
Article:
Law, Timothy R., Hancox, Jonny, Wright, Steven A. orcid.org/0000-0001-7133-8533 et al. (1 more author) (2019) An Algorithm for Computing Short-Range Forces in Molecular Dynamics Simulations with Non-Uniform Particle Densities. Journal of Parallel and Distributed Computing. pp. 1-11. ISSN 0743-7315

https://doi.org/10.1016/j.jpdc.2019.03.008

Journal of Parallel and Distributed Computing 130 (2019) 1–11

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

An algorithm for computing short-range forces in molecular dynamics simulations with non-uniform particle densities

T.R. Law [a,*], J. Hancox [b], S.A. Wright [c], S.A. Jarvis [a]

[a] Department of Computer Science, University of Warwick, Coventry, UK
[b] Health and Life Sciences Team, Intel Corporation, St. Clare House, London, UK
[c] Department of Computer Science, University of York, York, UK

HIGHLIGHTS

- We present the *projection sorting* algorithm for molecular dynamics simulations.
- We provide optimised implementations for Intel Broadwell and Knights Landing.
- We extend our implementations to multi-node environments using MPI.
- We investigate the performance of our algorithm in a biophysical MD simulation.
- We observe serial speedups up to 5×, and good MPI scaling behaviour.

ARTICLE INFO

Article history:
Received 15 September 2017
Received in revised form 5 September 2018
Accepted 14 March 2019
Available online 28 March 2019

Keywords:
Simulation
Molecular dynamics
Many-core
MPI
Algorithms
ARCHER

ABSTRACT

We present projection sorting, an algorithmic approach to determining pairwise short-range forces between particles in molecular dynamics simulations. We show it can be more effective than the standard approaches when particle density is non-uniform. We implement tuned versions of the algorithm in the context of a biophysical simulation of chromosome condensation, for the modern Intel Broadwell and Knights Landing architectures, across multiple nodes. We demonstrate up to 5× overall speedup and good scaling to large problem sizes and processor counts.

1. Introduction

Computational simulations are widely used across many scientific disciplines, spanning a variety of domains of investigation from crystalline atomic structures to cell pathways, with numerous software packages available. Of these, molecular dynamics (MD) simulations are some of the most well known. Prominent examples are LAMMPS [19], developed by Sandia National Laboratories to simulate materials under the influence of various physical potentials, and NAMD [18], which is more focused on biological applications and has been used to solve important recent medical problems, such as resolving the structure of the HIV-1 virus responsible for AIDS [26].

Simulations on this scale require enormous computational resources and the time is long since past where a single machine could provide the necessary power. In our current age of falling clock-speeds, exploiting the increasing amounts of available parallelism at all levels – from large networked clusters, down to optimal ordering of microarchitecture instructions – has become crucially important. Maturing many-core architectures such as Intel's Knights Landing (KNL) and NVIDIA's Pascal and Volta GPU architectures epitomise this philosophy of "going wide", but in turn demand much more from implementations in order to extract maximum performance. Significant work has gone into optimising MD applications for such architectures [3,9,17].

In this paper we build on our previous work [13] with a recent MD application designed to simulate chromosome condensation [6]. Specifically, we extend our previous work to run on new hardware and across large supercomputers, leveraging MPI for inter-node communications.

Our primary focus is the development and optimisation of an algorithm for pairwise short-range force calculations within MD applications which we call *projection sorting*. Owing to their computational expense and widespread applicability, pairwise short-range forces such as these have received significant attention in the MD literature. Specifically, we focus on MD simulations

* Corresponding author.
E-mail address: timothy.law@warwick.ac.uk (T.R. Law).

https://doi.org/10.1016/j.jpdc.2019.03.008

where the particles exhibit non-uniform density. Many MD simulations are designed to handle materials such as atomic lattices, or gases, which exhibit approximately constant particle density throughout the simulation domain. When simulating, for example, proteins or other macromolecules, they exhibit external structure and exist within an irregular environment. Algorithms designed for uniform densities can exhibit inefficiencies when applied in situations such as these.

In this paper, we make the following contributions:

- We present the *projection sorting* algorithm, an improvement to the computation of short-range interaction forces between particles under certain organisational conditions;
- We optimise our algorithm for modern architectures including Intel Broadwell and Intel Xeon Phi many-core architectures;
- We extend our implementation to multi-node enviro nments, enabling larger simulations to be computed;
- We investigate the performance of our algorithm in a modern biophysical MD simulation designed to investigate chromosome condensation on two platforms, one an Intel Xeon based platform, one based on Intel's Xeon Phi Knights Landing many-core architecture.

The remainder of this paper is structured as follows: Section 2 outlines related work in molecular dynamics simulations. Section 3 details our projection sorting algorithm. Section 4 describes our implementation of this idea in the context of the simulation from Cheng et al. [6], for both single and multi-node systems and subsequent performance investigations. Finally, Section 5 concludes this paper.

2. Background

MD is a computational method that uses simulation to study the dynamical behaviour of systems of particles (commonly atoms or molecules). It was developed in the 1950s by theoretical physicists looking to study such systems but lacking suitable analytical methods to do so [2,7]. Today, MD is used across a wide range of scientific fields, including chemical physics, materials science and biophysics.

Popular MD simulation packages include LAMMPS [19], NAMD [5,12,18,21], DL_POLY [22] and GROMACS [1]. Significant work has gone into optimising such applications for modern architectures [3,9,17].

2.1. Force calculations

MD simulations primarily involve computing forces between particles (based on factors such as position and velocity), and computing their trajectories through numerical integration. Forces to be computed typically fall into two categories: *short-* and *long-range*. Long-range forces are those that do not tend to zero over a finite distance, and therefore require inspection of all particles during calculation. Short-range forces on the other hand, do tend to zero over a finite distance. Only particles that fall within a limited "cut-off radius" (r_c) need be considered when calculating short-range forces. Short-range forces are often the most expensive components of MD simulations, therefore the cut-off radius and the particle density (ρ) within the simulation domain are two parameters that can greatly affect the overall performance. This paper is primarily concerned with the calculation of short-range forces. In particular, we focus on short-range forces based on "pair potentials". This simply means that the total force acting on a given particle is held equal to the sum of the forces acting on it due to each individual pairwise interaction. Most physical potentials are pair potentials, including Coulomb's law and Newton's law of gravitation.

2.1.1. Short-range force algorithms

The naïve approach to calculating pairwise short-range forces on a given particle is to iterate through all other particles, calculate the distances between each pair, and only apply forces for those within the cut-off radius. This takes quadratic time in the number of particles and is therefore not scalable to larger systems.

This can be accelerated by decomposing the simulation domain into disjoint uniform "cells". A list is maintained for each cell containing the particles resident. Particle-pair lookups are then restricted to a small neighbourhood of cells within the cut-off radius. This is called using "cell lists", or the "link-cell" approach [11,20].

As cells are cubic as opposed to the sphere indicated by the cut-off radius, cell lists still carry an inefficiency in the number of particle-pair checks required. This inefficiency increases as the cell size is increased. For example, when r_c is used, the immediate neighbourhood of 27 cells must be inspected. This has a volume of $(3r_c)^3$, which is nearly 6.5 times greater than the sphere volume of $\frac{4}{3}\pi r_c^3$. Assuming a uniform particle density, 84% of the particle-pair checks are unnecessary. While the inefficiency tends to zero with the cell size (see Fig. 1a), programming overheads will erase the benefit past a certain point, as the number of cells to be inspected increases cubically as the size is decreased. Gonnet discusses reducing the inefficiency by sorting the particles along the axis connecting cell centres [8], although this introduces its own overheads.

