4 Technical Report (publishable)

ARCHER eCSE Technical Report

Algorithmic Enhancements to the Solvaware Package for the Analysis of Hydration

Technical staff: Arno Proeme (EPCC, University of Edinburgh)
PI: David Huggins (TCM, Department of Physics, University of Cambridge)

4.1 Acknowledgement

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk)

4.2 Abstract

This report details the work undertaken as part of ARCHER eCSE project 03-03 to port, optimise, and enhance the Solvaware software package for the analysis of solvation (hydration) in biomolecular systems. Each planned work package is reported against and results as well as a technical discussion are presented.

4.3 Introduction

Solvaware consists of a suite of tools in the form of Perl scripts and C++ code that taken together largely automate a workflow enabling users to generate molecular dynamics (MD) trajectories and analyse these using a novel implementation of an algorithm based on the statistical mechanical method of inhomogeneous fluid solvation theory (IFST). This is a transformative technique in structure-based drug design, as it allows users to model and understand the thermodynamics of individual water molecules around biomolecules.

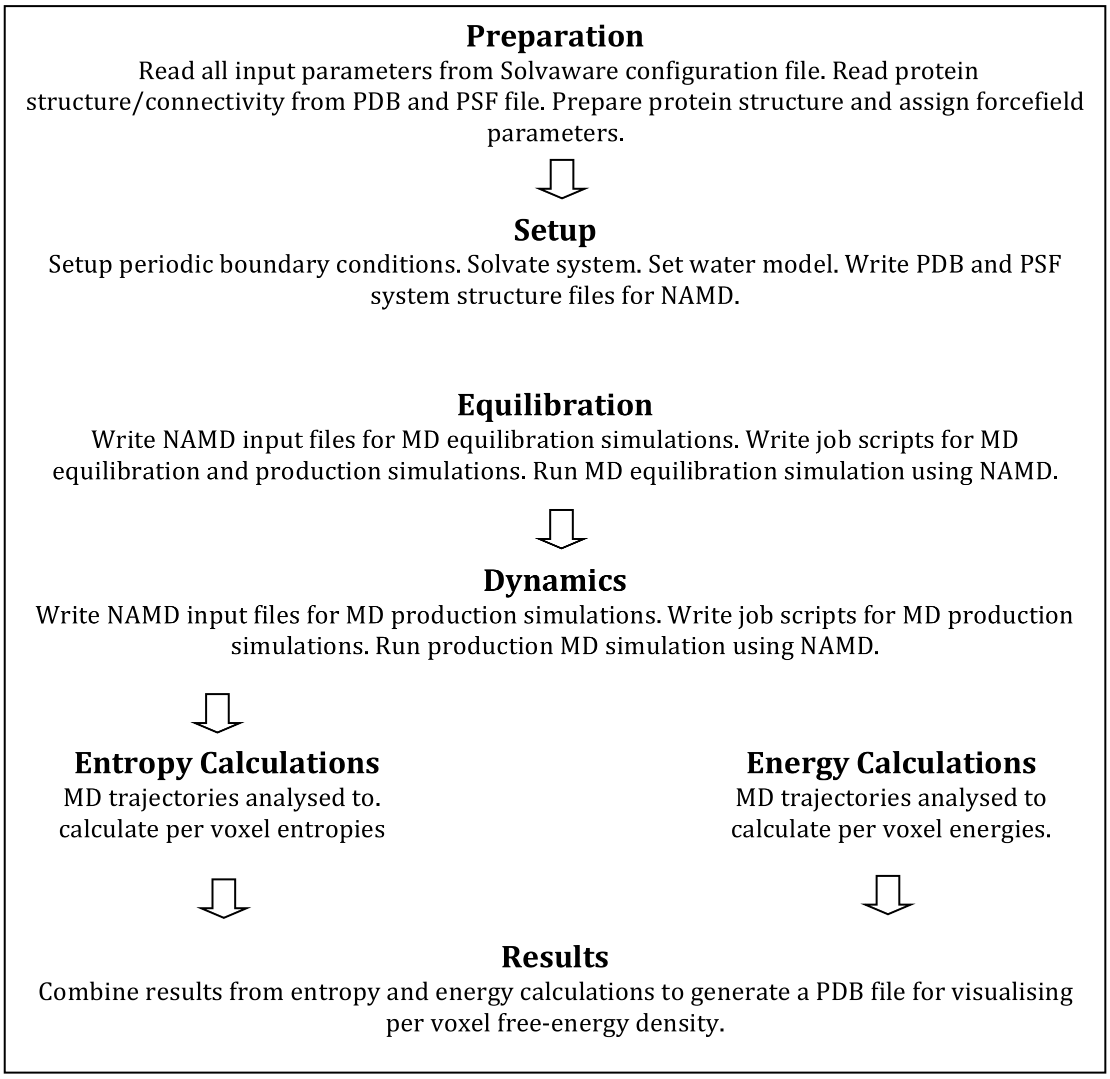
The first stage of the workflow is to prepare the input structure and generate MD trajectories using the popular third-party high-performance MD package NAMD[1]. The flow diagram in Figure 1 illustrates the workflow.

