EECS 587 Homework Problem

Due at start of class, 13 November 2014

As for all homework in this class, you must do this problem on your own.

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. The problem is further 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. Your function must be accessed by a function call, i.e., you cannot speed it up by in-lining it, nor having the compiler do this. You will use $a = 1$, $b = 1000$, $\epsilon = 10^{-12}$, $s = 12$, and

$$f(x) = \sum_{i=1}^{100} \frac{\sin\left(x + \sum_{j=1}^{i} (x + j)^{-3.1}\right)}{1.2^i}$$

These parameters and function may change, depending on how long the problem takes to run. To improve numerical accuracy you should do the summations in reverse order, i.e., starting with $i = 100$ and ending with $i = 1$ and similarly for the j-loop.

Using OpenMP, run your program on the PSC Blacklight computer for $p = 2$, 4, 16, 32, and 64 cores. 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 only need to time it once per value of p.

After you have turned in your homework, 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 this function you may 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, 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:

$[0, 1]$
$[0, \frac{1}{2}]$, $[\frac{1}{2}, 1]$
$[0, \frac{1}{4}]$, $[\frac{1}{4}, \frac{1}{2}]$, $[\frac{1}{2}, 1]$
$[0, \frac{1}{8}]$, $[\frac{1}{8}, \frac{1}{4}]$, $[\frac{1}{4}, \frac{1}{2}]$, $[\frac{1}{2}, 1]$
$\cdots$
$[\frac{1}{8}, \frac{1}{4}]$, $[\frac{1}{4}, \frac{1}{2}]$, $[\frac{1}{2}, 1]$
$[\frac{1}{8}, \frac{3}{16}]$, $[\frac{3}{16}, \frac{1}{4}]$, $[\frac{1}{4}, \frac{1}{2}]$, $[\frac{1}{2}, 1]$
$\cdots$

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

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

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

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.