The situation can often be improved by using Verlet, or "neighbour", lists [23]. This approach was first developed by Loup Verlet in 1967. A list is maintained per particle, storing only the particles within a certain "Verlet radius" ($r_v \geq r_c$). Rather than building this list every timestep, each list is reused a number of times (k). Doing so requires knowledge of the maximum distance a particle can move in a single timestep, called the "skin distance" of the simulation (r_s). Then, the Verlet radius is set $r_v = r_c + kr_s$. It should be noted that Verlet lists and cell lists are not mutually exclusive; cell lists are often an efficient way of building Verlet lists. In this way, the cost of traversing the cell lists is amortised over k timesteps.

One disadvantage of Verlet lists is that the number of particle-pair checks is now tied to the speed at which particles move, which can be undesirable when this is a large, or highly variable, quantity. The rebuild period k should be chosen to strike a balance between the frequency of expensive list rebuilds, and the increased size of the lists. It is also clear that the viability of using Verlet lists depends heavily on the skin distance; it must be small relative to the cut-off radius or the number of spurious pairs will be too high. Fig. 1b shows how the fraction of spurious pairs increases rapidly with both k and r_s. Nevertheless, many modern MD codes, including LAMMPS [19] and NAMD [18], use Verlet lists.

2.2. Spatial locality

Another class of optimisations involves reordering stored particles such that those local to each other in three dimensional simulation space are also local in the computer's memory (a one dimensional space). Spatial locality is important in contemporary computer architectures, whose performance often depends on being able to work around memory latency by means of reuse within multiple layers of cache, and the ability to predict and prefetch data likely to be needed in the near future. Yao et al. discuss sorting particles along an axis of the simulation domain [25]. Anderson et al. demonstrate a successful application of a more sophisticated approach, whereby particles are ordered according to their distance along a space-filling Hilbert curve [3]. The Hilbert

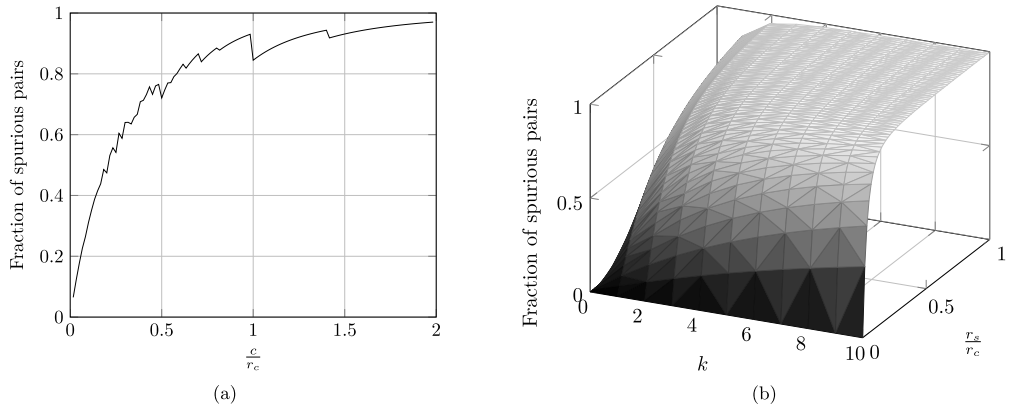

Fig. 1. (a) Cell lists – fraction of spurious pairs as a function of cell size for a fixed r_c (assuming a uniform particle density). (b) Verlet lists – fraction of spurious pairs as a function of rebuild period k and skin distance r_s for a fixed r_c (assuming a uniform particle density).

curve is chosen due to its locality preserving properties [15]. We would note that this reordering process serves an orthogonal purpose to the particle reordering we discuss in this paper. Space-filling curves are used to improve locality in computer memory, whereas projection sorting is used as a means to reduce the particle neighbour search space.

Contrary to the methods discussed in this section, where neighbouring particle pairs are determined explicitly by binning, we propose an algorithm that uses sorting to implicitly find neighbours. This has the advantageous side-effect of also ensuring spatial locality.

3. Projection sorting

In this paper we develop the *projection sorting* algorithm, an alternative approach to computing short-range forces (discussed in Section 2.1.1). In this section we first provide an overview of the algorithm, and then discuss it in relation to the state of the art.

3.1. Overview

It is easy to see that when two particles are separated by a distance greater than r_c along any single axis (or any unit vector $\hat{\vec{v}}$), the Euclidean distance between them cannot possibly be less than r_c. This is formalised for two particle position vectors $\vec{a}$ and $\vec{b}$ and an arbitrary $\hat{\vec{v}}$ in Eq. (1), and demonstrated graphically in Fig. 2.

$$\forall \hat{\vec{v}}, \vec{a}, \vec{b} \in \mathbb{R}^3, |(\vec{a} - \vec{b}) \cdot \hat{\vec{v}}| \leq \|\vec{a} - \vec{b}\| \tag{1}$$

It follows that if one were to order the particles by their scalar projection onto such a vector, then for each particle there would exist a contiguous block of particles extending either side within r_c along $\hat{\vec{v}}$. Only particles within this block could possibly be within the cut-off radius (note that this is a necessary condition, but not sufficient; a full distance check must still be carried out). Outside of this block all particles could be disregarded. This fact leads to the following three step algorithm:

1. **Selecting** $\hat{\vec{v}}$: The first step is to select a suitable vector $\hat{\vec{v}}$ to use for calculating the projections. For a simulation with non-uniform particle density, the choice of $\hat{\vec{v}}$ can greatly impact the number of particles for which the projections fall within the cut-off radius r_c. Imagine a configuration of particles where all are positioned along a straight line. Choosing $\hat{\vec{v}}$ to be perpendicular to this line would result in all projections being the same, whereas choosing the line itself would spread the projections out as much as possible, and consequently minimise the number of distance checks and maximise performance. In general the choice of $\hat{\vec{v}}$ should be informed by knowledge of the specific simulation, and should ideally maximise the difference between any pair of particle projections. Possible ways of determining a reasonable value for $\hat{\vec{v}}$ in general include principal component analysis and linear regression.
2. **Particle sort:** The particles are then sorted according to their scalar projections onto $\hat{\vec{v}}$. This creates a one-dimensional spatial ordering within which forces can be efficiently calculated.
3. **Force sweep:** For each particle i, loop over all particles j, where j is bounded by k_{lo} and k_{hi}, the first particles below and above i respectively for which the difference between the scalar projections of j and i onto $\hat{\vec{v}}$ exceeds r_c. It is guaranteed by Eq. (1) that no particle outside this set will fall within the cut-off radius. The search space can be further reduced to particles j, $i < j < k_{\text{hi}}$ by using Newton's third law (N3).

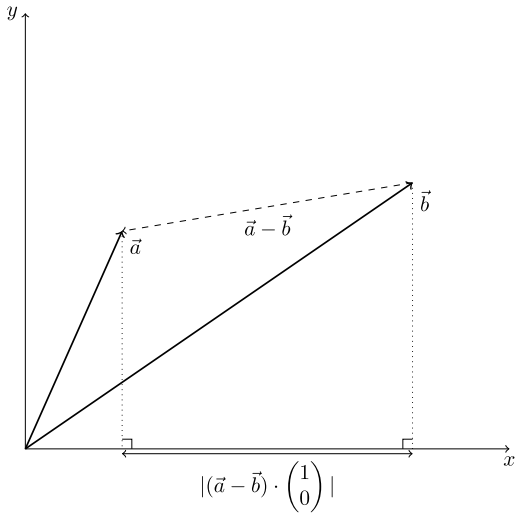

Fig. 2. 2-dimensional illustration of projecting two position vectors onto the x-axis. Eq. (1) clearly follows from Pythagoras' theorem.