Each of the steps in this stage is run in sequence and controlled by a Perl script. Each Perl script reads a Solvaware configuration file, which contains approximately 50 options. The last two steps run simulations using NAMD whereas the first two steps run utilities written in C++ that are part of Solvaware, as well as one external dependency, namely the freely available Solvate utility[2].

Steps 1-2 are very quick, taking on the order of a minute to complete in serial on modern processor architectures. Depending on the system being studied and the computational resources allocated, step 3 (equilibration) takes on the order of 3 hours to complete, and step 4 (production dynamics run) takes on the order of 24 hours. In terms of storage, step 4 produces a NAMD output trajectory in the DCD binary file format that is 3GB-10GB, again depending on the size of the system.

[1] http://www.ks.uiuc.edu/Research/namd/

[2] http://www.mpibpc.mpg.de/grubmueller/solvate



Preparation
Read all input parameters from Solvaware configuration file. Read protein structure/connectivity from PDB and PSF file. Prepare protein structure and assign forcefield parameters.

Setup
Setup periodic boundary conditions. Solvate system. Set water model. Write PDB and PSF system structure files for NAMD.

Equilibration
Write NAMD input files for MD equilibration simulations. Write job scripts for MD equilibration and production simulations. Run MD equilibration simulation using NAMD.

Dynamics
Write NAMD input files for MD production simulations. Write job scripts for MD production simulations. Run production MD simulation using NAMD.

Entropy Calculations
MD trajectories analysed to. calculate per voxel entropies

Energy Calculations
MD trajectories analysed to calculate per voxel energies.

Results
Combine results from entropy and energy calculations to generate a PDB file for visualising per voxel free-energy density.

Figure 1 - Flow diagram illustrating the workflow.

The latter stages of the Solvaware workflow involve the IFST analysis, which is implemented in C++. This amounts to calculating the mean enthalpy and performing a k-nearest neighbour (KNN) search in 3 dimensions on snapshots from the MD trajectory to calculate the mean entropy. This allows the solvation (hydration) free energy for the solute to be computed, which can give novel insight into intermolecular interactions.

4.4 Project Work Packages

WP1 Develop a Solvaware package on ARCHER

An improved Solvaware package containing the enhancements implemented across all work packages has been made centrally available on ARCHER. In this section we describe the enhancements that were made specifically to facilitate the creation, adoption, and efficient usage of this package by ARCHER users as well as its potential deployment and use on other, similar systems.

As part of the process of porting Solvaware to ARCHER existing Perl scripts were generalised away from close integration with the University of Cambridge's Darwin machine on which they had thus far been used by removing explicit executable paths and options specific to this machine. Scripts were modified to read machine-specific options specified at package installation time or at run time, as appropriate, as well as settings supplied by the environment module system that is generically used to load and modify user environments – including executable paths – on high-performance computing systems. Scripts were modified to read some of the run-time options from a configuration file that users are already expected to modify to set up their problem for Solvaware to solve, thereby enhancing transparency and ease of use. Efficient machine-specific installation and runtime defaults for ARCHER and Darwin were included.

