

Assignment 4—Simulated Annealing

This assignment is due at class time Wednesday, 17 November 2004 and is to provide you with experience doing a simulated annealing problem in MPI.

The basic assignment is Quinn's problem 10.10, page 272.

You should run this on two processors and on four processors.

You should try debugging this program on a small array, perhaps only 5×5 so that if you have to, you can compute the correct answer by hand. You are free to use the pseudocode of Figure 10.16 as a baseline.

You are free to use any random number generator you wish, but you should document carefully which one you are using. The built in `rand()` function would be perfectly acceptable.

Be *absolutely sure* that you do not generate the same sequence of random numbers in every processor.

You should time your computation. In addition to timing the computation itself, you should keep track of how many random numbers you use and you should have a routine that allows you to time the generation of random numbers. Your documentation in the code and comments printed in the output should include the results of this timing (number of random numbers per second generated) so that someone reading the code or the output would be able to know what fraction of the overall time is spent just in generating random numbers and what fraction is being spent doing more relevant computation.

Do *not* put a timing call before and after every call to the random number generator. Timing calls are expensive. The cost of generating random numbers should be roughly linear with the number generated, so if you time the generation of 1 million and ten million such numbers, say, you can take the difference and divide by nine to get the average cost of generating one million numbers and the overhead involved in doing the function calls.

Variation: If you finish early, and you should (assuming you start early), you might also try the following. This is a computation that attempts to explore a too-large space by a pseudo-random walk. Try the effect of swapping large chunks of your array. The assignment is to run the code for one million iterations. Every 100,000 iterations, say, try exchanging 1/2 or 1/4 of the array with a neighboring processor. This will require recomputing the energy function from scratch, so there's a penalty for doing this. But maybe you'll get better values of the objective function, and it will be worth it.

As another variant, since recomputing the entire function is costly, is to exchange a random 2×2 or 4×4 block. Recomputing this is cheap.