}$ = Face centre,
$\mathbf{n}$ = Face normal unit vector,

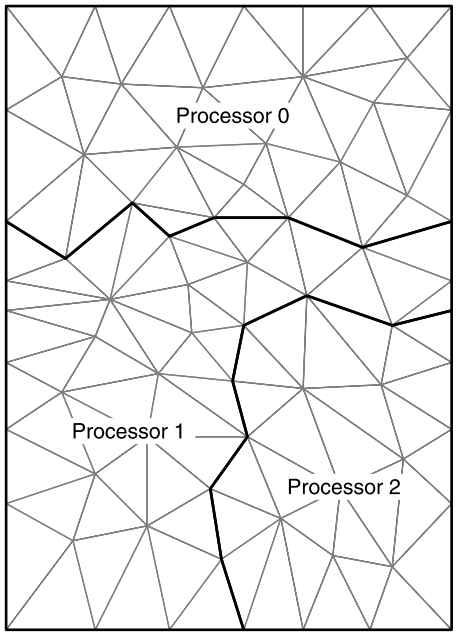

Figure B1. An arbitrary unstructured mesh, which is periodic top–bottom and left–right. It is decomposed into three irregular portions for parallel processing as marked.

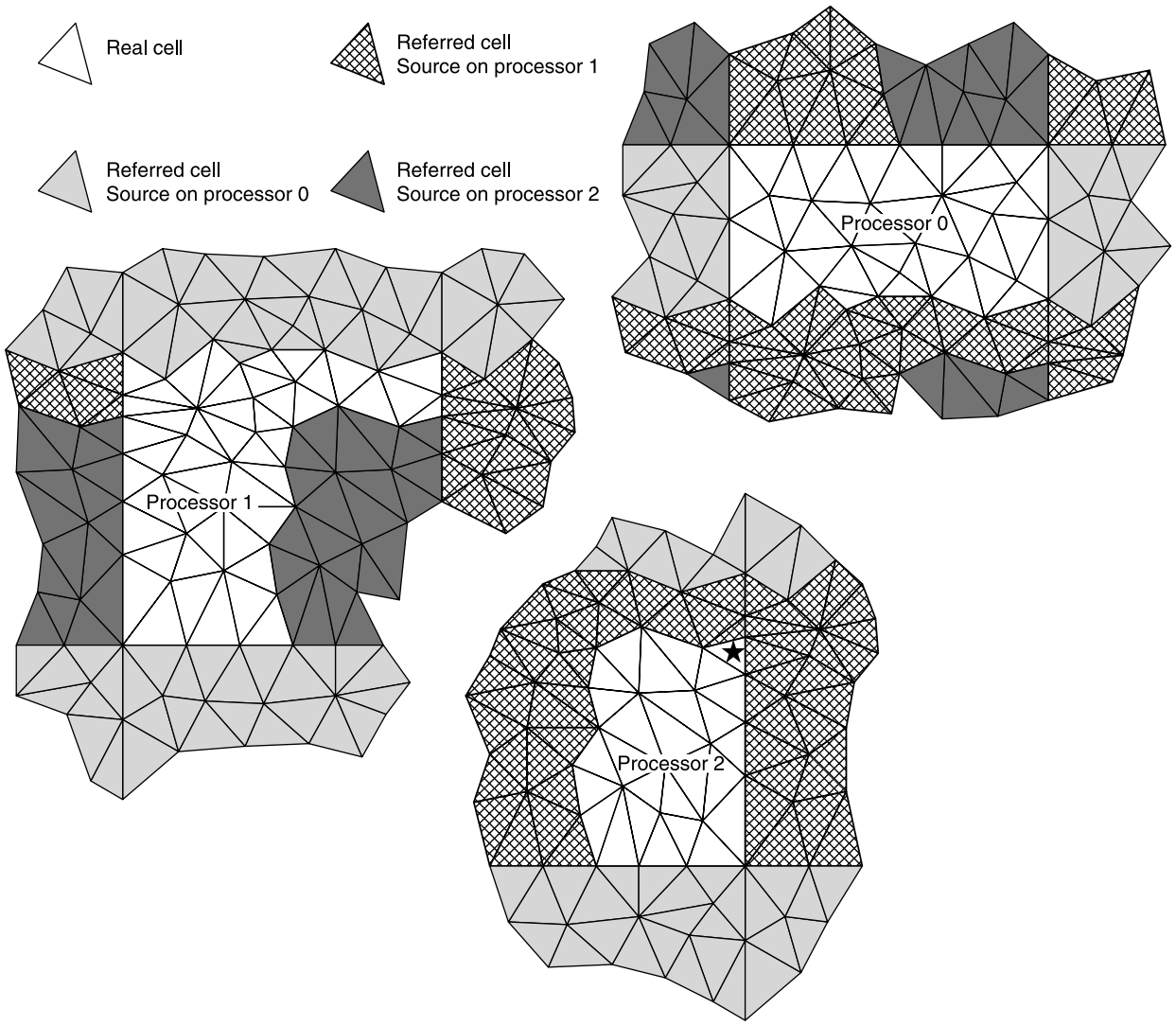

Figure B2. The final configuration of referred cells around the portions of the mesh in Figure B1 on each processor. A circle of radius r_{cut} drawn from any part of a real cell would be fully encompassed by other real or referred cells, thereby providing all molecules in that cell with the appropriate intermolecular interactions, either across a periodic boundary or from another processor. Note that many cells are referred several times; for example, the source cell marked with ★ on processor 2 is referred to processors 0 and 1 twice on each.

$\mathbf{S}_{\alpha^0\beta^0} = \mathbf{C}_{\beta^0} - \mathbf{C}_{\alpha^0}$. Position shift from $\mathbf{C}_{\alpha^0}$ to $\mathbf{C}_{\beta^0}$,
$\mathbf{R}_{\alpha^0\beta^0}$ = Tensor required to rotate $-\mathbf{n}_{\alpha^0}$ to $\mathbf{n}_{\beta^0}$, given by,

$$\mathbf{R}_{\alpha^0\beta^0} = -(\mathbf{n}_\alpha \cdot \mathbf{n}_\beta)\mathbf{I} + (1 + \mathbf{n}_\alpha \cdot \mathbf{n}_\beta) \times \left(\frac{\mathbf{n}_\alpha \times \mathbf{n}_\beta}{|\mathbf{n}_\alpha \times \mathbf{n}_\beta|}\right)^2 + \mathbf{n}_\alpha \mathbf{n}_\beta - \mathbf{n}_\beta \mathbf{n}_\alpha, \qquad \text{(C1)}$$