Single-node and multi-node NAMD benchmarking was performed in order to determine optimal use of ARCHER for typical biomolecular systems simulated with NAMD in order to be analysed by Solvaware – see figures 2 and 3.

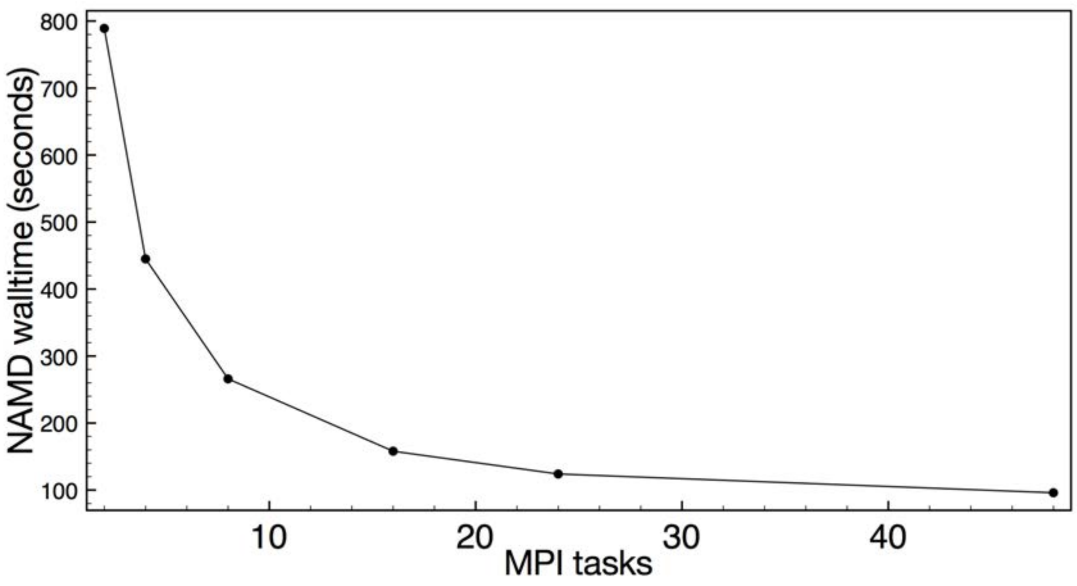


Figure 2 – single-node NAMD benchmark of cb7amo test case on ARCHER. Best time is 96 seconds using 48 tasks (hyperthreading enabled). Best time without hyperthreading enabled is 124 seconds for 24 tasks.

As shown in figure 2, benchmarking revealed that for systems typically explored using Solvaware a performance gain of 20-30% during the time-consuming MD simulation stage can be achieved by enabling Intel hyperthreads. This was included in the default job script for NAMD generated by Solvaware, thus ensuring efficient use of ARCHER by users of Solvaware.

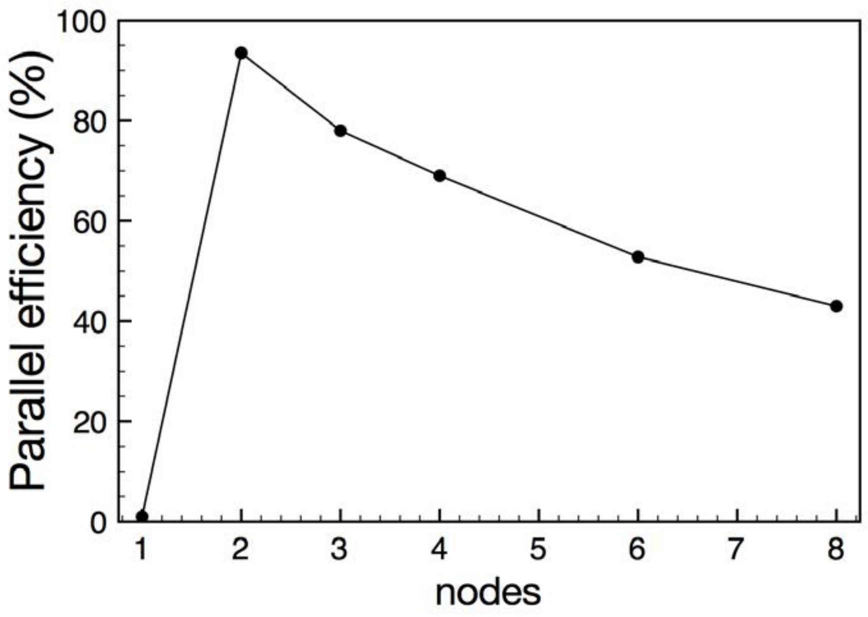


Figure 3 – multi-node NAMD benchmark of cb7amo test case on ARCHER. Parallel efficiency drops to ~50% from 6 nodes onwards.

