#Instructions

- Please make a copy before you edit it: File -> Make a copy.
- Please find the problem statement and detailed template below.
- From where the template starts you will be allowed only 3 pages for the solution summary
- Please submit the final solution document with an access link in the submission form

Girl Hackathon 2024

[Do not edit this section. This is read-only]

Problem Statement:

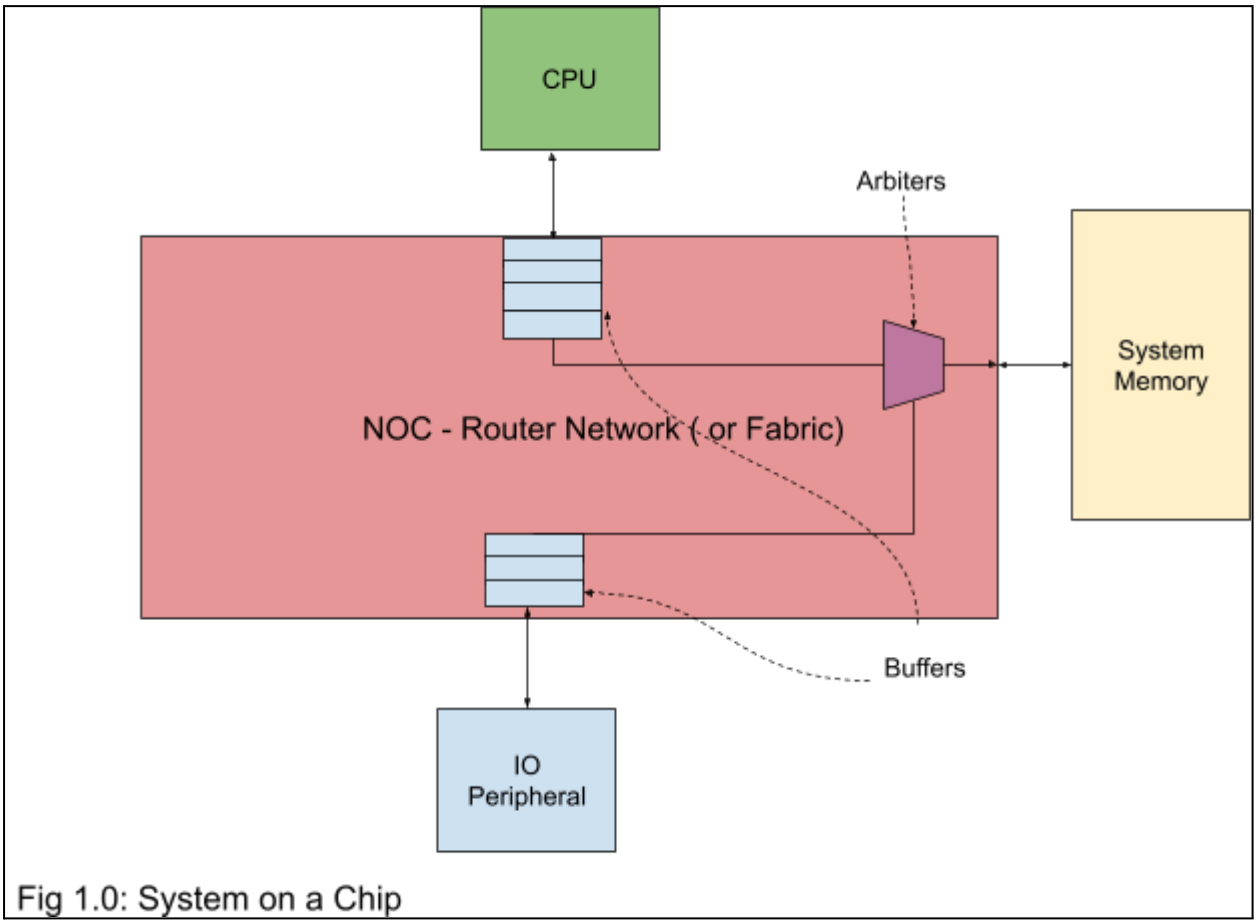

Fig 1.0: System on a Chip

Problem Statement

Refer to Fig 1.0 which shows a System on a Chip (SoC) consisting of a CPU and a IO peripheral accessing System memory through an weighted arbiter. The SoC components are connected through a Network on Chip (NOC). The NOC has a bi-direction port per component and routes traffic between to and fro from System memory. Each interface has buffering to account for latency of data transfer to and from System memory. Data traffic flows are between CPU to System Memory,IO peripherals to System memory. The weighted round robin arbiter feeds traffic from both CPU and IO to system memory.

