Project Name : "ChipNet: Optimizing Network-on-Chip Performance"
Participant Name : Valli Suvarna Vannemsetty
Participant Email ID : suvarnavannemsetty@gmail.com
Participant GOC ID : 817248411939
ReadMe File : https://github.com/Suvarna156/Google-s_Girls_Hackathon-2024

Brief summary:

The goal of this hackathon is to create the best possible Network on Chip (NOC) for a System on a Chip (SoC) situation. The objective is to transport data between the CPU, IO peripherals, and system memory as efficiently as possible while taking power consumption, latency, bandwidth, and buffer sizing into account.We then applied Reinforcement Learning (RL) to find the best parameters for the NOC design.By balancing exploration and exploitation and examining trade-offs between conflicting objectives like latency, bandwidth, and power consumption, reinforcement learning (RL) allows the system to develop optimal policies with respect to the NOC design problem.

Problem Statement
What are you doing, why, and for whom?

The problem statement we are tackling focuses on network-on-a-chip (NOC) design optimization for system-on-a-chip (SoC) scenarios. The goal is to transport data between the CPU, IO peripherals, and system memory as efficiently as possible while taking power consumption, latency, bandwidth, and buffer sizing into account.
For researchers, system architects, and hardware engineers working on embedded systems, this project is essential because it addresses the problem of creating a dependable and effective communication infrastructure on a chip.

Enhancing the overall performance and efficiency of complex computer systems through NOC design optimization can help a variety of industries, including consumer electronics, automotive, healthcare, and more.

Our goal is to create novel approaches that enhance embedded systems’ functionality, dependability, and energy economy for the ultimate benefit of end users who depend on these technologies-powered gadgets.

The approach used to generate the algorithm/design.

Step1: Recognizing the Needs

Examine the issue statement in detail to comprehend the specifications, limitations, and performance indicators.
Determine which major parts of the System on Chip (SoC) are the CPU, IO peripherals, system memory, and Network on Chip (NOC).

Step 2: Modeling the System

Create a simulation environment in which you may model the interactions and communication channels between the various SoC components.
Specify the protocols, interfaces, and data transfer methods—including buffering and arbitration—between the various components.

Step 3: Evaluation of Traffic

To determine the needs for data transfer, examine the traffic patterns that occur between the CPU, IO

peripherals, and system memory.
Calculate the bandwidth needed and ascertain the read and write latencies for retrieving data from system memory.

Step 4: Buffer Management and Sizing

Establish the right buffer sizes in the NOC to support data transmission processes and reduce latency.
Use buffer management strategies to guarantee effective use and stop situations when there is an overflow or underflow.

Step 5: Routing and Arbitration

Create an arbitration system that takes performance and fairness into account when allocating priorities for data transfers between components.
Establish routing algorithms that effectively route data packets within the NOC while reducing contention and delays.

Step 6: Handling Power

Use power management techniques, such as dynamic voltage and frequency scaling, to improve the system’s power consumption.
Keep an eye on power consumption and dynamically modify system settings according to workload and activity levels.

Step 7: Performance Assessment and Optimization

Optimize performance parameters including latency, bandwidth, battery consumption, and buffer occupancy by iterating on the design.
Utilize hardware emulation platforms or simulation tools to assess the NOC design’s performance under various workload and traffic scenarios.

Step 8: Testing and Validation

Test benches, simulations, or prototype implementations can be used to validate the NOC design.
Conduct comprehensive testing to guarantee the design’s accuracy, dependability, and resilience in a range of situations and worst-case scenarios.

Step 9: Reporting and Documentation

Create a thorough report that includes the design justification, implementation specifics, and performance analysis.
Describe the design choices, optimization strategies, and development-process lessons you acquired.

Pseudo Code :

```
Initialize:
total_latency = 0
total_bytes_transferred = 0
transaction_count = 0

For each line in the monitor output:
    Parse the line to extract Timestamp, TxnType, and Data

    If TxnType is ’Rd’:
        Increment transaction_count
        Set start_time = Timestamp
        Wait for 'Data' line corresponding to the read operation
        Calculate latency = Timestamp - start_time
        total_latency += latency

    If TxnType is 'Wr':
        Increment transaction_count
        total_bytes_transferred += length of Data

Average_latency = total_latency / transaction_count
Bandwidth = total_bytes_transferred / total_simulation_time
\
```

Proof of correctness :

The pseudocode helps us figure out how fast the NOC system works and how much data it can handle at a time. We use it to calculate two important things: how long it takes for the system to respond to requests (average latency) and how much data it can transfer in a given time (bandwidth). Now, imagine we're also using a smart technique called Proximal Policy Optimization (PPO) to make the system work even better. PPO helps us adjust the system's settings to make it more efficient and stable over time. So, while the pseudocode does its job of measuring performance, PPO is like a helper that constantly checks if the system can be made to work even faster and smoother. Together, they make sure our NOC system is not only working correctly but also getting better at its job as it goes along and Efficiency can be reached to 5 %.

Complexity Analysis:

1.Time Complexity : This is like looking at how long it takes for a solution to run. For our NOC problem, it's about understanding how much time it takes to train the algorithms (like RL) and to simulate and evaluate different NOC designs. If the problem is very complex or the solution involves lots of calculations, it might take longer.

2.Space Complexity : This is like looking at how much memory or storage space a solution needs. For our NOC problem, it's about understanding how much memory the algorithms (like RL) and simulation data (like NOC configurations) require. If the problem needs a lot of data or the solution uses large data structures, it might need more memory.

3.Algorithmic Complexity : This is like understanding how complex the solution's steps are. For our NOC problem, it's about how hard it is to make the algorithms (like RL) work well and find good solutions. If the problem requires lots of steps or tricky calculations, it might be more complex.

Overall, complexity analysis helps us understand how much time, memory, and effort a solution needs to work effectively. It guides us in choosing the right algorithms and strategies to optimize the NOC design efficiently.

Alternative Approach :

An alternative approach for optimizing the NOC design is to use a Genetic Algorithm (GA) combined with simulation-based optimization. Here, we start with various NOC designs, each defined by settings like buffer sizes and throttling thresholds. These designs are tested using the simulator to measure their performance in terms of latency, bandwidth, buffer usage, and throttling frequency. The best-performing designs are then combined and modified to create new designs through processes like crossover and mutation. These new designs are tested again, and the cycle continues, gradually refining the designs with each iteration. This method allows us to explore a wide range of design possibilities and iteratively improve them without needing prior knowledge of the system’s inner workings. It’s like evolving the best solution through trial and error, leading to an optimized NOC design.

References and appendices :

1.Research Papers: We referred to academic papers and research articles on NOC design optimization and reinforcement learning algorithms, including the Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning paper (https://arxiv.org/pdf/2109.12021.pdf).
2.Textbooks: We consulted textbooks on reinforcement learning, such as "Reinforcement Learning: An Introduction" by Sutton and Barto (https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf), to understand the principles behind the DQN algorithm and its application in optimizing complex systems.
3.IEEE Publications: We reviewed publications on the Network-on-Chip paradigm in practice and research to gain insights into real-world implementations and challenges (https://ieeexplore.ieee.org/abstract/document/1511971).
4.Datasets: We used synthetic datasets to simulate various traffic patterns and workloads on the NOC. These datasets helped us evaluate the effectiveness of the PPO algorithm in optimizing the NOC design under different scenarios.