where $\mathbf{I}$ is the identity tensor. The absolute position, $\mathbf{P}'$, of a molecule transformed across a boundary (with its containing cell) from its original position $\mathbf{P}$ is required. To derive the $\mathbf{P} \rightarrow \mathbf{P}'$ transform, first the position of $\mathbf{P}$ relative to the centre of the coupled face on α^0 is given by

$$\mathbf{P} - \mathbf{C}_{\alpha^0}. \qquad \text{(C2)}$$

Then this vector is rotated by the transformation tensor defined by the source and destination coupled face normals:

$$\mathbf{R}_{\alpha^0\beta^0} \cdot (\mathbf{P} - \mathbf{C}_{\alpha^0}). \qquad \text{(C3)}$$

The rotated position is transformed back to global coordinates:

$$\mathbf{C}_{\alpha^0} + \mathbf{R}_{\alpha^0\beta^0} \cdot (\mathbf{P} - \mathbf{C}_{\alpha^0}), \qquad \text{(C4)}$$

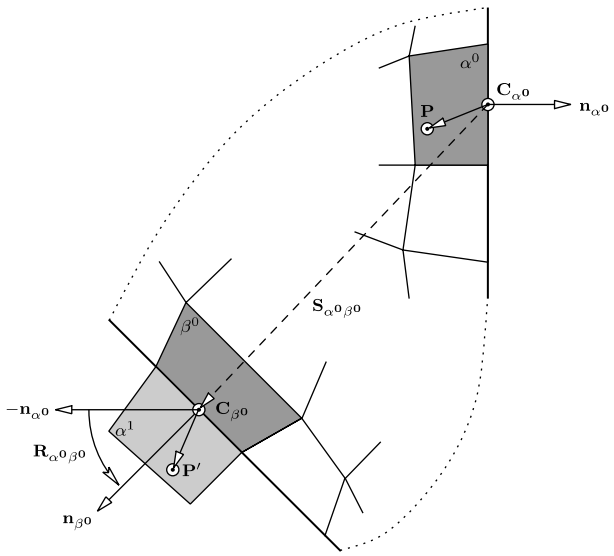

Figure C1. A molecule at point **P** is referred across the boundary (heavy line) to **P′** by a transformation defined by the face centre/face normal vectors of the faces of cells α^0 and β^0 on the boundary. The mesh is rigid, so all points of cell α^1 have the same relative spatial relationship as α^0. The vertices of referred cell α^1 are calculated by the same process.

