ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

FS 2010 Prof. R. Wattenhofer / C. Lenzen

Principles of Distributed Computing
Exercise 6

1 Shared Sum

In the lecture, we discussed how shared registers can be employed efficiently to allow each process to announce a value to all other processes. Now we look at a different scenario: Each process p_i computes a local variable x_i and we want to make the sum $x := \sum_{i=1}^{n} x_i$ available to all processes.

We want to guarantee the following: If a process updates x_i, it should first ensure that x is updated accordingly before proceeding. However, we do not want to use a large number of registers or a huge register. In the following, you are given a single register which can store $O(\log n)$ bits (the choice of the constant is up to you). Moreover, we assume that "x cannot become too large", i.e., the x_i (and thus x) are of size polynomial in n and hence can be encoded using $O(\log n)$ bits.

a) Give a solution using a shared register supporting the fetch-and-add operation with a constant update and access complexity. If possible, prevent both lockouts and deadlocks.

b) Give a solution using a compare-and-swap register, also with constant access complexity. If successful, an update should need a constant number of steps (otherwise the process may retry). Are lockouts excluded?

c) Give a solution using a load-link/store-conditional register. Compare it to the preceding solutions.

d) Assume now that the return value of compare-and-swap is not whether the operation succeeded, but the value stored in the register after the operation. Can the problem still be solved? Proof your claim!

2 Space Efficient Binary Tree Algorithm*

The adaptive collect algorithm using binary trees from the lecture requires to store a complete binary tree of depth $n-1$, resulting in exponential memory requirements.

Suppose the algorithm is modified the following way: Whenever a process leaves a splitter with result **left** or **right** it flips a coin to replace this result by **left** or **right** with probability 1/2 each. Prove that for this randomized variant of the algorithm it is *with high probability*[1] sufficient to allocate memory polynomial in n![2]

[1] I.e., with probability at least $1 - 1/n^c$ for a choosable constant $c > 0$.

[2] Problems marked with an asterisk (*) are hard. Example solutions to these problems will not be provided. However, anybody who solves such a problem will receive a prize!