Also included in the package repository are an installation script, and an environment module file that serves as a template for deployment on other machines.

Some scripts and utilities were rewritten to make the workflow more seamless especially for non-expert users, requiring less knowledge and intervention and providing more informative feedback during each step.

Identical code duplicated across Perl scripts was refactored into a shared library, thereby facilitating future development work.

Finally, documentation in the form of a user manual explaining how to use the package and detailing the meaning and relevance of available options was produced and made available on the ARCHER website.

WP2 Improve Efficiency of k-Nearest Neighbours Algorithm

The molecular dynamics simulation performed in the first stage of the Solvaware workflow produces a trajectory for all the atoms in the system, i.e. both those that make up a biomolecule of interest, typically a protein, as well as atoms that make up the solvent molecules, typically water, surrounding the biomolecule. This is stored in a DCD format binary file as a series of snapshots, or frames, each of which contains the positions of all atoms at one instant of the simulation.

A key part in the second stage of the Solvaware workflow is the determination by a k-nearest neighbours (KNN) algorithm of the k water molecules closest, under an appropriate distance metric and across all frames, to a given water molecule. Computing this for all water molecules allows the IFST algorithm to compute entropies and hence, in combination with a computation of the energies, the solvation (hydration) free energy.

Grid-based restructuring and nearest-voxels approximation

Prior to this eCSE project the KNN algorithm in the existing serial C++ code made a costly all-to-all comparison: for F frames chosen from the trajectory and W water molecules present in the system, and defining n = WF, the algorithm performed $O(n^2)$ distance metric evaluations. This naïve algorithm was problematic as computation of hydration energy for large systems or with high accuracy (many frames) was too costly.

To improve the efficiency of the KNN algorithm, it was restructured by introducing a three-dimensional spatial grid consisting of voxels, with one voxel associated to each point on the grid. This grid was first implemented as a three-dimensional STL vector whose components are instances of a newly-defined Voxel class:

```
vector<vector<vector<Voxel> > >  allvoxels
```

Each Voxel in turn stores a vector of newly-defined Water objects:

```
vector<Water> waterposes_
```

Finally each Water contains 11 doubles that store the position of a water molecule and its orientation in a quaternion representation, as well as an integer that stores the frame in which this water molecule was encountered in the DCD file. Before the KNN algorithm executes, the desired trajectory data is read from the DCD file and all water molecules stored in voxels according to their spatial location. The size of the grid is determined by the size of the cell used during the MD simulations, which is identical throughout the workflow and specified in a configuration file.

This restructuring allowed an extremely fruitful approximation to be made to the KNN algorithm, namely to restrict the KNN search to only consider water molecules located in voxels that surround the one containing a given water molecule of interest. The current implementation allows for arbitrarily far away voxels to be considered, down to just immediate neighbouring voxels. For the approximation to remain accurate if one chooses to increase the number of neighbouring waters, k, to be considered it stands to reason that one will need eventually to also extend the voxel search range to include voxels in which these further away neighbouring waters can be found.

The benefits of avoiding $O(n^2)$ scaling thanks to these enhancements to the code are already apparent at very modest scale. An IFST calculation over a mere 100 frames of a DCD file (which can contain $O(10^6)$ frames in total) takes roughly 900 seconds using the old all-to-all KNN code compared to less than a second for the restructured grid-based version, where both codes use k=1 (considering nearest water only) and with a restriction to immediate neighbour voxels (±1 in the x, y, and z directions) only in the grid-based case. The difference in the resulting entropy is less than 4%.