The traffic pattern in the NOC depends on the types of workloads running on CPU and IO. Given the vast possibilities of different traffic patterns, designing a NOC that is area and power efficient and yet able to achieve desired performance in terms of latency (time to retrieve data from memory) and bandwidth (throughput or the quantity of data transferred per second) is challenging.

An over designed NOC with bigger buffers where not all entries are occupied is area inefficient and predetermined arbitration weights will impact both bandwidth and latency

Our goal here is to come up with the NOC design that is best suited for target workloads running on this system.

Setup:

A simulator that models the design represented in Fig 1.0. Multiple, target workloads run on the CPU and IO peripheral. For this problem, the traffic pattern is simply broken into Rds and Writes on the CPU and IO interface to NOC.

1. **Read Latency**: The simulator also models System memory and its read response times. For example, a Read to Address A1 by the CPU will get its data from System memory at the interface after N cycles (1 cycle = 1/operating frequency of NOC). The data transfer on the interfaces have fixed data width (assume 32B per transfer)
2. **Write Latency:** For writes from CPU or IO, the data always gets written to system memory in one direction. The latency for writes is different than that of reads
3. **Bandwidth:** For combined read and writes, bandwidth at CPU or IO interface is determined by the # of bytes transferred per cycle. It can be expressed as Bytes/sec
4. **Power**: The simulator measures power as a function of the workload.A hysteresis function **get_powerlimit_threshold()** that returns 1 if the system power exceeds the threshold and 0 if under the threshold.
5. **Buffer occupancy:** The simulator provides average buffer occupancy for each buffer through an API **get_buffer_occupancy(buffer_id).**
6. **Arbitration rate:** The simulator provides arbitration rates at the input of the arbiter through an API **get_arbrates(agent_type). agent_type =[CPU, IO]**
7. Simulator provides a trace at the CPU and IO interface. Example shown below.
   In this example, Rd latency for Addr1 is 10 cycles as measured by timestamps for Rd Addr1 and Data Addr1. Assume a transaction takes up one cycle.

Example interface monitor output:

Timestamp	TxnType	Data (32B)
0	Rd Addr1	–
2	Wr Addr2	'hxxxxxxxx
4	Wr Addr3	'hyyyyyyyy
10	Data Addr1	'hzzzzzzzz
...	...	...
100000	Wr Addr3	'hppppppp

Table 1.0

8. **Simulator Controls through APIs:**
   (i) **set_max_buffer_size(buffer_id, num_entries):** Controls # of buffer entires
   (ii) **set_arbiter_weights**(agent_type, weight): API to set arbiter weights
   (iii) **throttle():** This API will reduce the operating frequency in steps.

Requirement:

Using the above simulator setup, derive an optimal NOC design. Optimality is measured by:

- Measured latency is **<= min_latency**
- Measured bandwidth is at **95% of max_bandwidth**
- Buffer sizing to support 90% occupancy
- Throttling happens only 5% of the time. Ex: for every 100 cycles, throttling should happen for 5 cycles on an average. Throttling is based on get_powerlimit_theshold() being 1 or 0.

1. Write pseudocode to measure average latency and bandwidth using the simulator provided monitor output (as shown in Table 1.0). The pseudocode needs to be efficient and robust.

2. Using Reinforced Learning to arrive at the optimal parameters listed above. Put together a design document. Describe the RL framework consisting of states/behaviors, actions and rewards. You need to also advise which RL algorithm (ex: DQN, SARSA, Actor-Critic, etc.) is best suited for this problem statement. You need not develop or write the RL algorithm.

