Wireless Network-on-Chip using Deep Reinforcement Learning

Suraj Jog

Zikun Liu, Antonio Franques, Vimuth Fernando, Sergi Abadal, Josep Torrellas, Haitham Hassanieh
Network-on-Chip architectures have been instrumental to multicore processors

1. Fast Synchronization Between Parallel Threads
2. Data Sharing between cores for Cache Coherency

Intel Teraflops Research Chip (2007)
World’s first programmable 80-core chip to deliver more than 1 Teraflops compute
NoC’s ability to ensure cache coherency and synchronization is slower than execution on each core.

Speedup gained by parallelism is outweighed by the wired network’s communication cost for keeping caches coherent.
Augment NoC with Millimeter-wave Transceivers

1. **Latency**: Enables every core to reach every other core in just 1-hop
Augment NoC with Millimeter-wave Transceivers

1. **Latency**: Enables every core to reach every other core in just 1-hop

2. **Broadcast**: Local changes in the cache of a core can be instantaneously replicated at all other cores with single packet transmission
Prototypes of wireless NoC transceivers that deliver multi-Gbps links while imposing low overhead.

60 GHz Fully Integrated Transceiver System
Prototypes of wireless NoC transceivers that deliver multi-Gbps links while imposing low overhead.

- **IEEE JSSC 2010**
  - 1 Gbps 60 GHz OOK Transceiver
  - 800 um

- **IEEE JSSC 2010**
  - Board Assembly with Folded Dipole Antenna
  - 1300 um

- **IEEE EuMIC 2014**
  - 6 Gbps 90 GHz Transceiver
  - 450 um

- **IEEE Trans Circuits Syst I 2015**
  - 18.7 Gbps 60 GHz OOK Demodulator
  - 530 um
A Key Bottleneck is Efficient Medium Access Design

A. Adapting to Dynamic Traffic Patterns

Traffic patterns can change drastically across different cores, different time intervals, and different applications.
B. Complex Dependencies from Synchronization Primitives

- Barriers and locks lead to non-trivial relationship between delivery time of packet on NoC and progress of execution

A Key Bottleneck is Efficient Medium Access Design
A Key Bottleneck is Efficient Medium Access Design

B. Complex Dependencies from Synchronization Primitives

• Barriers and locks lead to non-trivial relationship between delivery time of packet on NoC and progress of execution
A Key Bottleneck is Efficient Medium Access Design

B. Complex Dependencies from Synchronization Primitives

- Barriers and locks lead to non-trivial relationship between delivery time of packet on NoC and progress of execution

Traditional MAC protocols treat each packet as equally important
A Key Bottleneck is Efficient Medium Access Design

B. Complex Dependencies from Synchronization Primitives

- Barriers and locks lead to non-trivial relationship between delivery time of packet on NoC and progress of execution

Hand-Tuned protocols cannot optimize for these complex hard-to-model dependencies
Can we learn protocols that can adapt to dynamic traffic while optimizing for complex dependencies?
NeuMAC

Leverages Deep RL to generate new MAC protocols that can learn structure in traffic patterns and dynamically adapt the protocol.
Translation of Deep RL output to a MAC protocol

Design Constraints:

1. Can’t run deep inference every CPU time slot
   • Past work on RL for MAC does inference per packet
     \cite{ICA3PP2013, IEEEJSAC2019, ComputerCommunications2020, Electronics2020}

2. Versatile enough to emulate large span of protocols

3. Should use reasonable number of parameters
NeuMAC’s 2 Layer Parameterized Protocol

• **First Layer:** Underlying TDMA schedule

2-Layer Protocol Design

**Deterministic TDMA Scheme**

<table>
<thead>
<tr>
<th>Clock Cycles</th>
<th>t = 1</th>
<th>t = 2</th>
<th>t = 3</th>
<th>t = 4</th>
<th>t = 8</th>
<th>t = 9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NeuMAC’s 2 Layer Protocol Design

- **Second Layer**: Probabilistic CSMA schedule Overlayed

- NeuMAC assigns contention probabilities $p_i$ to cores for opportunistic channel capture

- Number of parameters is restricted to $N$ for an $N$ core processor

2-Layer Protocol Design

### Probabilistic CSMA Scheme Overlayed

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$p_4$</th>
<th>$p_5$</th>
<th>$p_6$</th>
<th>$p_7$</th>
<th>$p_8$</th>
<th>$p_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Deterministic TDMA Scheme