Shared-memory Parallelisation

This restructured code was however still serial, and could therefore not make efficient use of ARCHER compute nodes. It is in principle possible to modify the code to run concurrent but completely independent instances of the executable in such a way that each instance computes entropy contributions from a subsection of the grid following division of labour through instructions given to it at run time, with the results recomposed afterwards. Indeed this approach was taken to run the old code on the Darwin cluster. However this is not possible on ARCHER as the Application Level Placement Scheduler's aprun command does not allow for the execution of concurrent serial instances of an application on a single node. There was therefore a clear need to consider a parallel programming model such as OpenMP or MPI in order to enable the application to run at all efficiently on ARCHER. Moreover, introducing a parallel programming model into the code allows for much more flexibility than a static decomposition, with potential for load balancing, tuning of parallelism, and extension of the code beyond what can be dealt with through a simple static decomposition.

Although the initial project proposal set out the intention to introduce a distributed-memory parallelisation using MPI, it was decided to proceed instead with a shared-memory parallelisation of the grid-structured code using OpenMP, for a number of reasons:

(a) OpenMP was considered easier and quicker to implement incrementally as fewer modifications need to be made to the code and no thought needs to be given about how to decompose the problem and distribute the DCD data as would be needed in a distributed-memory approach. It is arguably also easier to debug, and provides a number of possible ways to tune parallel performance either through environment settings at runtime (e.g. loop scheduling) or by minor modifications to directives.

(b) For the foreseeable future, and certainly for the near future, the majority of use cases were thought to involve analysis of DCD files that should fit into single-node memory (64GB to 128GB available on ARCHER) in their entirety.

(c) Memory profiling of a more sophisticated version of the IFST algorithm revealed memory usage to be a bottleneck. Whilst some memory-intensive bookkeeping variables could be eliminated by restructuring the code to use a three-dimensional vector of Voxels as opposed to the current one-dimensional vector of Voxels, it was deemed that one shared-memory instance with N threads would be less likely to hit the single-node memory bottleneck than N distributed-memory instances, which would all likely be duplicating some data.

(d) It avoids the need to resolve contention issues on reading DCD files, which was originally anticipated to be necessary and planned as Work Package 4.

OpenMP Implementation

The computationally intensive section of the restructured serial IFST code executes the KNN algorithm after the DCD file has been read in and its water molecules distributed over the grid's voxels. This section starts with three nested loops over the x, y and z dimensions of the grid, and is therefore naturally parallelised with an **omp for** construct (embedded in an **omp parallel** region):

```
#pragma omp for collapse(3) schedule(runtime)
for (unsigned int xvoxel=0; xvoxel < xvoxels; xvoxel++)
      {
        for(unsigned int yvoxel=0; yvoxel < yvoxels; yvoxel++)
          {
            for(unsigned int zvoxel=0; zvoxel < zvoxels; zvoxel++)
              {
                  .
                  .
```

In order to obtain good performance it was found to be essential to use the **collapse** clause to collapse the iterations of all three loops into one larger iteration space of tasks, which are then assigned to available threads in accordance with the specified scheduling policy[3]. This amounts to one thread for each voxel tackling the computation of k nearest neighbours and entropy contributions for waters located in that particular voxel.

The runtime schedule was chosen to allow for easy experimentation with different thread scheduling policies (static, dynamic, guided, or auto) without needing to rebuild the code by setting the environment variable OMP_SCHEDULE. If OMP_SCHEDULE is not defined at runtime the default thread scheduling behaviour depends on the compiler. The GNU compiler, typically used to compile Solvaware, defaults to dynamic scheduling with a chunk size of 1 in case OMP_SCHEDULE is undefined.

In order to avoid a performance penalty due to synchronisation on writes to shared variables, which would need to be protected, the contributions to the system's total entropy as well as to the spatially dependent distributions of entropy and density were written to threadprivate variables inside the parallel loop and manually reduced to the shared variables in a critical region following the parallel loop. Results were verified to be correct.