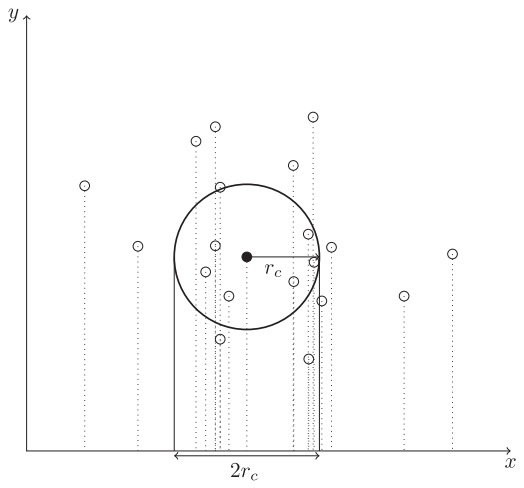

Fig. 3. 2-dimensional illustration of projection sorting on the *x*-axis. We wish to determine which particles are within the cut-off radius of the filled particle (the cut-off radius being marked by the solid circle). We calculate the projection of each particle (indicated by the dotted lines), and perform full distance checks only for those particles whose projections fall within the marked area on the *x*-axis.

```
for (i = 0; i < nb; i++) {
    fxi = 0.0;
    fyi = 0.0;
    fzi = 0.0;
    for (j = i + 1; j < nb; j++) {
        if (projs[j] - projs[i] > rc) break;
        dx = p_x[j] - p_x[i];
        dy = p_y[j] - p_y[i];
        dz = p_z[j] - p_z[i];
        dsq = dx*dx + dy*dy + dz*dz;
        if (dsq <= rcsq) {
            coeff = /* app-dependent */;
            fxi += coeff * dx;
            fyi += coeff * dy;
            fzi += coeff * dz;
#ifdef N3
            f_x[j] -= coeff * dx;
            f_y[j] -= coeff * dy;
            f_z[j] -= coeff * dz;
#endif
        }
    }
#ifndef N3
    for (j = i - 1; j >= 0; j--) {
        if (projs[i] - projs[j] > rc) break;
        // ...
        // as above
        // ...
    }
#endif
    f_x[i] += fxi;
    f_y[i] += fyi;
    f_z[i] += fzi;
}
```

Listing 1: C-like pseudocode demonstrating the projection sorting algorithm. It is assumed that the `projs` array contains scalar projections of each particle position, and that this array and the position arrays `p_x`, `p_y` and `p_z` are all sorted on these projections. The resulting force arrays will also be sorted in this manner. The `N3` preprocessor directive indicates whether or not Newton's Third Law is being used to halve the search space.

Listing 1 gives pseudocode for a simple implementation of this algorithm, and Fig. 3 demonstrates it graphically.

It should be noted that the vector $\hat{\vec{v}}$ need not be reselected every timestep (or indeed ever, depending on the simulation), but instead only when the conformation of particles changes in such a way that it is rendered suboptimal. In cases where the geometry is not particularly stable, a periodic "$\hat{\vec{v}}$ update" subroutine could be introduced which is called every T timestep. Reselecting $\hat{\vec{v}}$ should never make the performance worse, so the only downside to this process is the additional cost of the reselection itself. The parameter T would be tuned to balance this cost (which would be domain dependent) and the gains of using a more optimal vector $\hat{\vec{v}}$. If the geometry of the simulation changes drastically, it is possible that projection sorting may become less optimal, in which case the simulation could switch at runtime to using Verlet lists.

When partitioning the simulation into multiple spatial domains (e.g. for distributed MPI execution), $\hat{\vec{v}}$ can be different in each domain, which allows for tuning of the projection sorting algorithm based on local geometry. Use of N3 for force calculations complicates this somewhat, as implementations need to be careful to calculate forces on ghost particles correctly, but it is still possible.

In the remainder of this paper, we select $\hat{\vec{v}}$ by calculating the vector connecting two distant particles in the dataset and normalising it. We do this only once at the start of the simulation.

3.2. Comparison to Verlet lists

When seeking to compare projection sorting to Verlet lists, it is helpful to break each algorithm down into two parts: periodic computation and force computation. Periodic computation refers to work that needs to be done in preparation for the force computation: for Verlet lists this is building the lists themselves, including constructing cell lists, which needs to be done every k timestep. For projection sorting this includes calculating particle projections and sorting the particles by said projections, which needs to be performed before every force computation. Force computation refers to using the precomputed information to calculate forces for a given set of particle positions. In this section we discuss some of the trade-offs between projection sorting and Verlet lists.

3.2.1. Periodic computation

The algorithmic complexity of computing projections and sorting is straightforwardly $\mathcal{O}(n(1+\log n))$. Building cell and Verlet lists is more complex. The algorithmic complexity of the construction of cell lists (i.e. binning each particle depending on the cell it is in) particles is $\mathcal{O}(n)$ (calculating the cells and adding each particle to the appropriate list). Let $\mathcal{S}(i)$ be the set of particles in cell i. Construction of each particle's Verlet list depends on the contents of the surrounding $3 \times 3 \times 3$ cells. Let $k = \max_j |\mathcal{S}(j)|$, the size of largest cell. Then constructing all Verlet lists is $\mathcal{O}(nk)$ (omitting the constant factor $3^3 = 27$). Therefore the total complexity of the periodic computation associated with Verlet lists is $\mathcal{O}(n(1+k))$. Comparing the complexities therefore reduces to comparing the logarithm of the number of particles against the size of the cell lists. A concrete example from our simulations of $n = 128{,}000$ particles yields the values $\log n \approx 17$ and $k = 44$ (the maximum cell list size, although the mean non-empty cell list size is 7.1).

3.2.2. Force computation

A key advantage of projection sorting is the highly contiguous memory access pattern it affords during the force sweeps. As the particles are sorted they are accessed in a linear order, which allows the memory subsystems to work at peak efficiency, and enables highly efficient SIMD vectorisation. Runtime analysis of hardware counters reveals that the Verlet list implementations encounter nearly three times as many L1 data cache misses as the projection sorting implementations during the force sweeps, and issue over twice as many loads.

Another key metric is the number of particle-pair checks performed by each algorithm (previously discussed in Section 2.1.1). The number of checks is a good predictor of an algorithm's performance [24]. Whether projection sorting or Verlet lists require more spurious checks depends on the skin distance, rebuild period, and overall particle conformation. Typically, projection sorting will require fewer checks along the axis of projection, but more along orthogonal axes.

3.2.3. Communication costs

As MD simulations are almost always executed on clusters nowadays, it is instructive to look at the effect an algorithm might have on the communication costs of particle data between nodes. When using a spatial decomposition, adjacent processors need to share particle data so that particles on the edge of their subdomains have access to particles within their cut-off radius, but owned by another processor. As Verlet lists are constructed based on $r_v > r_c$, they require more data from adjacent processors than projection sorting does. The amount of communication increases cubically with an increased cut-off radius, so can be significant. Conversely, projection sorting needs to communicate more frequently, so again we have a trade-off dependent on skin distance.

