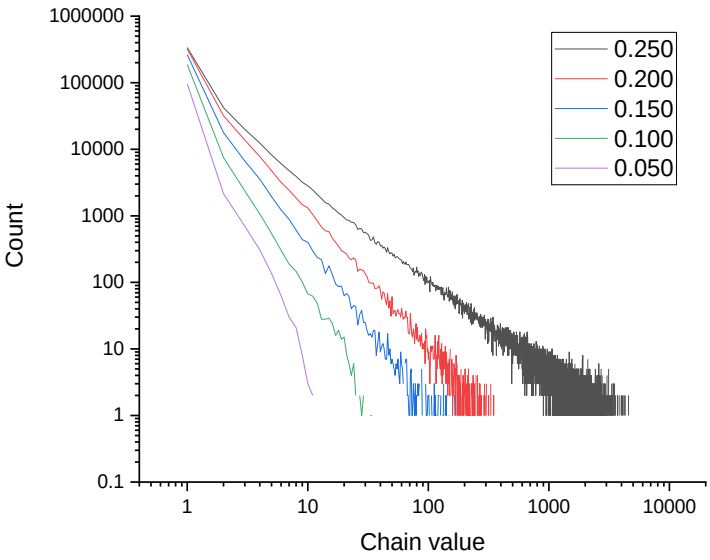
Assignment 5 Report

COMPLETE THE FINAL STEP OF THE ALGORITHM

Added the following lines to distribute force of earthquake to four nearest neighbours in non-edge cases, and then set check if this caused a new earthquake.

```
if (iy<Ny-1) {
  force[iy+1][ix] += alpha*oldforce[iy][ix];
  if (force[iy+1][ix] >= threshold) quake = 1;
}
if (iy>0) {
  force[iy-1][ix] += alpha*oldforce[iy][ix];
  if (force[iy-1][ix] >= threshold) quake = 1;
}
if (ix<Nx-1) {
  force[iy][ix+1] += alpha*oldforce[iy][ix];
  if (force[iy][ix+1] >= threshold) quake = 1;
}
if (ix>0) {
  force[iy][ix-1] += alpha*oldforce[iy][ix];
  if (force[iy][ix-1] >= threshold) quake = 1;
}
```

INITIAL PLOT



Initial graph for the 5 alpha values (0.050, 0.100, 0.150, 0.200, 0.250). With grid size: 35 x 35, and 100,000 steps. Noise on tails of data due to low number of steps.

PARALELLISING STEPS LOOP WITH MPI

Decided to use this strategy rather than a grid-based one due to grid needing a more complex topology to avoid limitations on grid size as well grid-based needing far more communications overall leading to more overheads and waiting to synchronize. (Many MPI_Sends and MPI_Recvs versus a single MPI_Reduce)

Added comms routines to initialise and finalise MPI, storing current rank and total ranks in **my_rank** and **p**.

Changed seed for random number generation to **my_rank** times a constant, to ensure that the same results aren't reproduced on every rank.

Cameron Fantham

Change limits on the steps for loop to istep=my_rank*Nsteps/p and istep<=(my_rank+1)*Nsteps/p, from istep=0 and istep<=Nsteps, to ensure each rank has an equal number of steps and these steps are ordered in terms of rank.

Added hist_global array and MPI_Reduce call after steps loop to sum elements of hist into corresponding elements hist_global on rank 0:

MPI_Reduce(hist,hist_global,histsize,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);

Write hist_val and hist_global to output file only on rank 0 with if (my_rank==0) loop.

Also edited naming convention of snapshot files to include alpha and my_rank in order to distinguish between them more easily.

With 48 tasks this runs the initial 35 x 35 grid etc. settings in 0.436318 seconds, compared to 9.827369 seconds running serially.

PARALELLISING STEPS LOOP WITH OPENMP

Added the following OpenMP pragma to steps loop:

#pragma omp parallel for default(none) private(istep,ix,iy,ichain,ih,filename) shared(p,isnap,maxchain,quake,maxforce,my_rank,Nsteps,Nx,Ny,alpha,histsize,histp,histval,force,oldforce,hist) schedule(static)

Allocating multi-threads for each rank could speedup runtimes, this will be tested against just using MPI only.

TIMINGS

Set -a 0.05, -n 5, -X 60, -Y 60 and -N 1000000

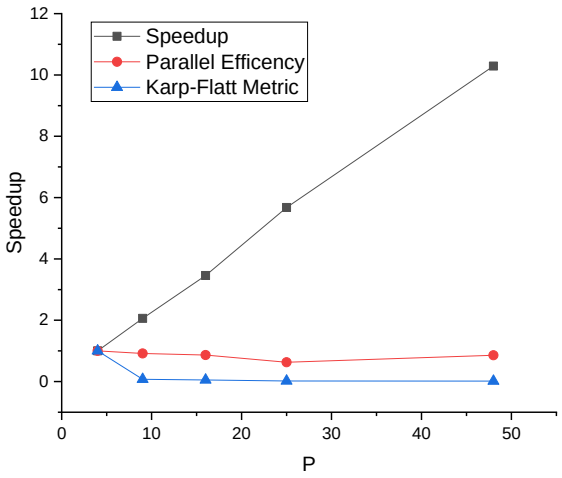
Timings from both OpenMP and MPI are used for each steps loop completed on each rank/thread, as well as time to complete all alpha values on each rank/thread. This is done to check for synchronization of ranks and load balance and helps ensure accurate timings. (P = Ranks * Threads)