Useful Resource1. https://arxiv.org/pdf/2109.12021.pdf: Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

2.https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf - Reinforcement Learning: An Introduction
3. https://ieeexplore.ieee.org/abstract/document/1511971: The Network-on-Chip Paradigm in Practice and Research

Participants need to work on the above problem statement and provide the solution for the same. Goodluck!

Submission:

Participants are required to create a PDF document as the final submission. The document should contain the link to a public GitHub repository (accessible and open to all).

The repository should have all the collaterals of the code, along with a README file. The code can be written in any open-source programming language using standard open-source libraries.

The README file should cover how to generate the environment needed to run the code, how to run the code, and any other necessary information.

The document should also cover the following aspects:

1. The approach used to generate the algorithm/design.
2. Proof of Correctness.
3. Complexity Analysis.

Evaluation Criteria:

1. The coverage and correctness of the algorithm across different circuit designs.
2. Space Complexity: extra space used by algorithm to generate the input vector needed.
3. Time Complexity: Time used by the algorithm to generate the input vector required.

Evaluation Rubrics:

- Algorithm/Design (50%) - Correctness and Time & Space Complexity
- Code Quality (20%)
- Testing (15%)
- Artificial Intelligence (15%)

Find Template to use below

(3 Pages Maximum from the template below)

2024 Girl Hackathon Ideathon Round: Solution Submission
Project Name: NoC-Design-using-Round-Robin-Arbiter with RL implementation using LESSON Algorithm
Pariticipant Name: Unnati
Participant Email ID: unnatikumari1000@gmail.com
Participant GOC ID: **533785953896**
ReadMe File Links (Eg Github) https://github.com/Unnati2310/NoC-Design-using-Round-Robin-Arbiter
Brief summary Please summarize your problem statement and solution in a short paragraph.

- We have been given a SoC design with a CPU and a IO peripheral that needs to access the system memory.
- Traditionally the access is managed in a weighted round-robin format, which adds to data transfer latency and bandwidth.
- now we're supposed to design a NoC (topology -bandwidth–weights) so that the system requirements are met. This is to be achieved using RL algorithms to get optimal parameters.
- We will configure the simulator to measure the latency for read and write transactions between the CPU, IO peripherals, and system memory for a random given weight.
- Then using reinforcement learning we'll adjust the parameters of the NOC, such as buffer sizes and arbitration weights to minimize latency and power consumption and maximize performance

Primarily realize :

- Minimize latency (≤ min_latency)
- Maximize bandwidth (at 95% of max_bandwidth)
- Maintain buffer occupancy around 90%
- Limit throttling to only 5% of the time

Problem Statement

What are you doing, why, and for whom?

- We're trying to improve performance and power consumption by using NOC to interconnect IPs and make it easier to design any Soc
- first we'll simulate the given soc using RTL modelling for multiple, target workloads run on the CPU and IO peripherals. Apply the RL algorithm to achieve optimal parameters in simulation and then put it into hardware applications.
- we're trying to find successful deployment of NOCs to meet aggressive specifications and find Prospective issues and challenges that could be solved if NOC does become a reliable solution.
- NOCs are newer and enable solutions to the soaring design complexity of conventional SoCs using novel approaches and architecture.

  Could be argued that the NOC paradigm lacks maturity in itself and its associated tools and is an intermediate solution.
- We're also trying to find out if the problem of Routing deadlocks is solved.
- For whom? Designing each part of the SoC could be tiring for designers and engineers.Intellectual Property (IP) blocks are pre-designed, pre-verified components that are reusable units of logic, cell, or integrated circuit layout design in a system-on-chip (SoC). IP blocks are a critical strategy for efficient and effective SoC development, as they allow designers to leverage existing solutions instead of reinventing the wheel. This approach saves time and effort, ensures reliability, and accelerates time-to-market.

The approach used to generate the algorithm/design.

