

Introducing Three Basic Concepts in Parallel Computation to 1st Year Computer Science Students in A Simple and Effective Way

Sukhamay Kundu
Computer Sc. Department
Louisiana State University
Baton Rouge, LA 70803, USA

and

Haklin Kimm
Computer Sc. Department
East Stroudsburg University of Pennsylvania
East Stroudsburg, PA 18301, USA

Abstract: What are some simple computation problems, with non-trivial applications, by which we can introduce the three basic concepts in parallel computation to 1st year Computer Science students who have little programming experience and knowledge of algorithms. Those three basic concepts are: (1) How parallel computation can reduce an algorithm's completion-time, including methods to determine the completion-time for a given number of computing agents (people or CPUs, for simplicity). (2) How to identify parallelizable parts of an algorithm given in a pseudocode form. (3) How to balance the computation-load of the agents, with perhaps some modification of the algorithm, to further reduce the completion time. We present one such "ideal" computation problem, which has several simple intuitive solution algorithms and multiple ways of parallelization. The results of a short-quiz following 2 one hour lectures covering one parallel algorithm of the ideal problem showed the students achieved a good understanding of the above three basic concepts. One could cover a bit more material in one or two additional lectures. A full discussion of different parallelizations of our ideal problem and their performance comparisons is fit for 2nd year and more advanced students.

Our ideal computation problem. Let Z be an array of 0's and 1's, and we need to move all 0's to the left of all 1's, with as few (ex)changes as possible. For example, $Z = [1, 0, 0, 1, 1, 0, 1, 0, 0]$ has to be converted to $[0, 0, 0, 0, 0, 1, 1, 1, 1]$. If 1's represent apple-trees and 0 peach-trees, then keeping the trees of each kind together helps the fertilizing and harvesting operations. Another application is grouping patients in a hospital with similar infections. If $m = \#(0\text{'s in } Z)$, then the minimum number of exchanges needed is $w_1 = \#(1\text{'s that must move}) = \#(1\text{'s in first } m \text{ positions in } Z)$, etc.

A simple algorithm: Successively exchange k^{th} rightmost 0 and k^{th} leftmost 1 for $k = 1, 2, \dots$, until the exchange-test " k^{th} rightmost 0 is to the right of k^{th} leftmost 1" fails. It uses only w_1 exchanges.

Many possible parallelizations: The writes in an exchange operation (replacing 0 by 1 and 1 by 0) can be parallelized. Likewise, finding a pair of rightmost 0 and a leftmost 1 can be parallelized. To find the k^{th} rightmost 0, we look at the items to the left of $(k-1)^{th}$ rightmost 0; for first rightmost 0, look to left from the last item in Z . Similarly, for k leftmost 1. The three steps (find a pair of 0 and 1 for exchange, do the exchange-test, and the exchange operation if the test succeeds) in each iteration of the do-until loop can be done sequentially. We can alternately do k^{th} exchange operation in parallel with finding $(k+1)^{th}$ pair of 0 and 1 and $(k+1)^{th}$ exchange-test. This way we can use parallelism of degree upto 4. We can use additional parallel operations in finding the k^{th} rightmost 0 and similarly in finding the k^{th} leftmost 1. For a given Z , it is fairly easy to determine the reduction in completion-time (or speed-up) for different combination of the above parallelization schemes.

Other algorithms and their parallelizations: Successively replace 10's by 01's until there is no 10. A simple variation of the algorithm is to replace an 10^n ($n \geq 1$) by 0^n1 using one 0/1-exchange. We can do the replacements in any order and thus do all replacements at any stage of the array in parallel. The patterns 10^n can be searched in parallel by partitioning the array into equal-width blocks (except perhaps one) of consecutive items. Note that an 10^n may be formed by the last and first few items of two blocks and those between them. We can search for the pattern (and replace them) in parallel.