Toward More Efficient NoC Arbitration: A Deep Reinforcement Learning Approach

JIEMING YIN, YASUKO ECKERT, SHUAI CHE, MARK OSKIN, GABRIEL H. LOH
JUNE 3, 2018
Machine learning benefits from computer architecture
- Faster CPU and GPU
- ASIC

Why not take advantage of machine learning and design better hardware?
NETWORK ON CHIPS

- Scalable solution for larger chips
  - Communication by packets
  - Routing of packets through routers
  - Efficient on-chip resource sharing
Use machine learning for feature selection off-line, instead of building a neural network in H/W
REINFORCEMENT LEARNING

- Unsupervised learning
- Agent interacts with the environment
- Agent learns the actions that lead to maximum long-term reward
- Environment returns a numerical reward to the agent for each action it takes

How to relate NoC arbitration to reinforcement-learning?
**REINFORCEMENT LEARNING-BASED NOC ARBITRATION MODEL**

- **Environment**: NoC
- **Agent**: recommending system for the arbiter (global agent)
- **State**: state of a router
- **Action**: select one input buffer
- **Reward**: a certain metric (e.g., network throughput)
**STATE REPRESENTATION**

- **Router state vector**
  - One vector per output port
  - Features: message type, global age, local age, distance, hop count

![Diagram of router state vector](image)

Output 1 state vector:

<table>
<thead>
<tr>
<th>Features for In1-1</th>
<th>Zero inputs for In1-2</th>
<th>Features for In2-1</th>
<th>Zero inputs for In2-2, In3-1, In3-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3 ... 0.7</td>
<td>0 ... 0</td>
<td>0.6 ... 0.2</td>
<td>0 ... 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.6 ... 0.2</td>
<td>0 ... 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Output 2 state vector:

<table>
<thead>
<tr>
<th>Zero inputs for In1-1, In1-2, In2-1</th>
<th>Features for In2-2</th>
<th>Features for In3-1</th>
<th>Zero inputs for In3-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ... 0</td>
<td>0.3 ... 0.4</td>
<td>0.5 ... 0.1</td>
<td>0 ... 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0.3 ... 0.4</td>
<td>0.5 ... 0.1</td>
<td>0 ... 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
ACTION SELECTION

Select the input buffer with the largest output value

Output 1 state vector:

```
0.3 ... 0.7 0 ... 0 0.6 ... 0.2 0 ... 0 0 ... 0 0 ... 0
```

Agent Neural Network (MLP)

Input values:

- In1-1: 9.7
- In1-2: 2.1
- In2-1: 8.6
- In2-2: 8.3
- In3-1: 9.8
- In3-2: 0.4
REWARD CRITERIA
- Reciprocal of average packet latency
- Global link utilization
- Fixed reward of 1
Separate target network to stabilize training \[1\]
Experience replay to break similarity of subsequent training samples \[2\]

EVALUATION

- Gem5 and Garnet
- NoC configuration:
  - 4x4 mesh topology
  - 1 VC per message class
- Workload
  - Synthetic traffic
    - Uniform Random, Bit-complement, Transpose
    - Request, data response, probe request
  - Operate at saturation point
  - 2-million-cycle epochs
- Agent neural network
  - Input layer: 90 neurons (6 ports * 3 message classes * 1 VC/class * 5 features)
  - 1 hidden layer: 54 neurons, Sigmoid
  - Output layer: 18 neurons, ReLU
- Comparison point
  - Round-robin arbitration
  - Idealized age-based arbitration [1]

[1] M. M. Lee et al., Probabilistic Distance-Based Arbitration: Providing Equality of Service for Many-Core CMPs. MICRO 2010
TRAINING RESULTS: AVERAGE LATENCY

Uniform Random
RR: 4855.8
Age-based: 28.7
RL-based: 56.1

Bit-complement
RR: 5198.6
Age-based: 24.7
RL-based: 36.9

Transpose
RR: 3600.8
Age-based: 19.8
RL-based: 41.8

Potential for exploring useful features, filtering out less valuable features.
SENSITIVITY STUDY

- Sensitivity on reward criteria
- Throughput studies
- Different network configurations
- Applying trained neural network to different configurations
CONCLUSION AND FUTURE WORK

- A first step in applying reinforcement learning in NoC arbitration
- Demonstrate effectiveness in feature exploration

- Explore more features
- Neural network interpretability
- Design efficient arbitration policy based on features
Thank You
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREOF, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.
Q-LEARNING

- Q-function: maximum expected “Future Discounted Reward”
  \[ Q(s_t, a_t) = \max(R_{t+1}) \]
  \[ R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \quad (\gamma: \text{discount factor}) \]

- Get the Q-function by experience using Bellman Equation
  \[ Q(s_t, a_t) \leftarrow r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \]

- Table-based Q-learning
- Deep Q-learning
TRAINING RESULTS: SENSITIVITY OF REWARD