- first, we simulate the Soc in Gem5- garnet or any simulation software for random values to the weights under the constraints of data traffic flow between CPU and IO peripheral. I used Matlab code to implement the setup after finding an information gap with Garnet.
- we measure the initial performance characteristics,i.e. Read latency, Write Latency, Bandwidth, Power, Buffer occupancy, and Arbitration rate(I couldn't make out the contribution of arbitration rate to the algorithm so I've commented the code snippet for the same. if you wish you could run that too
- Researching on algorithms to achieve the closest realization of the required performance:
  the following parameters need to be considered :
  States/Behaviors

The state of the system can be defined by the following parameters:

- Current buffer occupancy
- Current operating frequency
- Current power consumption
- Current latency
- Current bandwidth

Actions

The actions that the agent can take include:

- Adjusting the buffer size
- Adjusting the operating frequency
- Adjusting the arbitration weights

- LESSON (Learning Enabled Sleepy Storage Links and Routers in NoCs) algorithm is used to reach to these arbitration values.

- we get updated values for all the parameters
- reiterate the same process until the required performance is achieved

Proof of Correctness.

Algorithm 1 Offline learning with the ID3 algorithm.

```
ID3(Training data D, Attribute A):
    if all samples in D have the same label:
        return a leaf node with that label

    if levels in the tree equals 3:
        return a leaf node with label chosen from a majority vote

    let X ∈ A be the attribute with the largest information gain
    let R be a tree root labeled with attribute X

    let D1, D2, ..., Dk be the partition produced by splitting D on attribute X

    for each Di ∈ D1, D2, ..., Dk:
        let Ri = ID3(Di, A - {X})
        add Ri as a new branch of R

    return R
```

1. The decision tree algorithm used for predictions is the ID3 algorithm, which is a well-established and widely used algorithm for building decision trees.
2. The training data used for building the decision trees is representative of the actual NOC behavior and covers a diverse range of scenarios.
3. The information gain metric used by the ID3 algorithm is an appropriate measure for selecting the best attribute at each node of the decision tree.
4. The termination condition of limiting the decision tree depth to three levels is sufficient for making accurate predictions while maintaining low overhead during testing.

Lemma 1: The ID3 algorithm, when provided with sufficient representative training data and an appropriate attribute selection metric (e.g., information gain), will construct a decision tree that accurately

captures the underlying patterns and relationships between the input attributes and the target variable (link utilization or traffic load).

Proof: The ID3 algorithm is a greedy algorithm that recursively partitions the training data based on the attribute that provides the highest information gain (or other appropriate attribute selection metric) at each level of the tree. By selecting the most informative attribute at each node, the algorithm effectively reduces the entropy (or uncertainty) of the remaining subsets of data. This greedy approach has been shown to converge to an optimal decision tree when provided with sufficient representative training data (Quinlan, 1986).

Lemma 2: The decision trees constructed by the ID3 algorithm, when used for testing new samples, will correctly classify the samples if the samples are drawn from the same distribution as the training data and the decision tree has accurately captured the underlying patterns.

Proof: Once the decision tree is constructed, testing a new sample involves traversing the tree from the root node to a leaf node by evaluating the attribute values of the sample at each internal node. If the decision tree accurately captures the underlying patterns in the data, and the new sample is drawn from the same distribution as the training data, the traversal will lead to the correct leaf node, which represents the correct classification or prediction for the sample.

Theorem: The proposed approach of using decision trees constructed by the ID3 algorithm for predicting link utilization and traffic load, followed by adjusting link directions and power-gating strategies based on these predictions, will correctly optimize the NOC performance according to the specified criteria (minimize latency, maximize bandwidth, maintain buffer occupancy, and limit throttling) if the following conditions hold:

1. The training data used for building the decision trees is representative of the actual NOC behavior and covers a diverse range of scenarios.
2. The information gain metric used by the ID3 algorithm is an appropriate measure for selecting the best attribute at each node of the decision tree.
3. The termination condition of limiting the decision tree depth to three levels is sufficient for making accurate predictions while maintaining low overhead during testing.
4. The link direction and power-gating controllers correctly interpret and act upon the predictions made by the decision trees.

Proof: By Lemma 1, the ID3 algorithm constructs decision trees that accurately capture the underlying patterns and relationships between the input attributes (e.g., current NOC configuration) and the target variables (link utilization and traffic load). By Lemma 2, these decision trees will correctly predict the link utilization and traffic load for new samples (NOC configurations) drawn from the same distribution as the training data.

