Parallel Scientific Computing - Report

Step 1 - Multiple Free Objects

The time-stepping method used is velocity Verlet. It has a complexity of $O(N^2)$ as all pairs of forces are computed for gravity, which is a long-range interaction. In practice, we reduce the number of force computations by half by taking advantage of symmetry/Newton's 3rd law. The asymptotic complexity, however, remains the same.

To reduce the complexity we could implement numerical algorithms that approximate the force computations. Examples include Barnes-Hut and Fast Multipole Method.

Both algorithms insert the particles into an octree data structure. The Barnes-Hut simulation uses the structure to logically divide the space into 3-dimensional cells. It approximates the naive result by assuming that exact forces between particles in distant cells do not need to be computed. Instead, the forces between all pairs of particles are approximated as the single force between the cells' centres of mass. It has order $O(n \log n)$. Fast Multipole similarly treats particles that are close together as a single body, and computes forces in linear time via a multipole expansion.

The convergence behaviour of the implementation is tested by taking a two-body task and measuring the distance of the first body's reported final position from the correct result. The correct result is determined via numerical approximation with a very small time-step. The rolling mean of the error values, for varying time-step sizes, is shown in Figure 1. The observed result is not the expected $O(\Delta t^2)$.

Step 2 - Short Range Forces

The code for Step 2 implements the variable linked cell algorithm. The constant r_{cut} has been set to 2.5 and $\alpha = 1$, though this can be changed depending on the scenario. The domain is set as a $10 \times 10 \times 10$ cube with the origin at its center. This can be easily changed if required.

The implementation uses the linked-list methodology: In addition to the existing arrays for position, force, and velocity, two arrays of pointers have been added. The first represents the cells and holds the pointer to the first particle of each cell (the head of its linked list). The second array holds one pointer from each particle to another, representing the tail of the linked list. These arrays are updated as needed when a particle crosses to a new cell.

The linked-list approach has been chosen on the assumption that particles cross into new cells at a reasonable frequency. This approach is also appropriate for very fast-moving particles. If the particles only rarely crossed cells, because they mostly vibrate around their position, a contiguous data structure with all the particles of each cell stored in sequence would be more efficient.

Step 4 - Instruction-level Parallelism

The parallelised N-body simulation was tested on a Hamilton 7 parallel node. Each node has 24 CPU cores (Intel Xeon E5-2650v4, capable of 32-byte vectorisation) and 64GB memory. The simulation was compiled using GCC and a single, heavy simulation of 1000 particles with step size 0.0001 was used for every test. In-code time measurements of the task's execution were taken for increasing number of threads, from $1 - 24$. The results are shown in Figure 2 alongside an ideal strong-scaling model. The code performance closely matches the model with f, the parameter controlling the proportion of the program that must be run serially, set to 0.06.

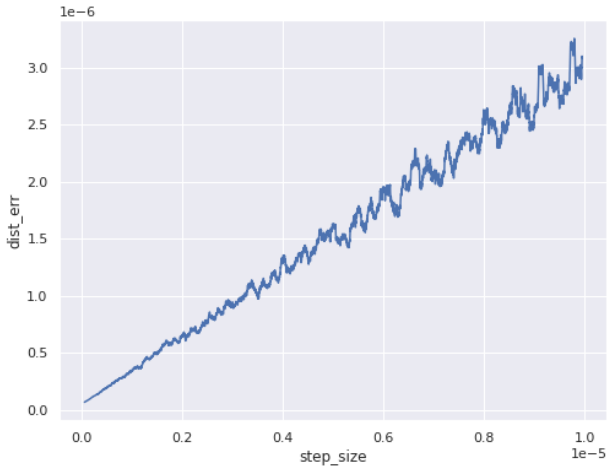

Fig. 1. Global error of Step 1 implementation. To reduce noise the rolling mean is shown.

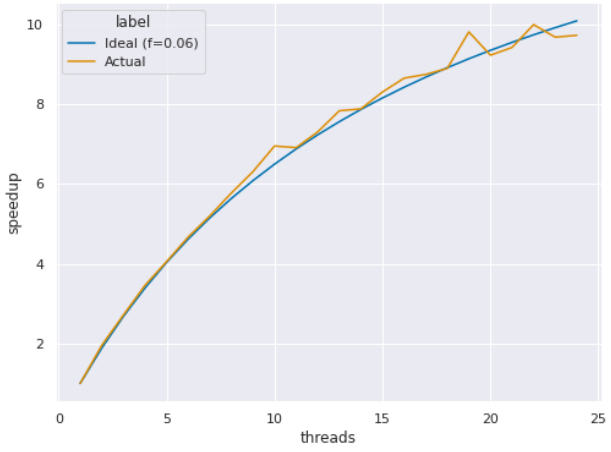

Fig. 2. Scaling behaviour of Step 4 implementation. Shown compared to an ideal Strong Scaling model with $f = 0.06$.