	P=1 1*1	P=4 4 * 1	P=9 9*1	P=16 16*1	P=25 25*1	P=48 48 * 1	P=4 1*4	P=16 4 * 4	P=48 12*4	P=16 8 * 2	P=48 24 * 2	P=48 4 * 12
Total run time (s)	>120	87.176	42.334	25.218	15.353	8.472	>120	63.695	20.839	44.937	15.184	82.421
Speedup	N/A	1	2.059	3.457	5.678	10.290	N/A	1.369	4.183	1.940	5.741	1.058
Parallel efficiency	N/A	1	0.915	0.864	0.631	0.858	N/A	0.342	0.349	0.484	0.478	0.088
Karp-Flatt metric	N/A	1	0.074	0.052	0.019	0.015	N/A	0.641	0.170	0.354	0.099	0.940

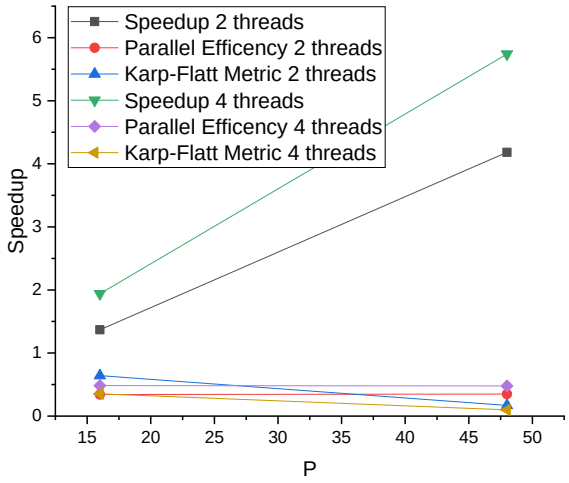
Speedup compared to P=4, as P=1 could not run in the 2-minute time limit. Any more than around 4 OpenMP threads tends to increase the runtime rather than decrease it.

Load balance:

From the individual timings for each rank it is clear there is some variance depending on how far the earthquake loop for a certain step goes, for example if one rank had more steps that had very long chains than another rank then more processing power should go to that task than the others in order to eliminate discrepancies in run times for each task. It might also be a worth while strategy to run the first 4 alpha loops in parallel with the 5th as it usually takes longer than all the rest combined.



Graph for MPI Only (1 thread)



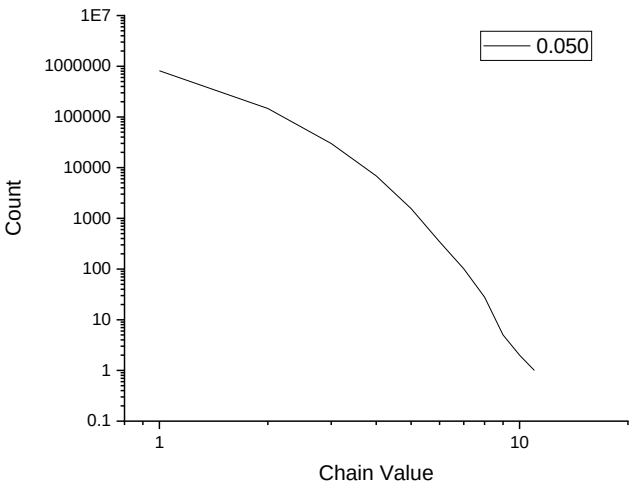
Graph for 2 and 4 threads

Clearly the parallel program runs much quicker using the equivalent P for only MPI rather than OpenMP + MPI. In fact, running with only 1 MPI rank but multiple OpenMP threads showed decreasing runtimes with increasing number of threads. Conversely, running with 4 ranks and 4 threads did run faster than using 4 ranks and 1 thread, so more threads do give some benefit at certain numbers of ranks. This could be due to increase in parallel overheads of running more OpenMP threads with MPI, competing against the speedup from having more threads, as well as sections within the steps loop that are serialised. Therefore, it is best to use only MPI and no OpenMP threads as the data structuring does not scale well to higher OpenMP threads.

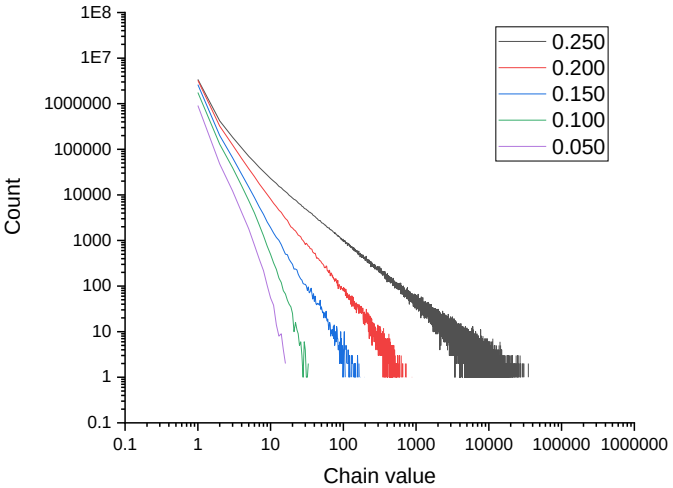
TESTING SIZE LIMIT

Set -a 0.05, -n 1 and -N 1000000

Maximum grid size was 525 x 525, with time 119.785694 seconds. Achieved using 48 MPI tasks across 3 nodes.



Graph for grid size 525 x 525, alpha 0.050



Graph for -a 0.05, -n 5, -X 120, -Y 120 and -N 1000000