NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
EXAMINATION FOR

SEMESTER 2 2012/2013

CS 4231 - PARALLEL AND DISTRIBUTED ALGORITHMS

May 2013 **Time Allowed: 2 Hours**

INSTRUCTIONS TO CANDIDATES

This is a closed book exam. You are allowed ONE HANDWRITTEN DOUBLE-SIDED A4 "cheat sheet".

1. Enter your matriculation number here:

2. Enter the answers to the questions in this answer book in the provided spaces.
3. This examination booklet has 13 pages, including this cover page, and contains 6 questions (questions 1, 2, 4, and 6 have 4, 2, 2, and 2 subparts, respectively). The last 2 pages are blank, for use as scratch paper if needed. If you want me to grade something on one of those pages please explicitly reference that page in the main pages.
4. Answer all questions.
5. The maximal marks achievable is indicated for each question.

Do not write below this line

Q#	1	2	3	4	5	6	Σ
Max	20	15	20	15	15	15	100
Sc							

Q1. (20 marks) **Some useful basic concepts.**

(a) (5 marks) Explain the difference between efficiency and cost-optimality. Can an algorithm be cost-optimal with efficiency no more than 50%? What about the opposite: can an algorithm have very high efficiency but not be cost optimal?

(b) (5 marks) Explain the difference between static and dynamic mapping, and how one decides which to use for a particular algorithm.

(c) (5 marks) Explain the difference between exploratory and speculative decomposition. Examples of algorithms that use one, or both, would be useful in (as part of) your explanation. If you give an example that uses both, be clear as to how each type helps.

(d) (5 marks) Explain what pipelining is and how it can help produce speedup. Again, giving an example algorithm would be useful in (as part of) your explanation.

Q2. (15 marks) **Basic algorithm classes.**

(a) (10 marks) **Sorting algorithms.**

Suppose we have an array A of n integers whose values are uniformly distributed over the interval $[0, 100n)$. Given p processors we can sort A efficiently as follows. Mentally divide the interval into p equal buckets of size $\frac{100n}{p}$; for example, bucket 0 (b_0) contains the items from $[0, \frac{100n}{p})$, bucket 1 (b_1) contains the items from $[\frac{100n}{p}, \frac{200n}{p})$, etc. Note that given an element a_k of A, you can determine which bucket a_k belongs to in $O(1)$ time by $\lfloor \frac{p \cdot a_k}{100n} \rfloor$, where $\lfloor \cdot \rfloor$ is the "floor" (a.k.a. round down) operation.

To sort A, we work in stages separated by barriers. Each p_i executes as follows:

Stage 1 Decompose A into p equal blocks of size $\frac{n}{p}$. Each process p_i gets its own set of buckets b^i_*, and sorts its $\frac{n}{p}$ items into its buckets.

Stage 2 Each process p_i takes control of all of the buckets b^*_i—that is, process p_0 takes control of b^0_0, b^1_0, etc: all of the elements which any process found to be in bucket 0. Each process then sequentially sorts the items in their buckets. Just as in the sophisticated parallel quicksort's pivot operation, each process then publishes the number of items it controls.

Stage 3 Just as with sophisticated parallel quicksort's pivot operation, the processes compute the prefix-sum of the number of items each of them control.

Stage 4 Just as with sophisticated parallel quicksort's pivot operation, the processes copy their items into the destination array using the destination indexes learned via prefix-sum.

Your task. What is the big-O runtime of this algorithm? Please show your work.

(More room for part (a), if needed.)

(b) (5 marks) **Graph algorithms.**

Recall the single-source shortest path problem. In class we showed how to parallelize Dijkstra's algorithm using map and reduce such that, given a graph of size n, and n processors, we can compute the single-source shortest path in time $O(n \cdot \log n)$.

Suppose we had n^2 processors available. How could we compute the single-source shortest path in time $O(n)$?

