Lecturer: Nir Shavit

Handout 13

This exam is due on Wedensday December 13 at 10:10 AM, to be handed in to me during class hours. You may consult with other students about the material and ask me any relevant questions, but no collaboration or discussion of the actual questions among students is allowed. Good luck.

1. **A.** What is the consensus number of the following 2 process *Am-I-Consensus* problem.

   Am-I-Consensus each of P and Q is given a binary input. If both have input v, then both must decide v. If they have mixed inputs, then either they must agree, or Q may decide 0 and P may decide 1 (but not vice-versa).

   B. What is the consensus number of a concurrent *atomic stack* object, whose sequential specification is that it is a *stack*. In other words, it allows each of n processes to execute *push* and *pop* operations with the usual *LIFO* semantics.

2. A *one time T&S* object is an object on which n (for simplicity, assume n is even) processors can each perform a single *T&S* operation, returning a value in $\{0, 1\}$. Its sequential specification is that the first *T&S* operation returns 0 and all others return 1.

   A. A waitfree implemetation of an n-process *one time T&S* object using 2-process *T&S* objects is given in Figure 1. Each processor $P_i, i \in \{0..n-1\}$ runs the procedure ONE_TIME_TEST_AND_SET starting from the $\lfloor i/2 \rfloor$-th leaf node (i.e. two processors start from each leaf node). Why does this implemetation of a *one time T&S* object not work? *Prove* your claim.

   B. Using only atomic read/write registers, modify the above implementation of Figure 1 of a *one time T&S* object so that it does work. *Argue* its correctness.

   C. Suppose you are given a "bag" filled with 2-process *T&S* objects. It is known that exacly one of the *T&S* objects is faulty, that is, for any *T&S* request it returns an arbitrary value of 0 or 1. Can one implement (in a waitfree manner) out of a

bounded number of objects from the bag, a non-faulty *one time T&S* object for two processes? *Prove* your claim.

```
type node = record of
    win : test&set object;
    parent : pointer to node;
end record;
shared vars tree : a binary tree of width n/2 of node with win set to 0;

ONE_TIME_TEST_AND_SET(current_node:node) returns(value);
    x := T&S(current_node.win);
    if x = 1 or current_node.parent = null
      then return (x)
      else  return (ONE_TIME_TEST_AND_SET (current_node.parent))
      end if
    end ONE_TIME_TEST_AND_SET
```

Figure 1: Code of ONE_TIME_TEST_AND_SET

3. We construct the following shared counter object for n processors from two component structures. One is a *bitonic* counting network of width 8 (The balancers in the network are implemented using a small critical section based on a standard *T&T&S* spin-lock) and the other is a *waiting filter* of width n. Informally, the waiting filter is a kind of barrier, to be traversed after the counting network, where each processor waits for the processors with lower values. More precisely, a waiting filter is an n-element array of boolean values, called phase bits, where indexing starts from 0. Define the function $phase(v)$ to be $\lfloor (v/n) \rfloor \bmod 2$. We construct the new counter implementation by having processors first traverse the counting network and then access the waiting filter. When a process exits the counting network with value v, it awaits by going to location $(v-1) \bmod n$ in the filter, and reading and testing till that location is set to $phase(v-1)$. When this event occurs, it notifies by setting location v to $phase(v)$, and then it returns its value.

   A. What clear advantage does the new shared counter have over the standard counting network based shared counter? (a one or two line answer please)

B. Assume an experiment where 10^6 *INC* operations are executed by 16 processors on a shared-bus multiprocessor (each performs 1/16 of them). Would you expect the new shared counter to have a higher or lower throughput (recall, higher throughput means it takes less time to complete all the operations) than the standard counting network based implemetation? Would you expect it to have a higher or lower throughput than the standard single *T&T&S* spin-lock based implementation? Explain your reasoning in accurate detail (10 lines at most, sharp and to the point).

C. Assume an experiment where 10^6 *INC* operations are executed by 60 processors (each performs 1/60 of them) on an interconnection network based multiprocessor like the Alewife machine or the Butterfly . Would you expect it to have a higher or lower throughput than the standard counting network based implemetation? Would you expect it to have a higher or lower throughput than the standard *T&T&S* spin-lock based implementation? Explain your reasoning in detail.