2202-COL380 Minor 2

Chinmay Mittal

TOTAL POINTS

35 / 60

QUESTION 1

Fill in the blanks 12 pts

1.1 **2 / 2**

✓ **+ 1 pts** *a) entity*

+ 0.5 pts a) only process/thread/task mentioned

✓ **+ 0.5 pts** *b) that entity already holds that lock*

✓ **+ 0.5 pts** *Explanation: By definition, a reentrant lock requires a well-identified entity which already holds the lock to reacquire it*

+ 0 pts Incorrect/Unattempted

+ 0.25 pts Within limit

1.2 **0 / 2**

+ 1 pts a known number of times

+ 1 pts Explanation: A re-entrant lock is by definition allowed to be reacquired >1 times. But it still can't be reacquired an unlimited number of times because every lock requires some resources, which are always limited.

✓ **+ 0 pts** *Incorrect/Unattempted*

1.3 **2 / 2**

✓ **+ 2 pts** *Correct*

+ 1 pts Partially correct

+ 0 pts Incorrect / unattempted

1.4 **2 / 2**

✓ **+ 2 pts** *Correct*

+ 1 pts partially correct

+ 0 pts Incorrect / unattempted

1.5 **0 / 2**

+ 1 pts strongly

+ 1 pts Explanation: By definition, weakly fair schedulers guarantee that every thread is given a chance to complete its protocol atleast once, and strongly fair schedulers guarantee the chance as often as the thread seeks.

✓ **+ 0 pts** *Incorrect/Unattempted*

1.6 **2 / 2**

✓ **+ 2 pts** *Correct with explanation*

+ 1 pts somewhat correct with explanation

+ 0 pts Incorrect / unattempted

QUESTION 2

Define and Explain 10 pts

2.1 **2 / 2**

✓ **+ 2 pts** *Correct Mathematical definition and correct Explanation*

+ 1.5 pts Correct Mathematical definition but lacks explanation

+ 1 pts Explained in words, no mathematical definition

+ 0 pts otherwise

2.2 **2 / 2**

✓ **+ 2 pts** *Correct Mathematical definition and correct Explanation*

+ 1.5 pts Correct Mathematical definition but lacks explanation

+ 1 pts Explained in words, no mathematical definition

+ 0 pts otherwise

2.3 **2 / 2**

✓ **+ 2 pts** *Correct Mathematical definition and correct Explanation*

+ 1.5 pts Correct Mathematical definition but lacks explanation

+ 1 pts Explained in words, no mathematical definition

+ 0 pts otherwise

2.4 **2 / 2**

✓ **+ 2 pts** *Correct*

The cost $$C$$ of a parallel program is defined as the product of its time and the maximum number of processors available for use at any step: $$C = t(n, p) * p$$

+ 0 pts Incorrect or not attempted.

2.5 **2 / 2**

✓ **+ 2 pts** *Correct*

Using multiple threads/processes involves the overhead of synchronising them, exchanging data, and waiting for the results from a different thread/process to proceed. The combined effect of all these overheads is termed parallelization overhead. Overhead is defined as $$p * t(n, p) - t(n, 1)$$

+ 0 pts Incorrect or not attempted

QUESTION 3

3 **2.5 / 4**

✓ **+ 2.5 pts** *Identifying that in iteration $$i$$, $$n / 2^i$$ processors are active.*

+ 1.5 pts Concluding that parallel work equals: $$O(n / 2) + O(n / 4) + ... + O(1) = O(n)$$

+ 0 pts Incorrect or not attempted.

QUESTION 4

4 **5 / 8**

+ 0 pts No/Inadequate answer

✓ **+ 2 pts** *Basic Algorithm*

✓ **+ 1 pts** *Fully distributed*

- 1 pts Error in algorithm/correctness

+ 2 pts Show safety

✓ **+ 2 pts** *Explain Lock-freeness*

+ 1 pts Answer and explain Wait-freeness

- 1 pts Error in Lock/Wait-free explanation

1. Even if there is no one else with a new medallion?
2. not enough
3. Not quite
4. Do you stop playing waiting for the control?

