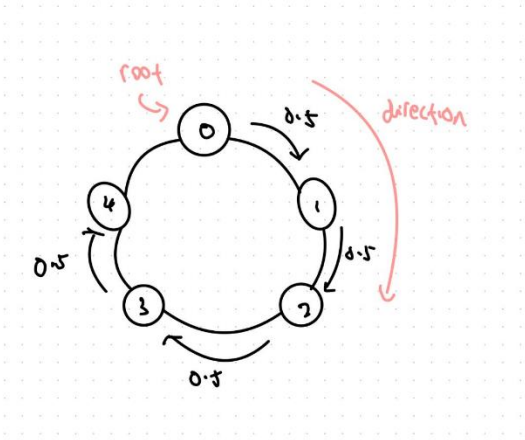
MONASH University

FIT3143 2024 - Applied Session Week 8

Student ID	Student Name	Student Email
Teh Jia Xuan	32844700	Jteh0015@student.monash.edu

Question 1 – Part A

Rank 0 will send the first message to rank1 and rank 1 forward the message to rank 2 and so on until rank 4. Each message will take 0.5 seconds to transfer to connected machine. Since message can only travel in one direction. The order will be the diagram below. So total time taken will be 2 seconds. It starts by sending rank 0 to rank 1 then it sends to rank 2, rank 3 and rank 4. Each machine must wait for the message from previous one before it can receive its own message.



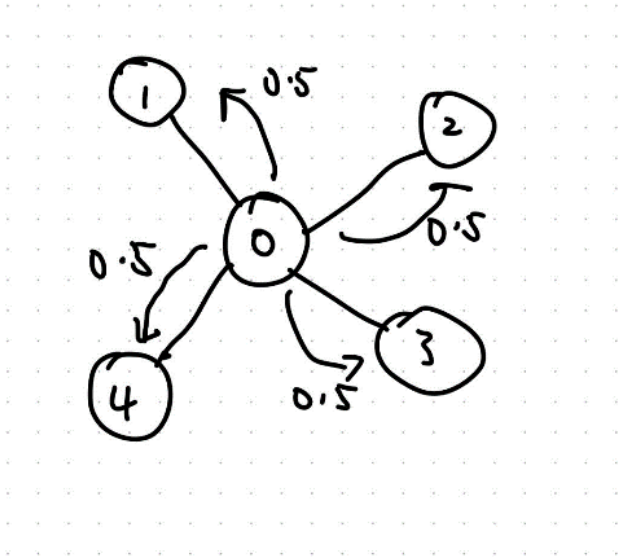
Time phase	Node 1	Node 2	Node 3	Node 4
1	0 -> 1			
2		1 -> 2		
3			2 -> 3	
4				3 -> 4

The diagram and table above is for further illustration assume that the message can only be sent in clockwise direction, it shows that 0 is the root node it will send message to node 1. Then node 1 will forward to node 2, node 2 will forward to node 3 and node 3 will forward to node 4. Each phase took 0.5 seconds. Thus, 4x0.5 seconds = 2 seconds.

Question 1 – Part B

By using star topology, the organisation of nodes is drawn as the diagram below. 0 is still the root node and it is directly connected to other nodes, the total time will still be 2 seconds if two messages cannot be sent at the same time. The sequence can be like this, rank 0 -> rank 1, rank 0 -> rank 2, rank 0 -> rank 3, rank 0 -> rank 4 and each takes 0.5 seconds. So total will be 2 seconds. However, each node is directly connected to the root so we avoid delays from message forwarding. Thus, star topology is better in general.

Time phase	Node 1	Node 2	Node 3	Node 4
1	0 -> 1			
2		0 -> 2		
3			0 -> 3	
4				0 -> 4



Question 1 – Part C

No, I want it to be sending in two directions as it will be more efficient and faster. The total time for it to be completed can be divided by half. For instance, sending in two directions can be rank 0 -> rank 1, rank 0 -> rank 4, rank 4 -> rank 3, rank 1 -> rank 2. Since, rank 0-> rank 1 and rank 0 -> rank 4, rank 4 -> rank 3 and rank 1 -> rank 2 can be sent at the same time phase. So the total time for rank 0 to broadcast to all nodes is 1 seconds as 2*0.5 seconds = 1 second which is faster than only sending message in one direction. However, We need to ensure that each node has a flag to indicate when it has received the broadcast, preventing other nodes from sending the message again.