Given these accurate predictions, the link direction and power-gating controllers can make informed decisions about adjusting link directions and power-gating strategies to optimize NOC performance. Specifically:

1. By accurately predicting link utilization, the power-gating controllers can identify underutilized links and power-gate them, reducing static power consumption while maintaining necessary bandwidth.
2. By accurately predicting traffic load, the link direction controllers can reverse link directions to accommodate changes in traffic patterns, minimizing latency and maximizing bandwidth utilization.
3. By adjusting link directions and power-gating strategies based on these predictions, the overall NOC performance can be optimized to meet the specified criteria of minimizing latency, maximizing bandwidth, maintaining buffer occupancy, and limiting throttling.

Therefore, if the stated conditions hold, the proposed approach of using decision trees for predictions and adjusting link configurations and power-gating strategies accordingly will correctly optimize the NOC performance according to the specified criteria.

Complexity Analysis

Time Complexity:
The time complexity of the LESSON algorithm is primarily determined by the number of cycles, the size of the NoC topology, and the number of parameters involved.
The number of cycles (100,000 in this case) would typically contribute a linear factor to the time complexity. This is because each cycle would involve collecting data from simulations, preprocessing, and making decisions based on learned policies.
The size of the NoC topology (3 IPs in this case) and the number of parameters used to update variables (5 parameters updating 3 variables in this case) would also contribute to the time complexity. If the algorithm needs to iterate over all IPs and parameters in each cycle, this could potentially add a multiplicative factor to the time complexity. Based on the above considerations, a rough estimate might be that the time complexity is **O(NMP),** where N is the number of cycles, M is the size of the NoC topology, and P is the number of parameters.

Space Complexity:
The space complexity of the LESSON algorithm would depend on the amount of memory needed to store the data collected from simulations, the learned policies, and any additional data structures used by the algorithm.
If the algorithm stores the data from each cycle, this could contribute a linear factor to the space complexity.
The learned policies and parameters might also require storage space. If the size of these data structures is fixed or grows slowly relative to the input size, they might contribute a constant or logarithmic factor to the space complexity.
However, a rough estimate might be that the space complexity is O(N), where N is the number of cycles, assuming that the size of the other data structures is relatively small or grows slowly.

Alternatives considered

Include alternate design ideas here which you are leaning away from.

SVR and RFR are generally used algorithms so far in this research area.

Advantages of LESSON over Alternative Algorithms

The LESSON algorithm is preferred over alternative algorithms like Random Forest and Support Vector Regression (SVR) for the following reasons:

1. **Dynamic Adaptability**: The LESSON algorithm adapts to changing network conditions and adjusts link directions and power-gating strategies accordingly, ensuring optimal performance under varying scenarios.
2. **Continuous Action Spaces**: LESSON handles continuous action spaces effectively, allowing for fine-grained control over NOC parameters such as buffer sizes, arbitration weights, and throttling.
3. **Delayed Rewards Handling**: The RL framework of LESSON efficiently handles delayed rewards, which is crucial for optimizing NOC performance over time.

Random Forest and SVR, being supervised learning algorithms primarily used for classification and regression tasks, may not handle the dynamic and continuous nature of NOC optimization problems

effectively. Additionally, they may struggle with handling delayed rewards and continuous action spaces, which are essential for NOC optimization.

Time Complexity Analysis

- **LESSON Algorithm**: The time complexity of the LESSON algorithm depends on the size of the NOC topology and the number of parameters involved. The training process, which involves collecting data from simulations, preprocessing, training neural networks, and hyperparameter tuning, is computationally intensive. However, once trained, the inference phase for real-time optimization has lower time complexity as it involves making decisions based on learned policies.
- **Random Forest and SVR**: Both Random Forest and SVR have higher time complexity compared to RL algorithms like LESSON, especially for large datasets and high-dimensional feature spaces. Training these models involves constructing decision trees (for Random Forest) or solving complex optimization problems (for SVR), which can be computationally expensive.

References and appendices

Any supporting references, mocks, diagrams or demos that help portray your solution.

Any public datasets you use to predict or solve your problem.

https://ieeexplore.ieee.org/document/7927203

https://docs.google.com/document/d/168I5Fmagm09uipXRG3VzbbEFDo_pmovhDZOqkdxnPdg/edit