In summary, projection sorting works by ordering particles according to their scalar projections onto a specific vector $\vec{v}$. This ordering enables efficient calculation of forces, and the sorting itself is cheap in comparison to other methods. The primary factors to consider when choosing between Verlet lists and projection sorting are (roughly in order of importance):

- Geometry of the simulation (the set of particles having a long axis favours projection sorting),
- Average movement of particles per timestep (lower allows for a smaller r_s),
- Projection sorting uses memory bandwidth more effectively,
- Higher SIMD width favours Verlet lists.

4. Experiments

In this section, we discuss highly optimised implementations of the projection sorting algorithm described in Section 3 and the combination of cell lists and Verlet lists described in Section 2, in the context of a modern MD simulation. We then compare the performance characteristics of the two.

4.1. Experimental setup

The vehicle for our implementation is a biophysical simulation designed to emulate chromosome condensation published by Cheng et al. [6]. The particles in this simulation are bundles of proteins known as nucleosomes, and are represented as homogeneous spheres of radius 5 nm. They are connected by "DNA-linkers", modelled as ideal springs following Hooke's Law. These particles are free to move according to Brownian motion, subject to certain constraints. One in particular concerns us: a repulsive force designed to keep particles from overlapping each other. Pairs of particles that come within 15 nm of each other are repelled. Fig. 4 illustrates how the magnitude of this force varies with distance.

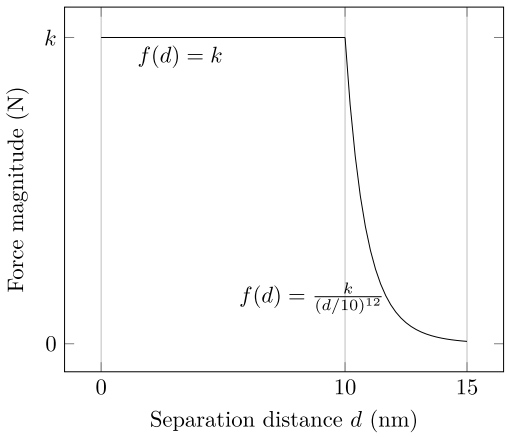

Fig. 4. Repulsive force on particle pairs within 15 nm. k is a configurable constant indicating the strength of the force.

In the original implementation, the majority (≈95%) of the runtime was spent computing these repulsion forces. This kernel is an excellent candidate for the projection sorting algorithm.

4.1.1. Datasets

The initial dataset described by Cheng et al. [6] was derived from a budding yeast cell and contains 2000 nucleosomes (see Fig. 5). As no larger real datasets were available while this work was being undertaken, we generate extended versions of the original using statistical methods. We define three normal distributions, each parameterised using the mean and standard deviation of the deltas between each nucleosome for the x, y and z axes respectively. We then generate new conformations of length N particles by sampling these distributions to perform an N step random walk. After generation we simulate the conformation for 100,000 timesteps to reach a relatively stable state, free of artefacts caused by the random walk process. Synthetic datasets are generated for $N = 2^k \cdot 10^3$, $2 \leq k \leq 11$.

4.1.2. Machine specifications

To run our experiments, two machines are used. Firstly, Orac, a cluster at the University of Warwick consisting of 84 nodes, each with 2 Intel Xeon E5-2680 v4 processors, for 2352 cores in total. Each node is equipped with 128 GB of RAM, and connected with an Intel Omni-Path X16 100 Gb/s interconnect. Orac is used to run the Xeon experiments.

To run the Xeon Phi experiments we use the ARCHER Knights Landing platform at the Edinburgh Parallel Computing Centre. This consists of 12 nodes, each equipped with a Xeon Phi 7210 and 96 GB of RAM. The nodes are connected with a high performance Cray Aries interconnect. Each Xeon Phi also has 16 GB of fast onboard MCDRAM, which in this case is used to cache accesses to the external memory.

We refer the reader to Table 1 for more details.

4.1.3. Compilation and execution

All of the code is compiled using the GNU C++ compiler, v6.3.0 with the following performance flags: `-O3 -fopenmp`. The machine architecture is specified with `-march=broadwell`, and `-march=knl` for Xeon and Xeon Phi respectively.

On Orac the application is launched using the `mpirun` command. As the application domain partitioning code currently only supports power-of-two-sized decompositions and Orac has 28

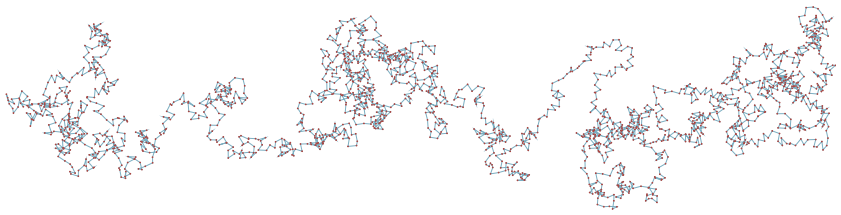

Fig. 5. A visualisation of one of the datasets used — a conformation of yeast DNA dotted with nucleosomes.

Table 1
Summary of hardware configurations on Orac and the ARCHER Knights Landing platform.

	Orac	KNL platform
# nodes	84	12
Processor model	Xeon E5-2680 v4	Xeon Phi 7210
Sockets × Cores × Threads	2 × 14 × 2	1 × 64 × 4
Clock (GHz)	2.4	1.3
L1{i,d}/L2 (kB)	{32,32}/256	{32,32}/512
Memory (GB)	128	96+16
SIMD ISA	AVX2	AVX-512

cores per node, MPI ranks are distributed evenly among the available cores using the following flags: `--map-by socket:SPAN`, `--bind-to core` and `--rank-by core`. Each rank is assigned to a unique core, and uses a single thread. We run up to 256 MPI ranks, which is the largest power-of-two core count that Orac is configured to allow. These ranks are spread as evenly as possible across the 280 cores available on 10 nodes.

On ARCHER the application is launched using the `aprun` command. We use varying combinations of MPI ranks and OpenMP threads, which is explained in more detail below. We use an explicit mapping from software threads to hardware threads, supplied using the `-cc` flag. We run up to four nodes (each Xeon Phi supports up to 256 hardware threads).

All experiments are run for 1000 timesteps. When using Verlet lists, we set the skin distance to the empirical minimum $r_s = 15$ nm, and $k = 5$, which yields the best performance for the given value of r_s.

Detailed timing information is collected across all areas of the code using the `rdtsc` hardware counter in order to achieve high accuracy with minimal overhead.

4.2. Implementation

The basic structure of the application is based on the miniMD mini-app from Sandia National Laboratories' Mantevo suite [10], which is in turn based on the larger and more complex LAMMPS [19]. We use a spatial decomposition which theoretically scales well with processor count, at the price of more complex load balancing in the case of simulations with non-uniform particle densities (like this one).

Each MPI rank stores an array of local particle positions and ghost particle positions, and a corresponding array of forces on each particle. At each timestep the forces on the local particles are calculated. If N3 is enabled, some forces on ghost particles will also be determined. These forces are then communicated back to the owning rank. The local particle positions are then updated based on the forces. If any particles have moved into the domain of a different MPI rank these are communicated, and new ghost particles are shared. This process then repeats. This communication process is identical between projection sorting and Verlet lists.

