CSC/ECE 506: Architecture of Parallel Computers
Sample Test 1 with Answers

This was a 120-minute open-book test. You were allowed to use the textbook and any course notes that you might have. You could also use your laptop, but you were not allowed to communicate with any other student, by electronic or other means, during the test.

You were to answer five of the six questions. Each question was worth 20 points. If you answered all six questions, your five highest scores counted.

Question 1. Of the three kinds of programming models we have studied,

- Data parallel (write "D")
- Message-passing ("M")
- Shared address space ("S")

which are the following statements *most true* of, and which are they *least true* of? In some cases, there may be a "tie" between two of the systems; in that case, you may give either system as an answer.

	Most true of	Least true of
(a) Has high synchronization overhead.	M, S	D
(b) Can be used on vector processors.	D	M, S
(c) Each processor has its own memory.	M	S
(d) Susceptible to coherence problems (copies can get inconsistent).	S, M	D, M
(e) All processors execute the same program.	D	M, S
(f) Uses synchronization primitives to control access to shared data.	S	D, M
(g) All data replication is performed by explicit software operations.	D, M	S
(h) Tends to require copies to be made of subarrays.	M	S, D
(i) All processes can name (or "address") the same data items.	S	M
(j) Processors expect to access any memory location fairly quickly.	D, S	M

Question 2. Suppose a problem consists of 25% serial work and 75% parallelizable work.

(a) *(3 points)* If the problem takes 400 seconds to run on a uniprocessor system, how long will it take on a system with 4 processors?

Answer: Total execution time = time to complete serial part + time to complete parallel part

Time to complete serial part = $400 \times 0.25 = 100$ sec.

Time to complete parallel part = $\dfrac{400 \times 0.75}{4} = 75$ sec.

Total execution time = 100 + 75 = 175 sec.

(b) *(4 points)* What would be the efficiency on this system?

Answer: Efficiency = $\frac{\text{speedup}}{\text{\# of processors}} = \frac{\frac{400}{175}}{4}$ = 400/700 = 4/7 ≈ 57%.

(c) *(3 points)* How long does this problem take on a system with *p* processors? Give a formula based on *p*.

$(400 \times 0.25) + \frac{400 \infty 0.75}{p} = 100 + \frac{300}{p}$.

(d) *(4 points)* What is the efficiency on this system?

Answer: $\frac{400}{100 + \frac{300}{p}} \div p = \frac{400}{100 + \frac{300}{p}} \times \frac{1}{p} = \frac{4}{p+3}$.

(e) *(3 points)* What is the fastest this program can possibly run? That is, what is the execution time as $p \rightarrow \infty$?

Answer: By Amdahl's law the execution time cannot be faster than the serial part, so the fastest possible execution time is 100 sec.

(f) *(3 points)* Suppose our program has a real-time deadline of 120 sec. How many processors would we have to use to meet this deadline? Or can it not be met at all?

Answer: We need $100 + \frac{300}{p}$ = 120, or 100*p* + 300 = 120*p*. From this we can see that 300 = 20*p*. So we would need at least 15 processors.

Question 3. *(5 points each)* Given this loop,

```
for (i = 1; i < n; i++) {
      S1: a[i] = a[i] * c[i-1];
      S2: b[i] = b[i] * 2.7181828;
      S3: c[i] = a[i] * a[i+1];
      S4: d[i] = d[i] * b[i];
}
```

(a) List all of the dependences, and indicate which are loop-carried (LC) and which are loop independent (LI).

Answer:

`S1[i]`	→T	`S3[i]`	Loop-independent
`S1[i]`	→A	`S3[i-1]`	Loop-carried
`S1[i]`	→T	`S3[i+1]`	Loop-carried
`S3[i]`	→A	`S1[i+1]`	Loop-carried
`S2[i]`	→T	`S4[i]`	Loop-independent

(b) Break up, or distribute the statements in the body of the loop, to expose one opportunity for parallelism.