[3] OpenMP API, version 4.0 - July 2013 – sections 2.7.1 and 4.1.
Available at http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

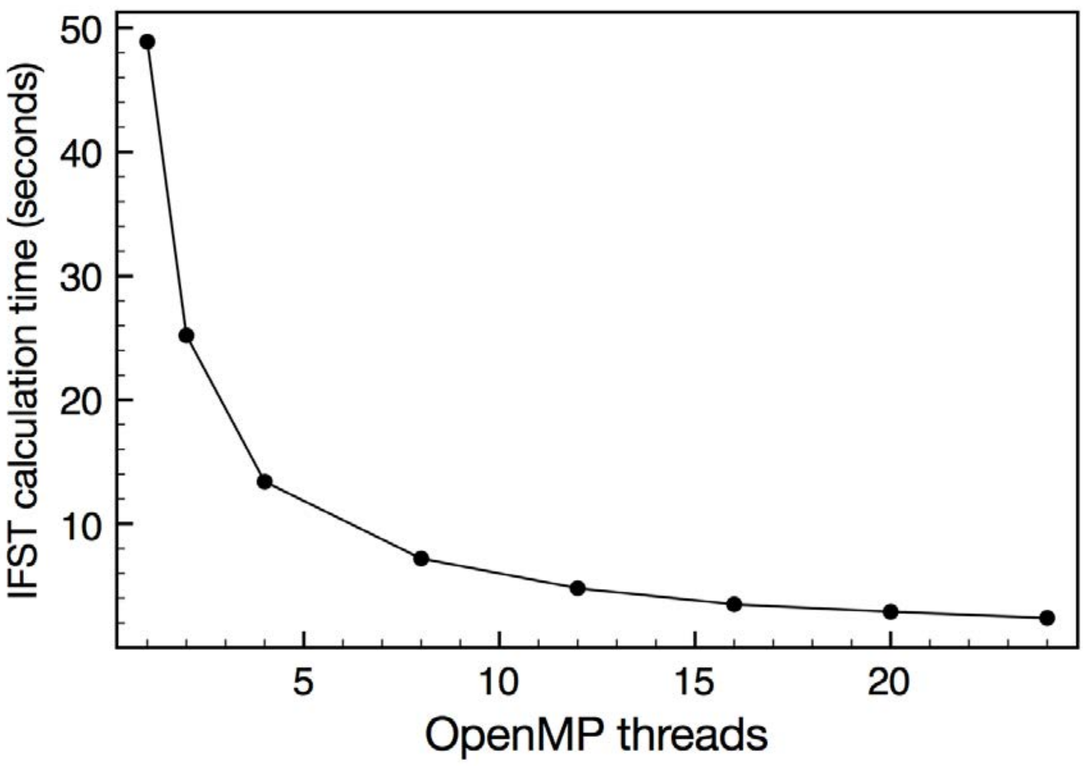


Figure 4 - walltime spent in parallelised IFST calculation for cb7amo benchmark. Analysing 1000 DCD frames, with OMP_SCHEDULE not set hence corresponding to dynamic thread scheduling with chunksize 1 according to the GCC default.