Currently the simulation implements a static load balancing strategy wherein the problem domain is decomposed into small cubic "patches" of uniform size. Each patch is then weighted according to how many particles it contains, and patches are distributed among processors by a recursive bisection algorithm, such that all processors end up with a cuboidal subdomain of approximately equal total weight. This strategy is used by NAMD [16] and is based on work from Berger and Bokhari [4].

NAMD implements a layer of dynamic load balancing on top of this, which distributes pairwise force calculations between neighbouring processors. Our implementation does not currently do this, although it is a possible extension. The static load balancing performs well enough to show the differences between Verlet lists and projection sorting.

The application is parallelised in a so-called "hybrid" fashion. Within each MPI rank, OpenMP may be used to leverage multiple threads. We use this capability to take advantage of simultaneous multithreading (SMT), or hyperthreading, on Knights Landing.

Our implementation has two main build-time switches: use of projection sorting or Verlet lists, and enabling or disabling halving of the neighbour space using Newton's third law. We refer to these four variants below as *PS N3*, *PS no N3*, *VL N3* and *VL no N3*.

4.2.1. Projection sorting

Force computation using the sorted conformation is hand-vectorised using both AVX2 and AVX-512 intrinsics. We refer the reader to our previous work [13] for more detailed information.

4.2.2. Verlet lists

Given the small cut-off radius in this simulation, it is desirable to construct cell-lists based on a fine decomposition, otherwise there will be many spurious checks. Calculating these naïvely (by allocating a three dimensional array of bins) uses an infeasibly large amount of memory, which increases cubically with the problem size. Instead we use a lock-free hash map implementation (based on Marçais et al. [14]) to construct the lists in a much smaller amount of space, without sacrificing high performance when running multiple threads.

Verlet list rebuilds (based on the cell-lists) and force computation using Verlet lists were hand-vectorised using both AVX2 and AVX-512 intrinsics as described by Pennycook et al. [17].

4.3. Experiments

In this section we seek to empirically establish the performance differences between our implementations of Verlet lists and projection sorting in a realistic context. In order to do this, we present the following comparisons:

- Raw performance comparisons between the two algorithms for fixed numbers of processors and problem sizes (see Section 4.3.1).
- A scaling study for each implementation, showing how the performance changes with varying numbers of processors and different problem sizes (see Section 4.3.2).
- Qualitative performance comparisons between the Xeon and Xeon Phi platforms (see Section 4.3.3).

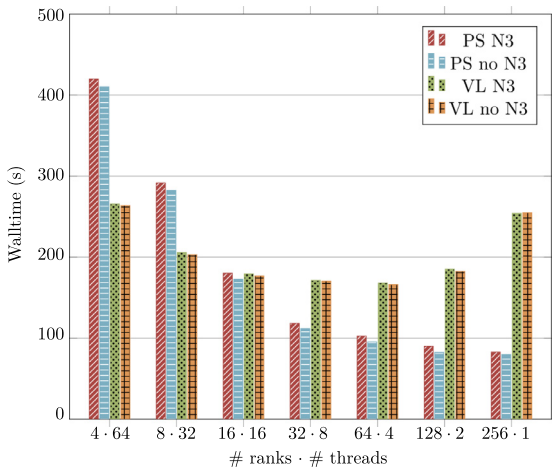

Fig. 6. Observed simulation times on the ARCHER Knights Landing platform across a variety of different runtime configurations (combinations of MPI ranks and OpenMP threads) on a single Xeon Phi node. The *x*-axis shows the number of MPI ranks and OpenMP threads used, for example 4·64 indicates 4 MPI ranks and 64 OpenMP threads per rank. In all cases the 2,048,000 particle dataset is simulated for 1000 timesteps. Lower walltime is better.

Prior to running any experiments on the ARCHER Knights Landing platform, we first perform a series of experiments to determine the optimal configuration of MPI ranks and OpenMP threads on each Xeon Phi node for our application variants (optimal in the sense of lowest overall simulation walltime). The results of these experiments are shown in Fig. 6. We find that projection sorting performs best at 256 MPI ranks with a single thread per rank, likely because sorting is not a thread-scalable operation. The results when using Verlet lists are more balanced: 64 MPI ranks and four threads per rank yield the fastest results. To simplify the comparison, we opt to use 64 MPI ranks and four threads per rank for all subsequent Knights Landing experiments. This configuration offers the best performance for Verlet lists and only 20% below best performance for projection sorting. It is important to use the same configuration for both as the number of MPI ranks per node can dramatically affect the performance as the simulation is scaled. Using the same number of MPI ranks per node for all application variants ensures comparable results.

4.3.1. Raw performance comparisons

Fig. 7 shows the speedup attained by projection sorting relative to the performance of Verlet lists, for the entire simulation (including kernels unrelated to the repulsion calculations). This is significant for all four configurations, reaching nearly 5× on Orac.

We observe that PS attains a greater relative speedup when N3 is disabled, although enabling N3 results in better absolute performance. On ARCHER, PS sees approximately 1.7× speedup. We believe this lower speedup is due to better hardware support for gather/scatter and vector compression on KNL, which VL relies on.

Fig. 8 breaks these speedups down into periodic calculations, force calculations and comms costs across Xeon and Xeon Phi. Notable are the communication costs, which are higher for PS. This is because VL only needs to communicate fully every $k = 5$ timestep, which fits with the approximate 0.2× speedup we see. We note also that the speedup improves with the number of processors. We conjecture that this is due to the lower amount of communication required per processor in PS (discussed in Section 3.2.3). The higher communication costs are more than offset by the significantly lower periodic costs however. On ARCHER these reach speedups of over 10×.

Fig. 9 shows the absolute CPU core time spent per particle per timestep (the aggregate of all simulation kernels) for each application variant. If the application scaled perfectly linearly we would see flat lines here, as it is the time increases at scale due to processing overheads. The Xeon Phi times are higher as each single KNL core is slower than a single Xeon core. Time per particle is often presented as a performance statistic for MD simulations, however we would note that these numbers are not directly comparable to other MD applications, as they include time spent in domain-specific calculations of tension, angular and condensing forces.

4.3.2. Scaling studies

Figs. 10 and 11 show the application's strong and weak scaling on Orac and ARCHER. Both algorithms scale well, with VL reaching over 150× for 256 MPI ranks on Orac. VL generally scales slightly better than PS, likely due to the lower communication costs. We expect that this gap could be closed if PS was adapted to use a skin distance (see Section 5.1).

Fig. 11 shows some anomalous super-linear scaling for PS, likely due to the lower amount of data per processor improving cache effectiveness.