Time phase	Node 1	Node 2	Node 3	Node 4
1	0 -> 1			0 -> 4
2		1 -> 2	4 -> 3	
3				
4				

Question 1 – Part D

I will design the program using ring topology

1. the root node sends the message to neighbouring node which is rank 1
2. rank 1 will then forward it to next node which is rank 2
3. then the program continues forwarding until rank 4

I won't use the star topology as if the program can only send one message throughout the whole network. If using star topology the flow is rank 0-> rank 1, rank 1-> rank 0, rank 0-> rank 2, rank 2-> rank 0, rank 0-> rank 3 and so on. So, it is inefficient as it always need to forward back to root node before it can forward to other nodes. This will increase the communication overhead as it requires to forward back to the root node before it can send to other nodes, so it is 2 times forwarding of the ring topology as ring topology needs to forward 4 times for 5 nodes, where star topology needs to forward 8 times for 5 nodes.

Question 2 – Part A

According to the amdahl's law the formula is 1/ (scomp + sfabric + p/n). The serial portion of the task is non parallelisable meaning it cannot be speed up by adding more processors. If the time spent on network is high the processor needs to wait for data from another which adds up to the serial portion of the task so making it slow down even with many processors. Thus, slow down the speed up. Here are the reasons:

1. Communication overhead, in parallel systems processes need to communicate with each other to exchange data or synchronise. If the communication cost is high then the time spent waiting for data from other processors increases

2. Increase network traffic, if more processors added in the future. It supposes will increase the speed up, according to the formula (P/N) when increase in N, P will decrease. But if the communication time doesn't decrease although the computational time is decrease, all of the processors are waiting for data from others. Hence, it eventually slowed down the speed up. Not to mention, more processors can lead to more communication.

Question 2 – Part B

S = 1/(scomp + sfabric + p/n)
4 = 1 / (scomp + sfabric + (1-s/ infi))
4 = 1 / (scomp + sfabric + 0)
Scomp + sfabric = ¼
Serial = ¼

Percentage = ¼ * 100 = 25%

Question 2 – Part C

This does not guarantee that Bob's program will achieve a speed up larger than 4 even though Alice upgraded the network latency and bandwidth. Although improving the network will reduce the communication overhead and potentially improve speed up. However, if the serial portion is significant it will still limit the maximum achievable speed up regardless the improvement of network communication. The overall speed up still depends on the size of serial portion and the parallelizable portion of the program. Therefore Bob can't achieve speed up larger than 4 with upgraded network communication.

MONASH University

Question 3 – Part A

The partitioning is not ideal as for the result n it needs n-1, n-2, n-3 to get the answer. So the answer is dependent on its previous value. So if Bob partition like this f(n) will need to wait for f(n-1), f(n-2), f(n-3) to complete before proceeding. Hence, it wasted the computational power as it makes the other cores to halt. The workload distribution is not efficient due to Bob is not utilising all 64 CPU cores as only a few cores can be actively computing for f(n) as for larger n it depends on the previous computed values. So the cores will be idle waiting to earlier computation to complete.

Time phase	Core 1	Core 2	Core 3
1	f(3)	f(4) idle	f(5) idle
2		f(4)	f(5) idle
3			f(5)
4			

To further illustrate with the table above, core 1 is computing f(3) but core 2 have to wait for core 1 to finish computing as f(4) is dependent on the result of core 1. Hence, it stays idle. Same goes to core 3. Thus, the workload distribution is not efficient as all the cores need to remain idle for their previous cores to be done before proceeding.

Question 3 – Part B

Not easy to be parallelised because the result of f(n) needs to wait for f(n-1), f(n-2) and f(n-3). Thus, if we assign f(n) to processor 1, f(n-1) to processor 2, f(n-2) to processor 3 and f(n-3) to processor 4. Processor 1 will remain idle while waiting processor 2,3 and 4 to finish their job. Thus, it has strong sequential dependencies as we cannot compute f(n) until we have f(n-1), f(n-2) and f(n-3) results. This limits the ability to perform computations in parallel.

Question 3 – Part C

A closed form expression eliminates this dependency issue as we get to know the values of f(n) by using the closed form expression. Thus, it can be computed independently. By having a closed form expressions it can reduce computation time as it has direct formula that can compute the result. Hence it is good for parallel computing as we can distribute n across all the processors and each of them can compute the answer independently without needing answers from other processors, meaning there are no dependencies between processors. Since each cores can be assigned an independent task so there no core is idle. Besides that, it reduces memory usage as it doesn't need to store the previous values to compute the next value so it reduces memory overhead. Hence, closed form expression is a better choice for parallel computing.