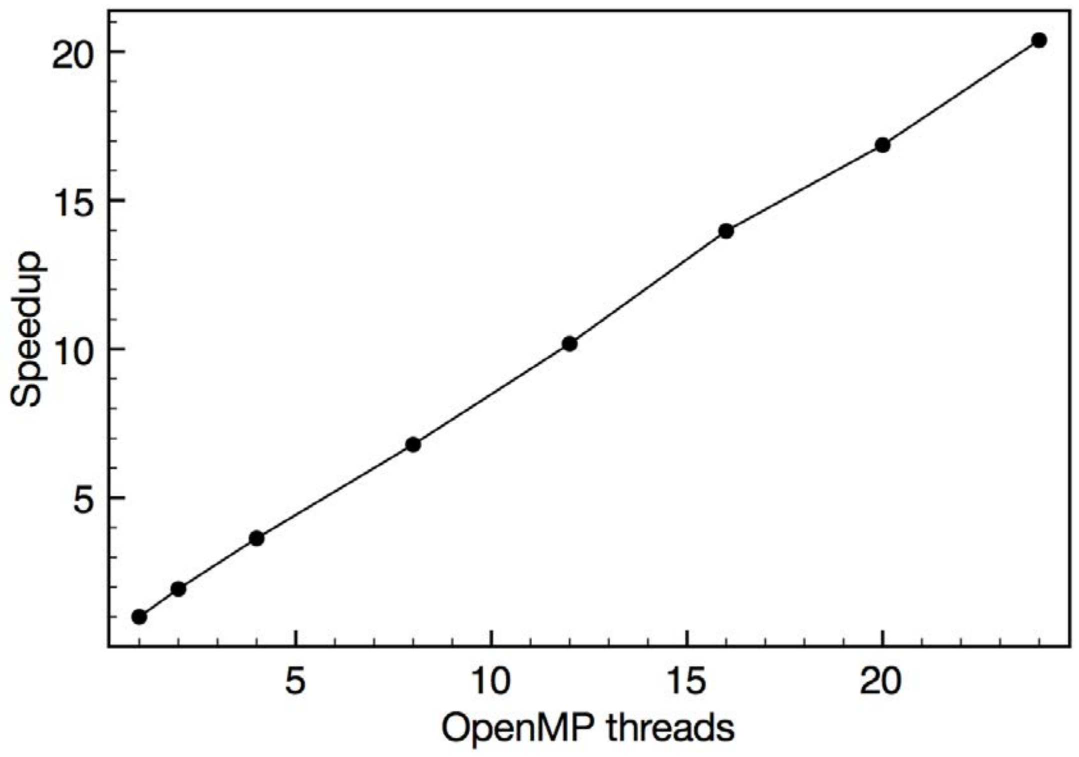


Figure 5 - Speedup of parallelised IFST calculation on ARCHER for cb7amo benchmark (same data as figure 4).

To summarise, the KNN algorithm and IFST calculation as a whole have been optimised serially, and parallelised to make efficient use of ARCHER. Timing results for the parallelised IFST calculation show excellent scaling – see figures 4 and 5 – with parallel efficiency of 85% at 24 threads.

WP3 Implement k-means clustering

A new utility has been added to the repository called ClusterDensityByKMeans. This utility takes uses the following inputs.

- The system PSF and PDB files generated in stage 2 of the workflow in figure 1
- The DCD file generated in stage 4 of the workflow in figure 1
- A user defined number of frames.
- A user defined hydration site radius
- A user defined region of interest, specified by a point in 3D and a radius

The utility generates a PDB file containing atoms at regions with a high number density of water. The clustering was achieved using a k-means approach. More specifically, the water positions from each frame were overlaid to create a density profile. These points were then clustered using the k-means algorithm, exiting when all water positions were within the user defined hydration site radius from a cluster centre. The resulting PDB can be visualised in VMD or an alternative molecular visualiser.

WP4 Optimise I/O of Molecular Dynamics trajectory

The project proposal for this work package was based on a concern that each instance (e.g. MPI process) of the analysis software reads the same 3GB-10GB NAMD trajectory and that this would cause I/O file contention issues if they all attempt to read at the same time. Options considered were to stagger the reads, split up the DCD files, or pre-process the trajectories. However as discussed in the section on work package 2, the shared memory approach adopted for the IFST analysis resolves the concern over potential file contention issues.

WP5 Extend functionality of Solvaware package

A number of further enhancements were made to the Solvaware package in accordance with the intended goals set out in the project proposal. These include:

- The addition of an additional water model (TIP3P).
- The addition of another forcefield (CHARMM36)
- The facility to perform energy minimisation on the solute in preparation for MD simulation

- Automated analysis of MD trajectory quality fed back to the user as part of the workflow. Example output of the temperature and energy is shown in Figure 6 and Figure 7 respectively.