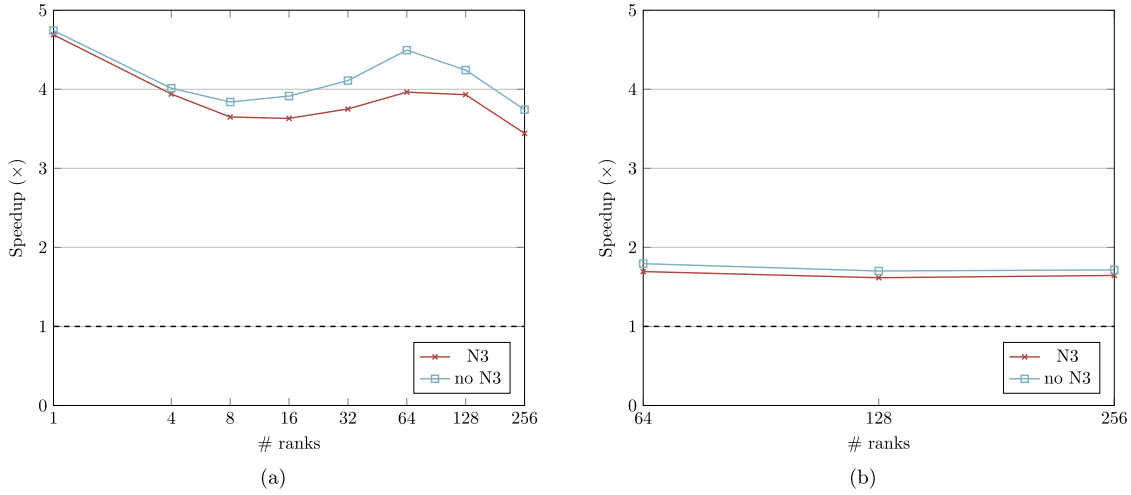

Fig. 7. (a) shows the speedup of the full simulation of 2,048,000 particles when using PS, relative to a baseline using VL on Orac up to 256 MPI ranks, with a single thread per rank. (b) shows the same on the ARCHER Knights Landing platform up to four nodes, with 64 MPI ranks per node and four OpenMP threads per rank. Higher speedup is better.

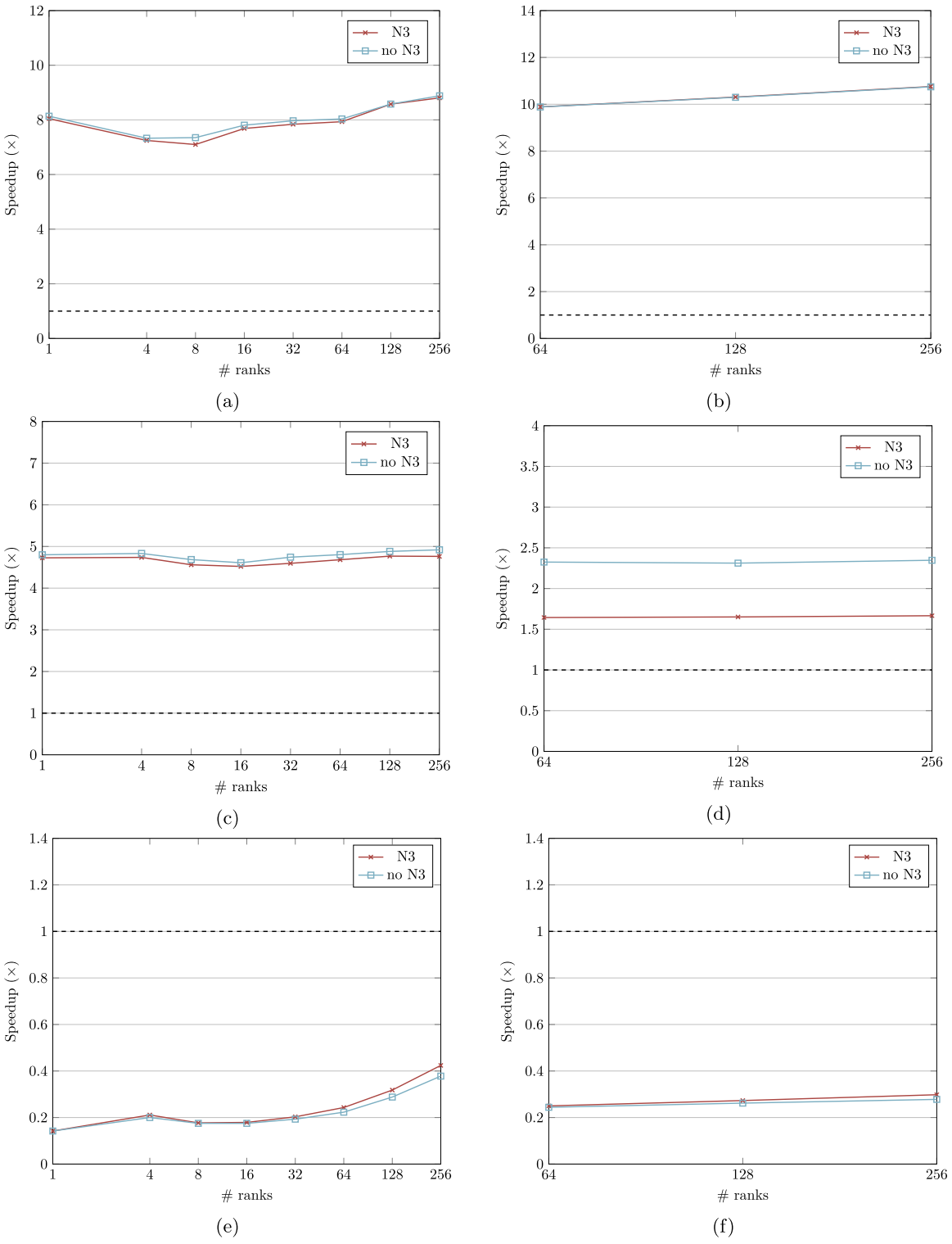

Fig. 8. Speedup for PS kernel costs relative to VL kernel costs on Orac and the ARCHER Knights Landing platform. Orac results are shown up to 256 MPI ranks with a single thread per rank, when simulating 2,048,000 particles. ARCHER results are shown up to four nodes with 64 MPI ranks per node and four OpenMP threads per rank, when simulating 2,048,000 particles. (a) and (b) show periodic costs for Orac and ARCHER respectively, (c) and (d) show force calculation costs, and (e) and (f) show communication costs. Higher speedup is better.

4.3.3. Xeon vs. Xeon Phi

It is inherently difficult to compare performance between Xeon and Xeon Phi architectures, as they differ in almost every important aspect. Equal numbers of cores are not comparable, as KNL cores are much slower individually. In our previous work [13] we found that the many-core architecture was well suited to the force calculations as it has double the vector width available.

Fig. 8d shows that disabling N3 gets a greater relative increase to force calculation speedup than on Orac (cf. Fig. 8c). This is because KNL crucially uses SMT to keep its execution units busy, and multiple threads with N3 enabled necessitate atomic accesses

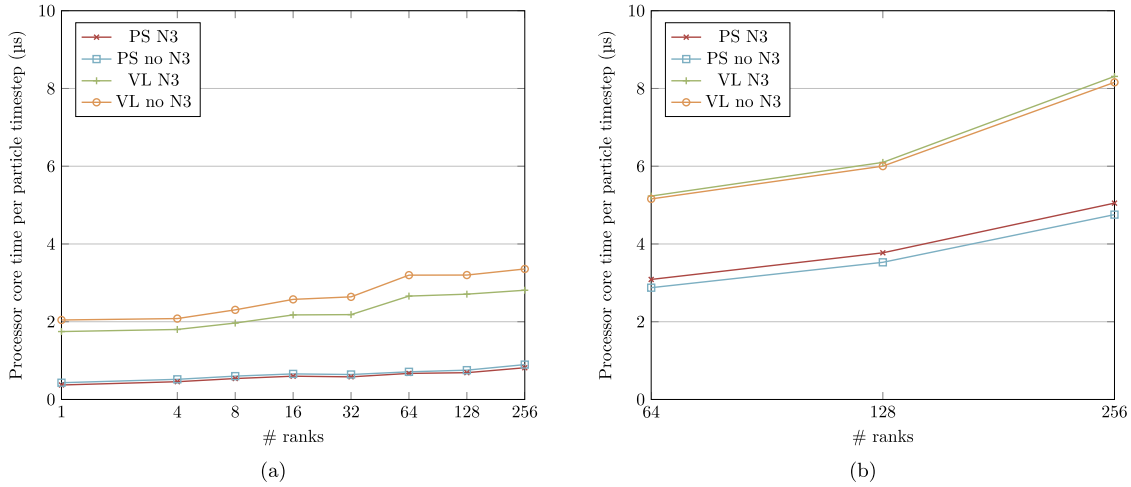

Fig. 9. (a) shows the mean time in microseconds spent updating a single particle from one timestep to the next (normalised by the number of processor cores in use) for each application variant on Orac, derived from simulations of 2,048,000 particles over 1000 timesteps. (b) shows the same for the ARCHER Knights Landing platform. Lower is better.