QUESTION 5

5 **4 / 6**

+ 0 pts No/Inadequate answer

✓ **+ 2 pts** *scatterV (single)*

+ 1 pts Destination buffer type/count

✓ **+ 1.5 pts** *Send buffer type/count*

+ 1.5 pts Send buffer addressing (displ)

- 0.5 pts Minor Error/Inefficiency in type/addressing

- 1 pts Error in type/addressing

- 1.5 pts Many/Major error in type/addressing

Multiple sends/Row-wise scatter

+ 2.5 pts Scatter per row/block

- 1 pts Incorrect addressing/Type

+ 0.5 *Point adjustment*

QUESTION 6

6 **1.5 / 6**

+ 1 pts proper understanding of eager messages.

+ 0.5 pts Partially correct matching

+ 1 pts proper explanation of how matching take place

+ 1 pts considering wild card matching

+ 1 pts to account for thread safety and order

✓ **+ 1 pts** *correct delivery when MPI receive call comes before message*

✓ **+ 1 pts** *correct delivery when message comes before MPI receive call*

+ 0 pts No/Incorrect answer

- 0.5 *Point adjustment*

5 In eager messages Envelope, content are sent together in single packet only.

QUESTION 7

7 **3 / 6**

+ 6 pts Fully Correct

✓ **+ 1 pts** *Understanding of MPI Gather*

+ 2 pts Used sequential loop and correct communication time = $$ p \times (s + n / p) $$ **OR** $$(p - 1) \times (s + n / p) $$

✓ **+ 2 pts** *Used log tree*

+ 3 pts Correct communication time using log tree $$= \left(s + \frac{n}{p}\right) + \left(s + \frac{2n}{p}\right) + \left(s + \frac{4n}{p}\right) + ... + \left(s + \frac{pn}{2p}\right)$$

$$ = s(log_{2}{p}) + \left(\frac{n}{p}\right)\left(1 + 2 + 4 + ... + \frac{p}{2}\right) $$

$$ = s(log_{2}{p}) + \left(\frac{n}{p} \right) (p - 1)$$

+ 0 pts Incorrect / Not Attempted

6 from where you have introduced logp ?

7 p / 2 (not 1 / 2)

8 Can you explain this term in regrade request ?

QUESTION 8

8 **1 / 8**

+ 0 pts No/InadequateAnswer

+ 2 pts Recognize need for locks in the same order

+ 2 pts Correct progression of locks

+ 1 pts Progress of extractMin

+ 1 pts Progress of add

Coarse grained locking

+ 1 pts Single lock

+ 1 pts Discuss safety and progress

+ 1 pts CAS usage (recognize need for key uniqueness?)

+ 1 *Point adjustment*

9 How?

Entry No: 2020CS10336 **Name:** CHINMAY MITTAL

Fill these first on all sheets

COL 380 MINOR EXAM

SEMESTER II 2022-2023

1 hour, 60 marks

Maximum marks are listed in [] for each question. Justify your answers: most marks are designated for correct justification. Use the provided space. Follow Course's Academic Integrity Code.

1. [12] Fill in the blanks. Where /-separated options are provided, strike out *all* the wrong ones.

a) A reentrant lock can be reacquired by a/an _____ only if _____.

Explain: A reentrant lock can be reaquired by ~~a~~ the same entity if it has already been acquired by that entity before. Or it has not been acquired by any other entity.

b) A reentrant lock can be reacquired by an entity at-most 0/1/unlimited/a known number of times.

Explain: unlimited → the same entity can keep reacquiring the lock after acquiring it once.

c) If some entity X can complete its operations starting in any (global) state of a distributed protocol, the protocols is ~~safe/fair/starvation-free~~/lock-free/~~wait-free/none of these~~.

Explain: lock free. (Independent of the scheduler one thread can complete → other threads might not be able to complete. and hence) it is neither fair/starvation free/wait free

d) If every entity can complete its operation starting in any (global) state of a distributed protocol, even if it runs in isolation, the protocol is safe/fair/starvation-free/lock-free/wait-free/~~none of these~~.

can start from any global state.

