Monte Carlo Integration Parallelization Report

Alejandro Servetto

March 18, 2021

1 Methods

I did the Monte Carlo techniqe.

I used the pthread library for my thread implementation. The way I parallelized was to break up the summation. For `N` samples and `Nt` threads, I had each thread responsible for `N/Nt` samples of the summation. This was handled by a subroutine `sum_portion(float a, float b, int portion_size, int n)` where `portion_size` is `N/Nt`. Each thread would enter its own little subroutine, do its summation (with a thread safe random number generator) and write its result to a heap-allocated array. Each thread was assigned an index of the array to write to. The parent would wait for each baby thead to complete, look up the result in the global array, and add it to the sum. When all threads completed and all of the results were added to the sum, I normalized by multiplying `sum*(b-a)`. Return result and print to stdout.

2 Results

I have access to a 40 core machine for my research, so I ran these on that. I picked arbitrary numbers to do the integral on and for the sample size. Then did one run on each thread, recording the results. You'll notice that there was a few times where a runtime was slightly faster than the previous...I'm writing that off as sampling error, since it was only one trial each, and sometimes the OS prioritizes a given process less frequently, given other external cicrumstances. This would probably not be an issue if I and multiple iterations of each thread count and figured out the averages.

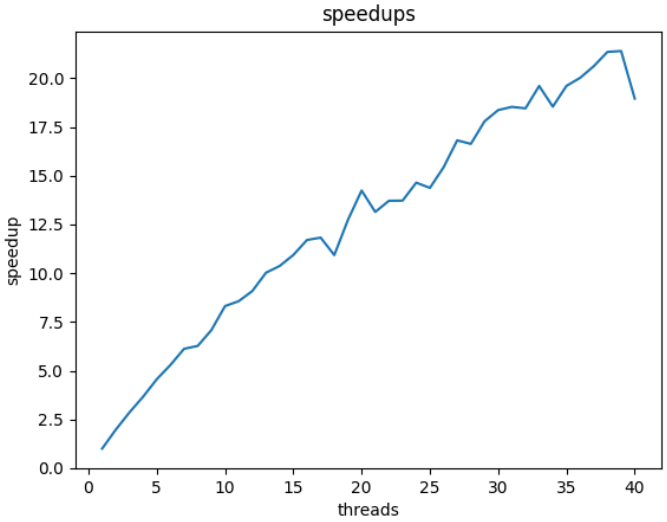

Figure 1: Speedup

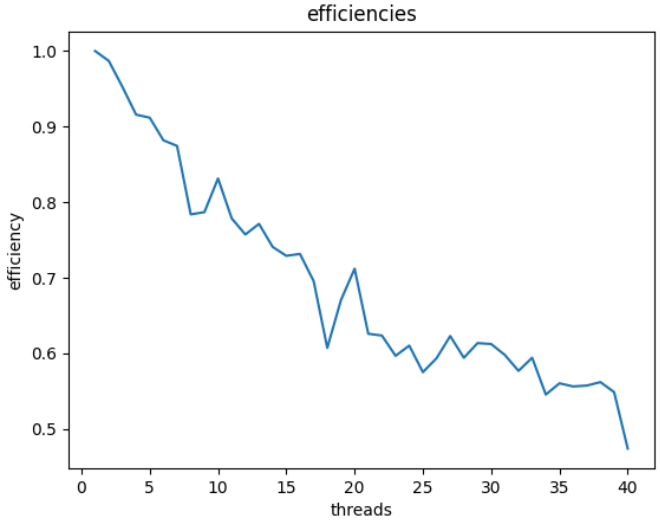

Figure 2: Efficiency

3 Discussion

Looks like the speedup is linear, but not one-to-one. For example, at 20 threads, you only get around a 15 times speedup. And it gets worse the more threads you add. This is also demonstrated with the efficiency; it decreases with every added thread, although it stays linear. So while each thread provided an additional speedup (minus error explained above, likely because of exernal OS scheduling circumstances for a given process), each thread was less helpful in making the computation faster.