To: Christina Taylor
From: Anwar Khaddaj
Date: March 9, 2023
Subject: CAAM 420/520 – Homework 3
I collaborated with Arshia Singhal and Jonathan Cangelosi on this homework.

Problem 1 (a) Nodes at the same level in the function tree can be processed at the same time as they can be independently processed.

(b) We need order dependency (order of operations) i.e. we need the children to be computed first in a post-order traversal. Also, for non-leaf nodes, we need to process the children of the node before the node itself which happens to be the operation. Thus, we need synchronization to ensure the lower levels are processed before higher levels.

(c) Submitted as code.

(d) To find the parallel time complexity of evaluating the sum using a function tree, we note that this problem has spin-down and fully parallelized phases, so we need to find the complexity of the each phase and then add up the results.

i. Spin-down phase: That's the case when there are some idle threads. This happens when the number of nodes are strictly less than the number of threads, i.e. when the level L is at most $T-1$ (Level $L = T$ has the same number of nodes and threads, so no threads will be left idle). Since the nodes at the same level in the function tree can be processed at the same time, the complexity of each level in this phase is actually 1. Thus, the complexity of this phase is:

$$\sum_{L=0}^{T-1} 1 = T * 1 = T = \log_2(N_T).$$

ii. Fully parallelized phase: We have more nodes than threads per level, so the complexity of each level in this phase is basically the amount of work per thread, i.e. number of nodes per level divided by the number of threads. Since it is a full binary tree, each level L (0 indexed) has 2^L nodes. Thus, the complexity of each level is $\frac{2^L}{N_T}$. Now, the fully parallelized region starts at level $L = T$ and ends at $L = \log_2(m)$. Therefore, the complexity of this phase is:

$$\begin{aligned}\sum_{L=T}^{\log_2(m)} \frac{2^L}{N_T} &= \frac{1}{N_T}\sum_{L=T}^{\log_2(m)} 2^L \\ &= \frac{1}{N_T}(2^T(2^{\log_2(m)-T+1} - 1) \\ &= \frac{1}{N_T}(2^{\log_2(m)+1} - 2^T) \\ &= \frac{1}{N_T}(2m - 2^{\log_2(N_T)})\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{N_T}(2m - N_T)\\
&= \frac{2m}{N_T} - 1\\
&= \frac{n+1}{N_T} - 1 \text{ (considering } n = 2m-1).
\end{aligned}$$

Therefore, the total time complexity is:

$$\log_2(N_T) + \frac{2m}{N_T} - 1 = \log_2(N_T) + \frac{n+1}{N_T} - 1.$$

(e) Using an OpenMP parallel-for loop implementation is faster than using the function tree for this simple sum problem. It's simply because the sum algorithm is not a complex problem, so using OpenMP's reduction parallel for loop is enough to solve it especially that this method is already optimized to solve this specific task of summing. However, the function tree is a complex algorithm that can complicate solving the problem especially that we have a spin-down phase which might increase the time complexity. Also, this function tree is more fitted to solve complex problems such as one related to having more operations than just the addition operator.
Morever, if the number of threads used is 2, the function tree complexity is 2 times slower than the OpenMP parallel-for loop, and if the number of threads is more than 2, the function tree complexity is more than 2 times slower than the OpenMP parallel-for loop. Thus, in general, the function tree complexity is roughly 2 times slower than the OpenMP parallel-for loop implementation.

Problem 2 Submitted as code.