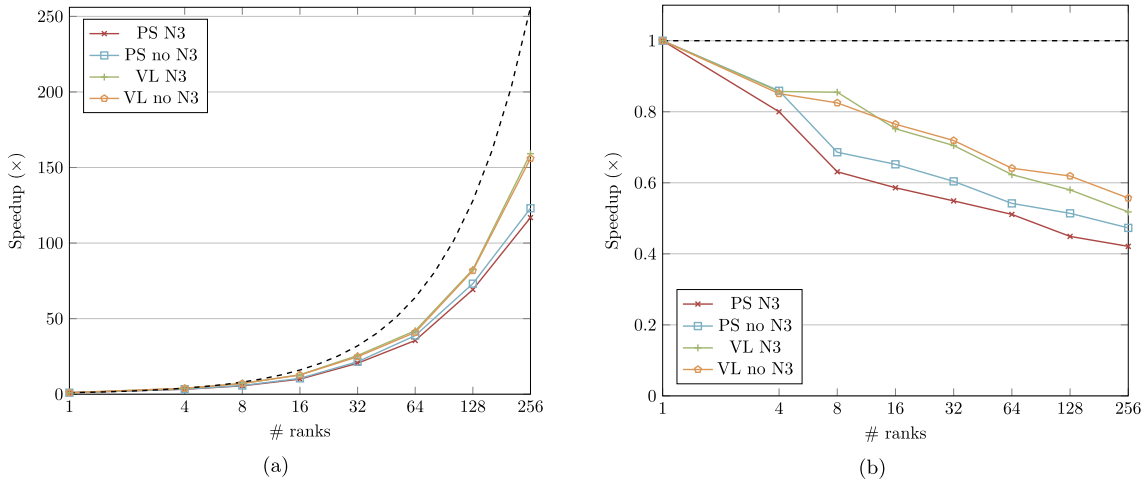

Fig. 10. Scaling on Orac up to 256 MPI ranks. (a) shows strong scaling over 2,048,000 particles and (b) shows weak scaling with 8000 particles per MPI rank. Higher speedup is better.

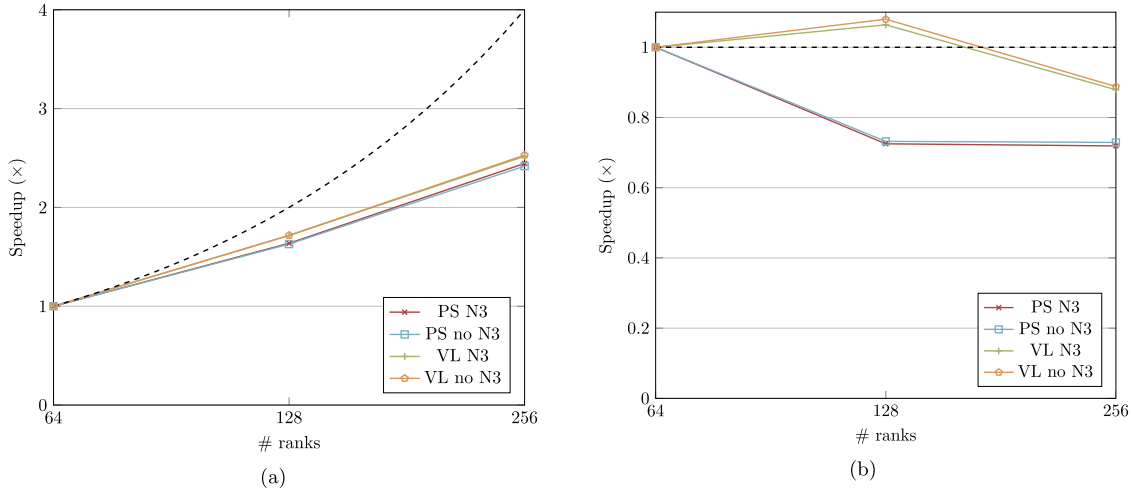

Fig. 11. Scaling on the ARCHER Knights Landing platform up to 256 MPI ranks. (a) shows strong scaling over 2,048,000 particles and (b) shows weak scaling with 8000 particles per MPI rank. Higher speedup is better.

to shared memory to prevent conflicts. KNL has poor support for vectorised atomic writes. On Orac SMT is not used, therefore the atomics are not necessary. This effect is also visible in the VL rebuilds, which use atomics as part of a lock-free hash table (compare Figs. 8b and 8a).

In summary, we have shown that – for this simulation – PS is significantly faster than VL, and scales comparably to high processor counts. This applies to both Intel Broadwell and Intel Knights Landing.

5. Conclusions

In this paper, we present *projection sorting*, an alternative to the traditional Verlet list algorithm for pairwise short-range force calculations, and show that it can be significantly more effective in a modern MD simulation of chromosome condensation.

We provide efficient parallel implementations of this strategy for the modern multi- and many-core architectures, Intel Broadwell and Knights Landing. We also extend the implementations to multi-node environments enabling the use of large clusters to significantly reduce runtimes.

We demonstrate relative speedups approaching up to 5× across a range of problem sizes and processor counts. Our algorithm scales comparably to the state of the art in both strong and weak senses, with the runtime being reduced by up to 125× over 256 processors. These results are not just theoretical, but are obtained from a real MD simulation. Our algorithm and optimisations have been and continue to be used to facilitate further experiments into chromosome condensation using this simulation.

5.1. Further work

It is conceivable to use projection sorting in concert with the skin distance, by only sorting the particles every k timestep, and continuing up to the "Verlet" cut-off radius, r_v, when performing force sweeps. As the sort is considerably cheaper than a Verlet rebuild, it is possible that the computational costs outweigh the benefits, but we note that it would also improve the communication costs, which may balance this increase.

We are also investigating alternate spatial decomposition strategies, to offset PS' main disadvantage: that it must inspect more particles in directions orthogonal to the projection axis. We hypothesise that a "pencil" decomposition – decomposing space into long thin slivers along the axis of projection – could significantly decrease the number particles to be processed, in addition to decreasing the communication complexity at high scale.

Implementation of projection sorting in other existing MD software would be valuable to establish the extent of its applicability in a more general sense. NAMD is a popular production MD package designed for simulation of large biomolecular structures which has been the subject of substantial HPC research, and has been shown to scale to over 500,000 cores. NAMD is a large code, and implementing a new algorithm would be a time consuming process, however we believe this would be a worthwhile endeavour, and is crucial to gaining further insight into the properties of the projection sorting algorithm, and directions in which to take it.