Answer:

```
for (i = 1; i < n; i++) {
      S1: a[i] = a[i] * c[i-1];
      S3: c[i] = a[i] * a[i-1];
}
for (i = 1; i < n; i++) {
```

```
      S2: b[i] = b[i] * 2.7181828;
      S4: d[i] = d[i] * b[i];
}
```

(c) Reorganize the loop to expose another opportunity for parallelism.

Answer: Break up the second loop above as follows:

```
.
for (i = 1; i < n; i++) {
      S2: b[i] = b[i] * 2.7181828;
      post(i);
}
for (i = 1; i < n; i++) {
      wait(i);
      S4: d[i] = d[i] * b[i];
}
```

(d) Suppose that each arithmetic operation takes time 1, each post takes time 1, and each wait takes time 1. A wait and a matching post can execute simultaneously. Loop control (incrementing the index variable, branching back to the top) takes negligible time. If an adequate number of processors are available, how long does it take to execute the loop from part (c), in terms of *n*?

Answer: The first loop gets started at time 0. It finishes its first multiplication at time 1, and its first post at time 2. The second loop completes its first wait at time 2, its first multiplication at time 3, its second wait at time 4, and its second multiplication at time 5. So it takes 2*n* + 1 time units to finish the loop, which is actually slower than the 2*n* time units it would've taken on a single processor. This illustrates the fact that it is not helpful to parallelize small (two-statement) loops.

Question 4. *[Average score 15.4]* (a) *(18 points)* Consider an algorithm to calculate the all-pair shortest path. If we are to parallelize this algorithm for each **for** loop, fill in the table appropriately for each variable used.

```
for (k = 0; k < n; k++){
      for (i = 0; i < n; i++){
            for (j = 0; j < n; j++){
                  if (a[i][k] + a[k][j] < a[i][j])    {
                        a[i][j] = a[i][k] + a[k][j];
                        p[i][j] = k;
                  }
            }
      }
}
```

Which loop parallelized? →	**for** *k*	**for** *i*	**for** *j*
Read-only	*n*	*k*, *n*	*i*, *k*, *n*
RW non-conflicting		*a*, *p*	*a*, *p*
RW conflicting	*a*, *i* ,*j*, *k*, *p*	*i*, *j*	*j*
Private	*i*, *j*, *k*	*i*, *j*	*j*
Shared	*a*, *n*, *p*	*a*, *k*, *n*, *p*	*a*, *i*, *k*, *n*, *p*

(b) *(2 points)* Do any of the shared variables need to be protected by a critical section? Explain.

Answer: In the **for** *k* parallelization, *a* and *p* need to be protected by a critical section, because multiple processes can update the same elements of this array.**Question 4.** *(5 points each)* We have seen several places where implicit or explicit synchronization is used in parallel programs. Sometimes synchronization statements are inserted to assure correct operation of the program; other times, they are used for performance reasons.

For each of the following cases, tell whether synchronization—

- insures correct behavior of the program (write "C"),
- is not required, but improves the performance of the program (write "↑"),
- is not required, and has an unknown effect on the performance of the program (write "?"), or
- is not required, and causes the program to run more slowly (write "↓").

For full credit, explain your answers! You need not write more than one sentence of explanation, though.

(a) In the Ocean simulation, letting the loops run in parallel, and inserting synchronization operations to assure that neighboring points have been updated before they are used in a calculation.

Answer: ↓. The algorithm will converge without such synchronization, and the number of synchronization operations would be so great as to have a serious effect on its performance.

(b) In the Ocean simulation, using some updated values (e.g., the value above and the value on the left) in calculating new values for points.

Answer: ↑. If no updated values are used, we have Jacobi iteration, which also converges, but more slowly.