and then shifted by the relative separation of the coupled face centres, i.e.

$$\begin{aligned}\mathbf{P}' &= \mathbf{C}_{\alpha^0} + \mathbf{S}_{\alpha^0\beta^0} + \mathbf{R}_{\alpha^0\beta^0}\cdot(\mathbf{P} - \mathbf{C}_{\alpha^0}),\\ &= \mathbf{C}_{\alpha^0} + \mathbf{C}_{\beta^0} - \mathbf{C}_{\alpha^0} + \mathbf{R}_{\alpha^0\beta^0}\cdot(\mathbf{P} - \mathbf{C}_{\alpha^0}),\\ &= \mathbf{C}_{\beta^0} - \mathbf{R}_{\alpha^0\beta^0}\cdot\mathbf{C}_{\alpha^0} + \mathbf{R}_{\alpha^0\beta^0}\cdot\mathbf{P}. \qquad \text{(C5)}\end{aligned}$$

The result is the same if the point is shifted by $\mathbf{S}_{\alpha^0\beta^0}$ prior to rotation by $\mathbf{R}_{\alpha^0\beta^0}$ around $\mathbf{C}_{\beta^0}$. The final position of **P′** is the same if any coupled face centre/face normal pair on the boundary is used. Therefore, all cells and all molecules referred across a particular boundary may use the transformation derived from one cell.

This result can be used to transform the positions of all of the molecules in a source cell to their correct position in a referred cell. Molecules being referred also operate on their own rotationally-dependent properties (e.g. angular orientation) using the rotation tensor.

This transformation is suitable for multiple periodic boundaries and arbitrary mesh decompositions where existing referred cells must be re-referred by other boundaries. This creates the cell relationships across edges, corners, and also on non-neighbouring processors. See Appendix B4 for an example. Cell re-referring is achieved by writing Equation (C5) as a generic transform:

$$\mathbf{P}' = \mathbf{y} + \mathbf{R}\cdot\mathbf{P}, \qquad \text{(C6)}$$

where

$$\mathbf{y} = \mathbf{C}_{\beta^0} - \mathbf{R}_{\alpha^0\beta^0}\cdot\mathbf{C}_{\alpha^0}, \qquad \text{(C7)}$$

$$\mathbf{R} = \mathbf{R}_{\alpha^0\beta^0}. \qquad \text{(C8)}$$

If **y** and **R** are the transformations required to move **P** to **P′**, then transforming **P′** to **P″** using **y′** and **R′** gives:

$$\begin{aligned}\mathbf{P}'' &= \mathbf{y}' + \mathbf{R}'\cdot\mathbf{P}',\\ &= \mathbf{y}' + \mathbf{R}'\cdot\mathbf{y} + \mathbf{R}'\cdot\mathbf{R}\cdot\mathbf{P},\\ &= \mathbf{y}'^{*} + \mathbf{R}'^{*}\cdot\mathbf{P}, \qquad \text{(C9)}\end{aligned}$$

and reduced to the generic form by defining,

$$\mathbf{y}'^{*} = \mathbf{y}' + \mathbf{R}'\cdot\mathbf{y}, \qquad \text{(C10)}$$

$$\mathbf{R}'^{*} = \mathbf{R}'\cdot\mathbf{R}. \qquad \text{(C11)}$$

Further re-referrals can be reduced to a single transform in a predictable way, with only a single vector/tensor pair required to be stored by the referred cell at any stage. Any number of cell referring transformations may be made sequentially, but the resulting transformation takes the initial source position (**P**) and directly transforms it to the final destination regardless of, and without having to intermediately visit, the other referred locations along the way.