- Clock Cycles: $t = 1, 2, 3, 4, 8, 9$
NeuMAC’s 2 Layer Protocol Design

- All $p_i = c > 0 \rightarrow$

2-Layer Protocol Design

<table>
<thead>
<tr>
<th>Probabilistic CSMA Scheme Overlaid</th>
<th>Deterministic TDMA Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1, p_2, p_3$</td>
<td>$t = 1$</td>
</tr>
<tr>
<td>$p_4, p_5, p_6$</td>
<td>$t = 2$</td>
</tr>
<tr>
<td>$p_7, p_8, p_9$</td>
<td>$t = 3$</td>
</tr>
<tr>
<td>$p_4, p_5, p_6$</td>
<td>$t = 4$</td>
</tr>
<tr>
<td>$p_7, p_8, p_9$</td>
<td>$t = 8$</td>
</tr>
<tr>
<td>$p_7, p_8, p_9$</td>
<td>$t = 9$</td>
</tr>
</tbody>
</table>

Clock Cycles
NeuMAC’s 2 Layer Protocol Design

- All $p_i = c > 0 \rightarrow$ CSMA Protocol with tunable aggression
- All $p_i = 0 \rightarrow$

2-Layer Protocol Design

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$ $p_2$ $p_3$</td>
<td>$p_4$ $p_5$ $p_6$</td>
<td>$p_7$ $p_8$ $p_9$</td>
<td>$p_1$ $p_2$ $p_3$</td>
<td>$p_4$ $p_5$ $p_6$</td>
<td>$p_7$ $p_8$ $p_9$</td>
<td>$p_1$ $p_2$ $p_3$</td>
<td>$p_4$ $p_5$ $p_6$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deterministic TDMA Scheme</th>
<th>Deterministic TDMA Scheme</th>
<th>Deterministic TDMA Scheme</th>
<th>Deterministic TDMA Scheme</th>
<th>Deterministic TDMA Scheme</th>
<th>Deterministic TDMA Scheme</th>
<th>Deterministic TDMA Scheme</th>
<th>Deterministic TDMA Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clock Cycles: $t = 1$ $t = 2$ $t = 3$ $t = 4$ $t = 8$ $t = 9$
NeuMAC’s 2 Layer Protocol Design

- Can gracefully shift from TDMA to CSMA scheme while supporting all intermediate protocols
- Allows fine grained control to each core’s action so as to generate highly optimized protocols

2-Layer Protocol Design

<table>
<thead>
<tr>
<th>Clock Cycles</th>
<th>t = 1</th>
<th>t = 2</th>
<th>t = 3</th>
<th>t = 4</th>
<th>t = 8</th>
<th>t = 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic TDMA Scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Probabilistic CSMA Scheme Overlayed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Translation of Deep RL output to a MAC protocol

Design Constraints:

1. Can’t run deep inference every CPU time slot
2. Versatile enough to emulate large span of protocols
3. Should use reasonable number of parameters
RL Formulation

Passively Observes State $s_t$

Policy Network

NeuMAC Agent

Action $a_t$

Wireless Network-on-Chip
RL Formulation

Passively Observes State $s_t$

Policy Network

NeuMAC Agent

Action $a_t$

Wireless Network-on-Chip
RL Formulation

Passively Observes State $s_{t+1}$

NeuMAC Agent → Policy Network → Action $a_t$ → Wireless Network-on-Chip
We use end-to-end execution time as reward $r_t$ instead of network metrics.
Evaluation

• We implemented NeuMAC on a 64 core multiprocessor using a cycle-accurate architectural simulator, Multi2sim

• To evaluate NeuMAC’s generalizability, we test on 9 different applications from diverse domains using k-fold cross validation
End-to-End Execution Time Speedups

Over CSMA
Over TDMA

1.69x
1.33x

CC
BFS
PageRnk
SSSP
Volrend
StrmClstr
Canneal
BdyTrck
Commnty
End-to-End Execution Time Speedups

Over CSMA  Over TDMA

CC  BFS  PageRnk  SSSP  Volrend  StrmClstr  Canneal  BdyTrck  Commnty

3.74x
NeuMAC can achieve up to 1.69x to 3.74x speedup over baselines.
• NeuMAC can achieve up to 1.69x to 3.74x speedup over baselines

• Similar trends over Optimal CSMA and Adaptive Switching – 1.37x to 1.56x speedups

• Scaling up to 1024 cores, NeuMAC improves throughput up to 64x and reduces latency by 3 orders of magnitude
To conclude

• NeuMAC can learn and adapt to highly dynamic and complex traffic patterns at a very fine granularity

• NeuMAC achieves 1.37x-3.74x application speedup, and can scale the gains with the number of cores

• mmWave NoCs opens up a new paradigm in chip interconnect designs, and allows programmers to overhaul the way parallel programs are written
Thank You!

Contact: sjog2@illinois.edu