Participant Name: Pratibha Garg
Project Name: NOC Design
Participant Email ID: pratibhagarg0934@gmail.com
Participant GOC ID: 275863363070
Readme File (Github) : https://github.com/Pratibha0934/NOC/

Brief Summary (Problem Statement and Solution for both questions):

The problem entails optimizing a System on a Chip (SoC) architecture, focusing on its Network on Chip (NOC). The goal is to design an efficient NOC that ensures minimal latency, achieves 95% of maximum bandwidth, supports 90% buffer occupancy, and throttles only 5% of the time based on power thresholds. The setup includes a simulator with APIs for controlling buffer size, arbiter weights, and throttling. The NOC connects CPU, IO peripherals, and memory, with traffic patterns influenced by workload types. Efficient design is crucial to balance area, power, and performance for target workloads. The solution for this problem statement assumes that after each read and write operation there will be a token released by the system in the form of “Data” followed by the address of the memory address upon which the read or write instruction has been implemented. The solution aims to average out the write and read latency over time by evaluating the read count and write count and then evaluating latency by adding the difference in timestamps when token is received and the timestamp when instruction was received. Similarly, the code evaluates the bandwidth by evaluating how many bytes are being used for each instruction i.e, evaluating total bytes over total cycles.

For Reinforcement Learning, we need to define the MDP, which includes the states, actions, rewards and the transition model.

We want to achieve optimal values for Buffer Occupancy values for each buffer, arbitration rates for each arbiter,read and write latency for CPU and IO, Bandwidth of CPU and IO and the power threshold status

The transition model would be deterministic, changing the values of the state parameters as exactly defined by the actions.
The action space and state space would be continuous as well.

Considering the complexity and dynamic nature of the problem, an algorithm like Deep Q-Network (DQN) could be suitable for this RL framework. DQN is capable of handling complex state spaces and continuous actions, making it suitable for optimizing the NOC design parameters.

Pseudo Code for the problem statement:

Assumption: After each read and write operation there will be a token released by the system in the form of “Data” followed by the address of the memory address upon which the read or write instruction has been implemented. This will indicate that the instruction has been completely done.

```
Initialize total_read_latency, total_write_latency, total_bytes, read_count, write_count to 0
Initialize read_transactions, write_transactions to empty dictionaries

For each entry in the interface monitor output:
  If TxnType is 'Rd':
    Add the address and timestamp to read_transactions
  Else if TxnType is 'Wr':
```

Add the address and timestamp to write_transactions

For each data arrival entry in the interface monitor output:
If the TxnType is 'Data'
if address is in read_transactions:
Calculate latency as current timestamp - timestamp in read_transactions
Increment total_read_latency by latency
Increment read_count by 1
Increment total_bytes by 32 (assuming 32B per transfer)
Remove the address from read_transactions
else if address is in write_transactions:
Calculate latency as current timestamp - timestamp in write_transactions
Increment total_write_latency by latency
Increment write_count by 1
Increment total_bytes by 32 (assuming 32B per transfer)
Remove the address from write_transactions

total_cycles = total_read_latency + total_write_latency

Calculate average_read_latency as total_read_latency / read_count
Calculate average_write_latency as total_write_latency / write_count
Calculate bandwidth as total_bytes / total_cycle (total_cycles is the simulation time)

Output average_read_latency, average_write_latency, bandwidth

Reinforcement Learning
For Reinforcement Learning, we need to define the MDP, which includes the states, actions, rewards and the transition model.

States : S = {(BO), (AR), (RL), {WL), (B), P), where BO is the tuple containing Buffer Occupancy values for each buffer, AR is the tuples of arbitration rates for each arbiter, RL is the read latency for CPU and IO, WL is the write latency for CPU and IO, B is the Bandwidth of CPU and IO, O is the power threshold status

Actions : They involve changing buffer size, changing arbiter weights and changing the operating frequency of NOC. These actions are continuous.

Rewards : Rewards can be defined as -

- Positive rewards for getting latency closer to min_latency, and highly positive rewards for latency <= min_latency and negative rewards for going away from min_latency
- Positive rewards for getting bandwidth closer to the 95% of max_bandwidth, and negative rewards for going away from max_bandwidth
- Positive rewards for getting buffer occupancy closer to 90% occupancy, and negative rewards for going away.
- Negative rewards if throttling happens for more than 5 cycles on average.

The transition model would be deterministic, changing the values of the state parameters as exactly defined by the actions.

Complexity Analysis:
The loop iterates over all cycles in the interface_monitor_output, so its time complexity is O(n). Inside the loop, we do incrementation to variables and calculate differences between the timestamps, which take O(1) time. Checking the existence of addresses in the transaction dictionaries takes constant O(1) time as well. Hence the total time complexity would be constant O(1).
Thus the overall time complexity is O(n).

Proof Of Correctness:
The code, under the assumption that we receive token “DATA #Addr” for completion of instructions, works in the most optimal and robust manner to evaluate the read latency, write latency and bandwidth for the data transfer, for any set of instructions received in any order.

The positive rewards for achieving values closer to the desired values and negative rewards for going away from them, for the different parameters that we wish to optimize, ensures that we get converging optimal Q-value functions.

Alternatives considered:
For RL, we considered the other algorithms such as SARSA and Actor-critic, but DQN seemed to be the most suitable for continuous action spaces and the complex continuous state space.

References:

1. The Network-on-Chip Paradigm in Practice and Research: https://ieeexplore.ieee.org/abstract/document/1511971
2. n-Bit multiple read and write FIFO memory model for network-on-chip: https://ieeexplore.ieee.org/document/6141440
3. DQN implementation for Atari games: http://arxiv.org/abs/1312.5602v1