Acknowledgments

This work was supported by the Francis Crick Institute which receives its core funding from Cancer Research UK (FC001003), the UK Medical Research Council (FC001003), and the Wellcome Trust, UK (FC001003), and by the Engineering and Physical Sciences Research Council, UK and Intel Corporation, United States (CASE award 1365607). This work used the ARCHER UK National Supercomputing Service.

References

[1] M.J. Abraham, T. Murtola, R. Schulz, S. Páll, J.C. Smith, B. Hess, E. Lindahl, GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers, SoftwareX 1–2 (2015) 19–25.
[2] B.J. Alder, T.E. Wainwright, Studies in molecular dynamics. I. General method, J. Chem. Phys. 31 (2) (1959) 459–466.
[3] J.A. Anderson, C.D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, J. Comput. Phys. 227 (10) (2008) 5342–5359.
[4] Berger, Bokhari, A partitioning strategy for nonuniform problems on multiprocessors, IEEE Trans. Comput. C-36 (5) (1987) 570–580.
[5] A. Bhatele, S. Kumar, C. Mei, J.C. Phillips, G. Zheng, L.V. Kalé, Overcoming scaling challenges in biomolecular simulations across multiple platforms, in: Proceedings of the International Parallel and Distributed Processing Symposium 2008, IEEE, 2008, pp. 1–12.
[6] T.M.K. Cheng, S. Heeger, R.A.G. Chaleil, N. Matthews, A. Stewart, J. Wright, C. Lim, P.A. Bates, F. Uhlmann, A simple biophysical model emulates budding yeast chromosome condensation, eLife 4 (2015) e05565.
[7] E. Fermi, J. Pasta, S. Ulam, M. Tsingou, Studies of Nonlinear Problems, Tech. Rep, 1955.
[8] 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) 570–573.
[9] A. Harode, A. Gupta, B. Mathew, N. Rai, Optimization of molecular dynamics application for Intel Xeon Phi coprocessor, in: Proceedings of the International Conference on High Performance Computing and Applications 2014, IEEE, 2014, pp. 1–6.
[10] M.A. Heroux, D.W. Doerfler, P.S. Crozier, Improving performance via mini-applications, IEEE Trans. Comput. (2009).
[11] R.W. Hockney, S.P. Goel, J.W. Eastwood, Quiet high-resolution computer models of a plasma, J. Comput. Phys. 14 (2) (1974) 148–158.
[12] W. Jiang, J.C. Phillips, L. Huang, M. Fajer, Y. Meng, J.C. Gumbart, Y. Luo, K. Schulten, B. Roux, Generalized scalable multiple copy algorithms for molecular dynamics simulations in NAMD, Comput. Phys. Comm. 185 (3) (2014) 908–916.
[13] T.R. Law, J. Hancox, T.M.K. Cheng, R.A.G. Chaleil, S.A. Wright, P.A. Bates, S.A. Jarvis, Optimisation of a molecular dynamics simulation of chromosome condensation, in: 2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), IEEE, 2016, pp. 126–133.
[14] G. Marcais, C. Kingsford, A fast, lock-free approach for efficient parallel counting of occurrences of k-mers, Bioinformatics 27 (6) (2011) 764–770.
[15] B. Moon, H.V. Jagadish, C. Faloutsos, J.H. Saltz, Analysis of the clustering properties of the Hilbert space-filling curve, IEEE Trans. Knowl. Data Eng. 13 (1) (2001) 124–141.
[16] M.T. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L.V. Kalé, R.D. Skeel, K. Schulten, NAMD: a parallel, object-oriented molecular dynamics program, Int. J. Supercomput. Appl. High Perform. Comput. 10 (4) (1996) 251–268.
[17] S.J. Pennycook, C.J. Hughes, M. Smelyanskiy, S.A. Jarvis, Exploring simd for molecular dynamics, using intel® xeon® processors and intel® xeon phi™ coprocessors, in: Proceedings of the International Symposium on Parallel and Distributed Processing 2013, IEEE Computer Society, 2013, pp. 1085–1097.
[18] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kalé, K. Schulten, Scalable molecular dynamics with NAMD, J. Comput. Chem. 26 (16) (2005) 1781–1802.
[19] S.J. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys. 117 (1995) 1–19.
[20] B. Quentrec, C. Brot, New method for searching for neighbors in molecular dynamics computations, J. Comput. Phys. 13 (3) (1973) 430–432.
[21] Y. Sun, G. Zheng, C. Mei, E.J. Bohm, J.C. Phillips, L.V. Kalé, T.R. Jones, Optimizing fine-grained communication in a biomolecular simulation application on Cray XK6, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis 2012, IEEE, 2012, pp. 1–11.
[22] I.T. Todorov, W. Smith, K. Trachenko, M.T. Dove, DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism, J. Mater. Chem. 16 (20) (2006) 1911–1918.
[23] L. Verlet, Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, Phys. Rev. 159 (1) (1967) 98–103.

[24] U. Welling, G. Germano, Efficiency of linked cell algorithms, Comput. Phys. Comm. 182 (3) (2011) 611–615.
[25] Z. Yao, J. Wang, G. Liu, M. Cheng, Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method, Comput. Phys. Comm. 161 (1–2) (2004) 27–35.
[26] G. Zhao, J.R. Perilla, E.L. Yufenyuy, X. Meng, B. Chen, J. Ning, J. Ahn, A.M. Gronenborn, K. Schulten, C. Aiken, P. Zhang, Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics, Nature 497 (7451) (2013) 643–646.

Timothy Law completed his Ph.D. in 2017 at the University of Warwick, in collaboration with Intel Corporation, as a member of the Department of Computer Science's High Performance and Scientific Computing group. His research focused on performance optimisation of scientific applications used for research in the life sciences, in particular molecular dynamics, genome assembly and sequence alignment.

Jonny Hancox is a Software Architect from the Health & Life Sciences team at Intel Corporation. His work is focused on helping scientists and clinicians to make best use of the latest computational tools. Jonny has worked in various technical roles throughout his career, with an emphasis on extracting value from unstructured and challenging datasets.

Steven Wright is a Lecturer at the University of York. His research in High Performance Computing focuses on the use of HPC in physics applications and energy-aware optimisation of supercomputing applications. Dr. Wright was previously a post-doctoral Research Fellow at the University of Warwick, working with the Centre for Computational Plasma Physics, colleagues from several UK universities, the UK Atomic Energy Authority, the Rutherford Appleton Laboratory and the UK Atomic Weapons Establishment (AWE). Dr. Wright completed his Ph.D. in 2014 at the University of Warwick, in collaboration with AWE plc. and Los Alamos National Laboratory, focusing on I/O in parallel applications.

Professor Jarvis studied at London, Oxford and Durham Universities before taking his first Lectureship at the Oxford University Computing Laboratory. Here he worked on the development of performance tools for the BSP programming library, as well as teaching at Brasenose, Lincoln and Keble Colleges. After a short secondment to Microsoft Research in Cambridge, he joined the University of Warwick, rising to Professor in 2009. Professor Jarvis acted as Director of Research from 2008 to 2013, leading the Department to rank 2nd (out of 89 UK Computing Departments) in the 2014 UK Research Excellence Frame-work (REF). In 2013 he was appointed Chair of Department. Professor Jarvis has been a Visiting Exchange Professor at New York University since 2017 and is currently a member of the Board of Trustees at the Alan Turing Institute, the UK's national institute for data science. He is presently Deputy Pro Vice Chancellor (Research) at the University of Warwick.