Parallel Programming Exercise 4-Dec-17

Exercise 1: False Sharing

Implement the manual reduction code below. How well does it scale to multiple cores? Is the scheduling of the parallel for loop important? Rewrite the code so that it scales better. What was the problem?

```
double local_sum[nthreads]; //initialize this to 0
[...]
#pragma omp parallel num_threads(nthreads) private(i)
{
   [...]
   for (i = id; i < N; i += nthreads) {
      local_sum[id] += a[i];
   }
}

for (i = 0; i < nthreads; i++) {
    sum += local_sum[i];
}
```

Exercise 2: Parallel Algorithms

Write a parallel program to compute the natural log of the factorial of the numbers 1 through N and store them in a vector. Do something which runs much faster than:

```
for (i = 1; i < N; i++) {
   factorials[i] = 0.;
   for (j = 1; j <= i; j++) {
      factorials[i] = factorials[i] + log(j);
   }
}
```

Note that a double has a maximum value which you will quickly reach if you try to store the products of many numbers. This is why the code above takes a sum-of-logs approach to the problem.

What would you do differently if instead you wanted to find the factorial of each element of a vector a, where the elements are random integers between 1 and 10? That is, a is initialized as:

```
for (i = 0; i < N; i++) {
   a[i] = rand() % 10 + 1;
}
```