Explain: Algorithm becomes fair, lock free, safe, wait free independent of the scheduler starting from any state (all entities can progress)

e) A scheduler is weakly/strongly fair if guarantees that every ready synchronizer is eventually scheduled, no matter how often it becomes ready.

Explain: weakly fair.

for being strongly fair if a thread becomes ready infinitely often then it must be scheduled infinitely often.

f) Bakery algorithm is ~~wait-free~~/~~lock-free~~/deadlock-free/fair/starvation-free

Explain: An unfair scheduler might block all threads by putting the thread with the minimum token to sleep while it is executing its critical section ⇒ No one progresses independent of scheduler ⇒ not wait free | lock free.

With a fair scheduler, the thread with the smallest token will always complete, then the thread with the next smallest token and so on ⇒ everybody progresses ⇒ starvation free and deadlock free ⇒ every thread eventually enters the critical section ⇒ fair

2. [10] Define (quantitatively) and explain the following terms:

a) Parallelization efficiency

Speedup $S_p = \frac{t_1}{t_p}$ ⟶ how much faster does the code become with p processors.

efficiency $\varepsilon_p = \frac{S_p}{p}$ ⟶ it is expected that $S_p = p$ in the best case but due to overhead it is lower

↳ ε_p quantifies this efficiency.

b) Iso-efficiency

It is how the problem size ($t_1(n)$) must scale with p as we increase the number of processors to maintain a constant efficiency

$$I(p) = \Omega\left(\bar{o}(n,p)\right)$$

↳ overhead

& $\varepsilon = \frac{t_1(n)}{p\, t_1(n,p)}$

let $n = f(p)$

$\Rightarrow I(p) = t_1(f(p))$

c) Weak scalability

If I have to increase the problem size as the number of processors grows to maintain the efficiency then the algorithm is said to be weakly scalable.

d) Parallelization cost

parallelization cost = $t_p * p$

t_p: Number of steps for which I employ them.

p: Number of processors I employ.

Entry No: 2020CS10336 **Name:** CHINMAY MITTAL

e) Parallelization overhead

parallel algorithms often have synchronization / communication overheads leading to the efficiency to be often less than one

$O(n,p) = pt(n,p) - t(n,1)$ → overhead quantifies this cost.

3. [4] Consider the following approach to reduction. What is the parallel work, given *n* data items. You may use big-O notation finally.

```
for step = ceil(log n)-1 down to 0
    in parallel for each processor i, 0 <= i < 2^step
        if(i+2^step < n)
            data[i] += data[i+2^step]
return data[0];
```

In the first step $\frac{N}{2}$ additions are done in one step → 1 unit of parallel work

In the next step $\frac{N}{4}$ additions are done in one step → 1 unit of parallel work

In the i^{th} step $\frac{N}{2^i}$ additions are done in parallel.

The total number of steps is $\lceil \log_2 N \rceil$

Hence the total parallel work is $O(\log N)$

4. [8] We have a multi-player game where each player gets to control certain aspects of the game-play if they earn a bonus medallion during their play. A player may earn a medallion at any time but only one player is allowed be a controller at one time. Hence, they must wrest control every time they earn a bonus. This is done through message-passing. Every bonus winner must get control (for whatever little time). (Assume messages are reliable and delivered in order between two players.) Provide a lock-free algorithm to allow the control to be safely taken. Explain why it is lock-free. Explain whether it is also wait-free?

We can implement a distributed mutex algorithm to simulate the following. Each player maintains a local timestamp (according to Lamport's time stamp algo. Whenever a player earns a bonus medallion, it creates a request to get the controller with its own local timestamp and broadcasts it to every other players. and adds this request to its own queue. The requests in the queue are ordered (smallest first) by the timestamps of the request.

A players will get the controllers. if its own request is at the top of its queue (smallest timestamp) and it has received messages (acknowledgment's) from all all the other entities.