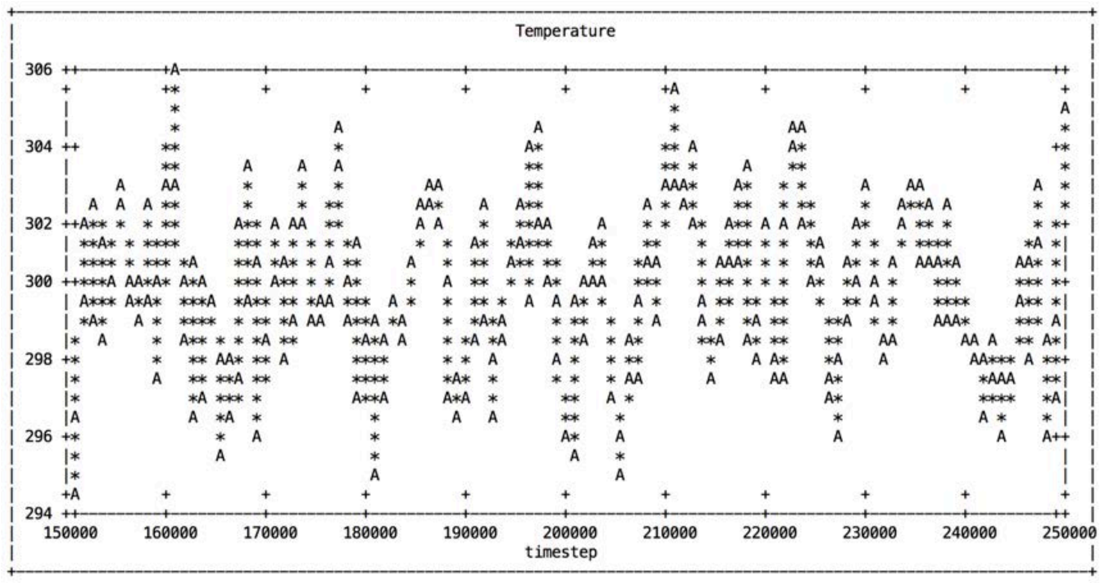


Figure 6 – Output of the temperature from the automated analysis of MD trajectory quality

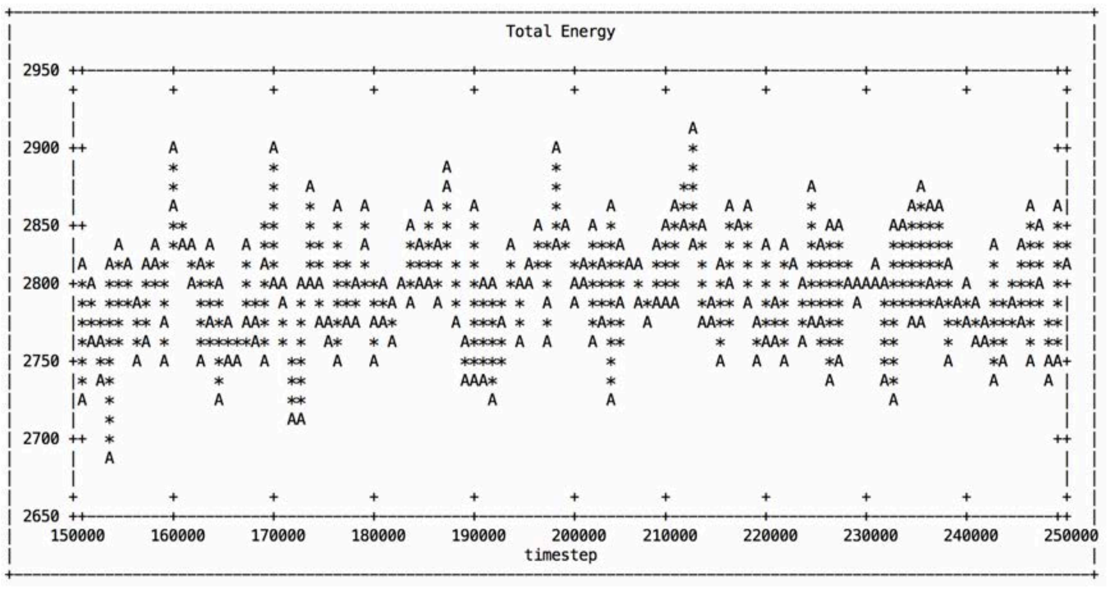


Figure 7 – Output of the energy from the automated analysis of MD trajectory quality

4.5 Conclusion

Solvaware has been ported to ARCHER, giving the ARCHER community access to high-quality software for the analysis of biomolecular hydration. The software has been parallelised using OpenMP, the efficiency has been dramatically increased, and the functionality has been improved.