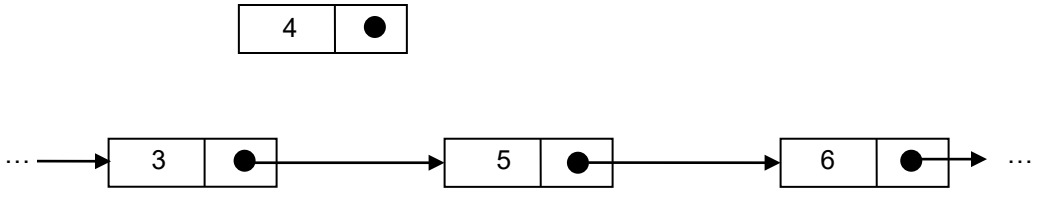
(c) In the SAS version of the Ocean simulation, forcing all processors to wait for the last one to finish before testing for convergence.

Answer: ?. If a processor were allowed to continue, it might execute an extra iteration. But it would eventually finish. What is not clear is whether the extra iterations would take more time than the synchronization that was removed.

(d) Doing a parallel sum reduction in Steele's data-parallel algorithm for matrix multiplication. *Note:* The question asks whether parallel sum reductions require synchronization (not whether they're faster than serial sum reductions).

Answer: C. Otherwise, updates to the global sum could be lost.

Question 6. A linked list is shown below. Suppose one thread is trying to insert 4, while another thread is trying to delete 5.



(a) *(6 points)* How many read- and write-locks will be **acquired** if the parallelization strategy is—

	Number of read-locks	Number of write-locks
(i) parallelization among readers	0	2
(ii) global lock approach	0	2
(iii) fine-grain lock approach	2	2

(In (iii), the write-locks are for the nodes whose `next` pointers are going to be modified).

(b) *(9 points)* Suppose that it takes 1 time unit to "visit" each node. Each time a node is reached in a list traversal, it counts as a "visit."

For example, in order to delete node 6, a thread would visit three nodes: node 3, node 5, and node 6. It would not have to visit the node *after* node 6, because it would just change node 5's `next` pointer to point to that node, without ever actually going to the node itself. It *would* have to visit node 6 to access its `next` pointer and copy it to node 5.

Traversals can be performed in parallel if locks do not prevent it. Thus, the insertion operation can visit its first node at the same time that the deletion operation visits its first node. Thus, if five nodes are traversed in parallel, only five time units are needed.

How many time units are needed to perform the insertion + deletion with each of the parallelization strategies? You may assume it's not necessary to "re-validate assumptions" by re-traversing the list. (Please show your work!)

(i) Parallelization among readers

Answer: Five. (1) Insertion visits node 3, (2) insertion visits node 4, (3) deletion visits node 3, (4) deletion visits node 4, (5) deletion visits node 5. (This assumes that the insertion occurs before the deletion. If the deletion happens first, the deletion does not have to visit node 4, so only four units would be needed.)

(ii) Global lock approach

Answer: Four. (1) Insertion and deletion both visit node 3, (2) insertion visits node 4, (3) deletion visits node 4, (4) deletion visits node 5. (If deletion occurs first, then insertion needs to restart the traversal and revisit node 3, because node 3 has been changed by deletion.)

(iii) Fine-grain lock approach

Four. (1) Insertion and deletion both visit node 3, (2) insertion visits node 4, (3) deletion visits node 4, (4) deletion visits node 5. (If deletion occurs first, insertion needs to re-start the traversal, as above.)

(c) *(1 point)* Suppose that it takes 1 time unit to acquire a lock, and one time unit to release it. If the *cost* of an operation is measured by

the number of nodes it visits + the number of locks it acquires

then which strategy is the cheapest for this sequence (insert 4, delete 5)?

Answer: The global lock approach. It costs 4 + 2 = 6. Parallelization among readers costs 5 + 2 = 7. The fine-grain lock approach costs 4 + 4 = 8.

(d) *(4 points)* This strategy (your answer to (c)) would not *always* be the cheapest, however. Explain why the other strategies might be cheaper for other combinations of list operations.

Answer: If write operations are rare, relative to other processing, then parallelization among readers is best. It has few lock operations, and few operations are likely to be delayed.

If there are many concurrent write operations on lists, then fine-grain locks are best. They don't require locking out all other operations on a list while an operation is in progress.