PH-M31 Assignment Question 6

Tom Bourton

E-mail: `701329@swansea.ac.uk`

1. Question 6 - State how the serial code should be modified if we want to write an OpenMP version of it.

If we want to modify the serial code into OpenMP we need to include the OpenMP libraries, and make use of the OpenMP work sharing constructs. We first need to add in a omp_set_num_threads(n) line to set the number of processors to n. Then, inside main we should enclose the section with a #pragma omp parallel, in the arguments of this we need to specify which variables we want to make shared (each thread accesses one global copy of the variable) and which we want private (each thread has it's own copy of the variable). Then we need to create a variable to assign an id to each thread using omp_get_thread_num(). We also need to initialise the random number generators, each thread should be given it's own random seed. We also need to define a chunk-size by taking the lattice size and dividing by the number of threads. Then we need to initialise the lattice of spins, we can add in a #pragma omp for loop to do this in parallel, #pragma omp atomic write can be used to ensure that multiple threads will not be trying to write to the same piece of memory at any point. We can calculate the Boltzmann weights serially so can use a #pragma omp single to do this.

It is also possible to do the updates to the lattice in parallel using a #pragma omp for loop, however there will be problems with threads trying to read/write to the same memory lattice points when on the boundaries, therefore we can define two checkboard style arrays, which should be made global, we can create this by allocating two size*size lattices, and then running over all points and taking (i+j) mod (2) where i,j are the lattice coordinates, therefore this will create a checkboard style array, we want one array where all white, even points = 1, odd black points = 0, and another where all white points = 0, black points = 1. Also a variable is required to keep track of the checkboard status, where checkstatus = 0 will correspond to the whitelattice points, checkstatus = 1 corresponds to the blacklattice. Then do the updates in parallel, first by comparing with the white checkboard and updating the corresponding points and then flipping the checkboard status and doing the updates with respect to the black checkboard, this will ensure that the 4 surrounding nearest neighbours for any lattice point, for any thread will always be free to be read from.

We can then parallelize the measurements function, again using an omp for loop and reusing the checkboard to ensure that we do not get race errors again where both processors would be trying to read/write the same piece of memory at the same time.

Now, we have Nproc arrays of the observables, so we can define a new totalobservables[measurements] to store the reduced total observables values, then we can define an omp critical region where we do a for loop and loop over the number of measurements, e.g. enetot[i]+=ene[i], this way we can reduce all measurement values from each thread into one array. Then end the parallel region, and divide each measurement in the total observable arrays by the number of processes. Then we perform the statistical analysis serially, as in the serial code, as again there is no huge benefit to parallelizing this region of code as it should not effect computation time too much.