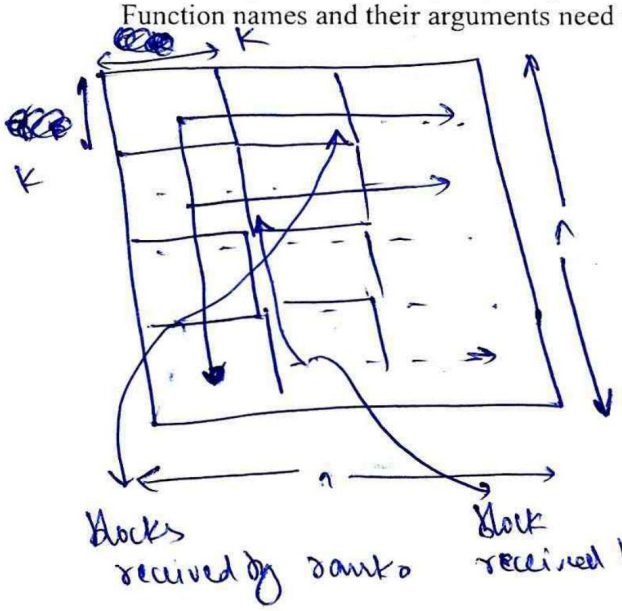
Once I get into the critical section. I will exit after some time broadcast to everybody that I have exited. and everybody will remove my request from the queue. (Everybody will remove the request at the top anyways after a given time)

This ensures that somebody always progresses (independent of the scheduler) because either a thread completes its critical section or eventually everybody will remove its request from the top of the queue and proceed others. This way eventually all threads which are not sleeping will make progress and our alg. is also wait free.

Fill ID on all sheets before you beg

Entry No: 2020CS10336 **Name:** CHINMAY MITTAL

5. [6] Provide MPI-like code for distributing an nxn int matrix block-wise from rank 0 to all ranks $< p^2$ (Assume $n = kp$ for some integer k). Explain the data type(s) and the collective function(s) you would use. Function names and their arguments need not be precise, but the functionality you are using should be clear.



We will create a datatype for a block of size K * K Blocks.

using MPI_Type_vector;

MPI Data_type Block_type;

MPI_type_vector(K, K, n, MPI_INT, &Block_type);

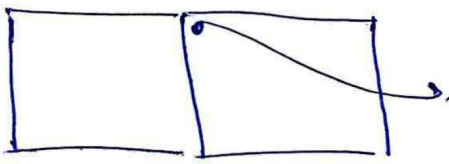
K rows of Blocks in each block row

K Stride

datatype for a Block.

We will use resize to allow the next Block to start just after the preceeding Block.

MPI_type_resize(0, K * sizeof(int), Blocktype, &NewBlocktype);



0 → beginning

this ensures that the next Block starts here.

We can use the scatterv function to distribute the Blocks to all the ranks. Each process receives a row of Blocks. We will need to compute the displacements to find the starting address of each Block.

We can now use the scatterv function to distribute the entire matrix.

Scatterv(A, 1, NewBlocktype, disp, 0, MPI_COMM, recv buffer, p, root,

A → 2D array

1 → Blocks received by a process

New Blocktype, disp → displacements

0 → root

6. [6] Explain the steps an MPI stub must take to match and deliver eager messages. (Note that multiple multi-threaded MPI ranks could be be using the same MPI stub.)

Whenever a receive call is made the MPI stub needs to take note of it to match with any incoming send calls later.

As soon as a send call is posted, the stub needs to match it with the earliest matching receive (of which it had earlier kept note)

If no such receive is found and a local buffer is available then the stub can use the local buffer for the incoming data.

If a matching receive is found then use the buffer provided by that receive to store the data.

otherwise ignore all the packets sent by the eager send and throw an error.

At the send side an eager packet is sent first for matching details and and then the data is started to be broadcasted without waiting for an acknowledgment receipt.

The receivers will acknowledge at the end whether it could successfully store the data or not.