Q3. (20 marks) **Algorithm design.**

Suppose we have an array `A` of integers with length n. We also have a function `f` whose specification is

```
Boolean f(int x);
```

The operation "filter" takes our array `A` and predicate function `f` and returns a new array `B` whose elements are exactly those elements of `A` that are accepted by `f` (*i.e.*, an element `A[i]` will be in `B` if and only if `f(A[i]) == true`. For example, starting from the array

```
[1,2,3,4,5,6,7,8,9,10]
```

and filtering with the function which returns `true` when applied to prime numbers, and `false` otherwise, then we will get

```
[2,3,5,7].
```

Your task: assuming that `f` runs in $O(1)$ time, please explain how to implement filter (given n processors) in $O(\log n)$ time. By "explain" I mean give a precise high-level description that is sufficient to prove to me that you understand why your algorithm will take $O(\log n)$ time. You do **not** have to write actual code.

Q4. (15 marks) **Locks.**

Recall the MCS algorithm covered in class (and again as part on the homework). The code is reproduced below for your reference:

```
class MCSLock {
  class QNode {
    bool locked = false;
    QNode next = null;
  }

  QNode tail;

  ThreadLocal<QNode> myNode; // Each thread has its own myNode
  MCSLock() {
    tail = null;
    myNode = new ThreadLocal<QNode>(new QNode);
  }

  lock() {
    QNode qnode = myNode.get(); // Get "our thread's" qnode
    QNode pred = GNS(tail,qnode);
    if (pred != null) {
      qnode.locked = true;
      pred.next = qnode;
      while(qnode.locked) {}
    }
  }

  unlock() {
    QNode qnode = myNode.get(); // Get "our thread's" qnode
    if (qnode.next == null) {
      if (CAS(tail,qnode,null))
        return;
      while(qnode.next == null) {}
    }
    qnode.next.locked = false;
    qnode.next = null;
  }
}
```

(a) (10 marks) This algorithm relies on an atomic get-and-set (GNS) operation in `lock()`. Sadly, many hardware architectures do not provide such an atomic operation. Please implement GNS in an "atomic-like manner" with a compare & swap (CAS):

(b) (5 marks) Recall the definitions of "deadlock-free" and "starvation-free" as characterizations of lock implementations. Assuming a genuinely atomic GNS operation, is the MCS lock deadlock-free? How about starvation-free? Does its status change when GNS is implemented with CAS? Explain your answers.

Q5. (15 marks) **Lock-free data structures.**

Recall Maged Michael's non-blocking stack from lecture 9 and the homework, the code for which is reproduced below:

```
bool push(node n) {
  node t;

  for(j=0;j<threads; j++) {
    if (H[j] == n)
      return false;
  }
  while(true) {
    t = top;
    n.tail = t;
    if (CAS(top, t, n))
      break;
  }
  return true;
}

node pop() {
  node t, n;

  while(true) {
    t = top;
    H[tid] = t;
    // What can happen if the next two lines are omitted?
    if (top != t)
      continue;
    // What can happen if the previous two lines are omitted?
    if (t == null)
      break;
    n = t.tail;
    if (CAS(top,t,n))
      break;
  }
  if (t != null)
    t.tail = null;
  H[tid] = null;
  return t;
}
```

- Please explain what can go wrong if the two lines between the comments in the code are omitted. The more detail you can provide, the better.

Q6. (15 marks) **Distributed programming.**

- (10 marks) Recall that map & reduce are particularly good fits for distributed programming because many of the ugly realities of distributed systems (load balancing, failure detection & recovery, etc.) can be hidden from the programmer by a library that implements map and reduce in a generic way (*i.e.*, parameterized over the underlying function or operation).

  Another basic operation we covered was prefix-sum (also known as "scan"). Is this operation also a good fit for distributed programming? Why or why not?

- (5 marks) Explain why speculative decomposition is particularly useful in the context of distributed algorithms. A short answer is fine, as long as you are clear.

(Extra page 1. I will not grade this page unless one of the main pages explicitly references it.)

(Extra page 2. I will not grade this page unless one of the main pages explicitly references it.)

END OF EXAMINATION