EECS 587 Homework Problem

Due at start of class, 17 October 2024

Once again you are allowed to work in teams of 2, with the same grading rules as before.

For this problem you will compute $\max\{f(x) : a \le x \le b\}$ for a given real-valued function f and endpoints $a < b$. In general it is impossible to find the maximum exactly, so instead you will be given an $\epsilon > 0$ and need to find a value within ϵ of the true maximum. This too is impossible in general, but here is simplified by the fact that f is differentiable and you are given a bound s on the absolute value of the derivative.

You are to solve the problem by examining intervals, determining if they could containing a value larger than one that you already have found. If not then you discard the interval, and otherwise look at it more closely. For example, suppose you currently have found a value M that f attains, and are examining an interval $[c, d]$ to see if it might have an even larger value. First set $M = \max\{M, f(c), f(d)\}$. With a little algebra one can show that in the interval $[c, d]$, f can be at most $(f(c) + f(d) + s(d - c))/2$. (Consider a straight line passing through $(c, f(c))$ with slope s, and one passing through $(d, f(d))$ with slope $-s$. The maximum possible is the intersection of these two lines.) If this value is less than $M + \epsilon$ then you need not search this interval any further. Otherwise divide the interval in half and examine $[c, (c+d)/2]$ and $[(c+d)/2, d]$.

You are to start with the interval $[a, b]$, set $M = \max\{f(a), f(b)\}$, and continually apply the above procedure until you've found a value guaranteed to be within ϵ of the true maximum. You are only allowed to generate candidate intervals by the above method; for example, you are not allowed to start by dividing $[a, b]$ into many subintervals, but rather must divide it in half, then examine each half and divide the half in half (if it can't be eliminated), etc. With only a few iterations you should be able to generate enough subintervals to keep the workers busy.

Write your program to work with arbitrary values of $a < b$, $\epsilon > 0$, $s > 0$, and arbitrary function f. All calculations are in double precision. In a couple of days we will supply the function and you are not allowed to modify it. The values of a, b, ϵ and s that we give you, and the function itself, might change, depending on how long the problems are taking. Initially you should just debug your problem with a simple function of your own choosing.

Using OpenMP, run your program on the Great Lakes computer for p =9, 18, and 36 cores. When submitting a slurm job for this assignment make sure to specify

#SBATCH –exclusive

in order to make sure your job occupies an entire node and is not slowed down by other programs.

You must submit your programs with an upper limit of 5 minutes or less.

As before, write a short report reporting the value of the maximum, briefly explain how you parallelized the problem, and analyze the timings obtained. Turn in the program as well. Timing should start just before the first interval is divided, and stop after the maximum has been computed. You are not allowed to use the OpenMP TASK construct.

After you have turned in your homework, we *might* do the following (we haven't decided yet): you will then be given a second set of parameters and functions and will run your program on this, turning in the timings. You are not allowed to alter the program for this second run. Therefore if you overly optimize the parallelization for the first function you might do poorly on the second function.

Note: there are two main search algorithms for such a problem: depth-first and breadth-first, which use a stack and a queue, respectively. In depth-first search (DFS), whenever you subdivide an interval you then examine one of its subintervals. I.e., suppose you start with [0,1] and need to subdivide several times. Over time, the stack of pending subintervals needed to be examined might look like:

$[0,1]$

$[0,\frac{1}{2}]$, $[\frac{1}{2},1]$

$[0,\frac{1}{4}]$, $[\frac{1}{4},\frac{1}{2}]$, $[\frac{1}{2},1]$

$[0,\frac{1}{8}]$, $[\frac{1}{8},\frac{1}{4}]$, $[\frac{1}{4},\frac{1}{2}]$, $[\frac{1}{2},1]$

$\ldots$

$[\frac{1}{8},\frac{1}{4}]$, $[\frac{1}{4},\frac{1}{2}]$, $[\frac{1}{2},1]$

$[\frac{1}{8},\frac{3}{16}]$, $[\frac{3}{16},\frac{1}{4}]$, $[\frac{1}{4},\frac{1}{2}]$, $[\frac{1}{2},1]$

$\ldots$

In breadth-first search (BFS) you always divide the largest interval remaining, i.e., the queue of pending intervals might evolve like:

$[0,1]$

$[0,\frac{1}{2}]$, $[\frac{1}{2},1]$

$[0,\frac{1}{4}]$, $[\frac{1}{4},\frac{1}{2}]$, $[\frac{1}{2},1]$

$[0,\frac{1}{4}]$, $[\frac{1}{4},\frac{1}{2}]$, $[\frac{1}{2},\frac{3}{4}]$, $[\frac{3}{4},1]$

$[0,\frac{1}{8}]$, $[\frac{1}{8},\frac{1}{4}]$, $[\frac{1}{4},\frac{1}{2}]$, $[\frac{1}{2},\frac{3}{4}]$, $[\frac{3}{4},1]$

$\ldots$.

An advantage of DFS is that the stack will never be very large, while in BFS it can be quite large. An advantage of BFS is that it easily parallelizes, while DFS does not. These can be mixed, e.g., using BFS to generate a large number of subproblems, and individual cores using DFS on a subproblem.

Typically DFS is implemented via an array D(0:k-1), where the bottom of the queue is at D(0) and if the top is currently at D(i) then adding an item (push) puts it at D(i+1), and removing the top (pop) moves the top to D(i-1). Typically BFS is implemented via a circular array B(0:k-1), where if the front of the queue is at B(i) then removing an item changes the front to (i+1) mod k, and if the back of the queue is at B(j) then adding an item changes the back to (j+1) mod k.

A modification of the queue-based approach is to use a priority queue. For example, you could give a priority to each interval in terms of the largest possible value it might have, and always search the one having the largest of these. However, priority queues are similar to depth-first search in that they are difficult to parallelize.