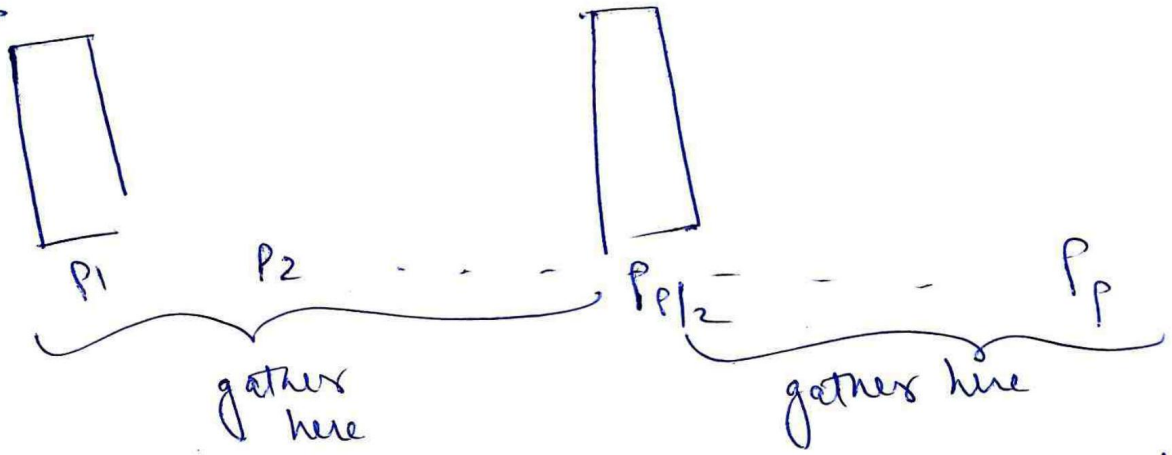
Entry No: 2020CS10336 **Name:** CHINMAY MITTAL.

7. [6] List the steps and the communication time required to implement MPI Gather of n items in a group of p processors. Assume a fully non-blocking network and that each transfer of data of size k from one source to one destination takes s+k time, s being a constant.

Assume every processor has $\lfloor \frac{n}{p} \rfloor$ or $\lceil \frac{n}{p} \rceil$ data items.

Since there is an overhead (s) for every communication.

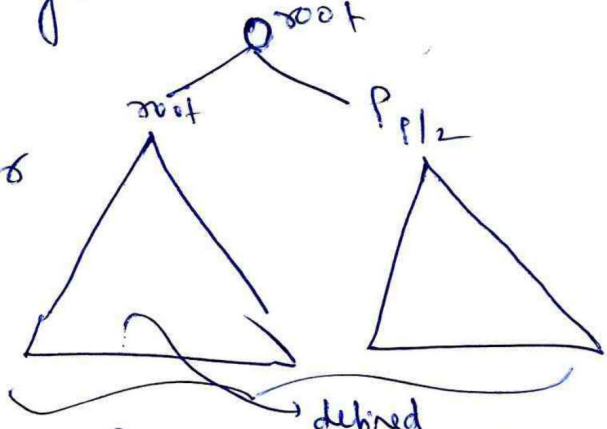
we would want to minimize the total number of communication pairs



We can thus make a tree kind of structure where multiple gathers are going on simultaneously.

Since the network is non blocking many pairs can communicate together simultaneously. It also saves on the communication overhead

Starting from the leaf layer parents accumulate the data from the children and this keeps happening till the top.



P processors recursively

Messages exchanged ⟶ $\frac{p}{2} + \frac{p}{4} + \frac{p}{8} \ldots\ldots 1$

Size of each message ⟶ $\lceil \frac{n}{p} \rceil \quad 2\lceil \frac{n}{p} \rceil \ldots\ldots 2^{\frac{p}{2}} \lceil \frac{n}{p} \rceil$

→ total time = (messages exchanged) * s + $\log p * \frac{p}{2} \lceil \frac{n}{p} \rceil$

8. [8] Implement a multi-threaded concurrent shared-memory minHeap using an array-based tree. Use Compare and Swap. (You may use locks, but 1 mark is reserved for using CAS.) Fine-grained locking is expected. Analyze its progress and safety properties. (Operations to support are *extractMin*() and *add*(*key*).)

extractMin()

return arr[2]

no need to lock

Since we maintain the top most node as the minimum till the point the call was made.

While adding we lock the last position in the array and only then add and heapify