September 24th, 2015 Pre-Class Questions

Elliot Cartee

Question 0

I started these questions on September 23rd. I spent slightly less than an hour on them (not counting viewing/listening to the slides)

Question 1

I found the more architecture-based material in the slides (i.e. the snoopy bus protocol) to be less clear, but this is likely because I have less of a background in computer architecture than many in the class.

Question 2

(2a)

Let's suppose that we run a Monte Carlo simulation with N the number of trials, p the number of processors, (t_{trial}) the time per trial , and (t_{update}) the time to update the global counters in the critical section . Let us also suppose that we are using a batch size of b.

Then the total amount of time spent running the trials will be:

$$\frac{N}{p}t_{trial}$$

While the total amount of time spent updating will be:

$$\frac{N}{b}t_{update}$$

therefore the total execution time will be:

$$\frac{N}{p}t_{trial} + \frac{N}{b}t_{update}$$

This is a rather simplistic model, and doesn't do very well in the limit as the batch size b approaches the problem size N, where a lot more than N trials might be run before the code checks for convergence.

(2b)

When I ran the OpenMP code on the cluster, I got the timings found in the `timing_omp.svg` file in this directory, and estimated t_{update} and t_{trial} as:

$$1.34760680e-07 \text{ and } 1.36498346e-08$$

(2c)

Based on my model, you are best off picking a very large batch size. As mentioned before, my model is rather simplistic and falls apart when considering large b, as in this case you may be doing a lot more than N trials. So the best choice of b should be one that is large, but still small enough compared to $\frac{N}{p}$ that you will not have to do too many extra trials. In fact, if you know N and p beforehand, you could use this to automatically choose a batch size that is large, but still small relative to $\frac{N}{p}$.

Question 3

One way the code performance could be improved (although not mentioned explicitly in the OpenMP pitfalls paper), would be to set `sum_X` and `sum_X2` as reduction variables, rather than adding up the partial sums in a critical section.

Also, the two critical sections could be named differently, since their is no reason for adding up the partial sums to wait for another thread to seed the random number generator, although this should not make much of a difference.

I'm not sure that any of the critical regions could be changed to atomic regions, but if we could do that it would help.

The code doesn't seem to do any flushing, but since we are using a much later version of OpenMP compared to the writing of the OpenMP pitfalls paper, I think this is no longer such an issue.

We could also add in something to check if the program has converged before entering the critical region.