Homework 31

Brian Knotten, Brett Schreiber, Brian Falkenstein

November 8, 2018

2

To give evidence that finding a fast efficient parallel algorithm (i.e. a $O(log^k(n))$ algorithm for some k with a poly number of processors) for N is at least as hard as finding a fast efficient parallel algorithm for the BFV problem, you could present a parallel algorithm for the BFV problem that is $O(log^k(n))$ for some k with a poly number of processors that is only missing code for N.
The algorithm should convert the input from the BFV problem to the input to the N problem in at most poly-log time, run the missing code for N using the converted input, and convert the output of the N problem to the output of the BFV in at most poly-log time.
Further, the missing code for N should give a correct output given the input iff there is a correct output for the BFV problem given the input.
Given this parallel algorithm for the BFV missing the code for N, if you were to find parallel code for N that runs in poly-log time with a poly number of processors then you would immediately have a poly-log algorithm for the BFV problem with a poly number of processors by "plugging in" the code for N into the parallel code for BFV.
Because the code we are plugging in is poly-log with a poly number of processors, our input/output transformations are also poly-log with a poly number of processors, and the code gives a correct output for BFV iff the code for N gives a correct output, the newly-constructed algorithm correctly solves the BFV problem in poly-log time using a poly number of processors.
Therefore finding a fast efficient parallel algorithm for problem N is at least as hard of a problem as finding a fast efficient parallel algorithm for the Boolean Formula Value problem.

16

17

Break the input into $\sqrt{n}$ chunks of size $\sqrt{n}$ and recursively solve the problem in parallel with $\sqrt{n}$ processors assigned to each chunk. We have n processors, so the $\sqrt{n}$ max values returned by the chunks can be "maxed" in constant time using our n processors (see hint).