Case Study: NoC Latency Estimation

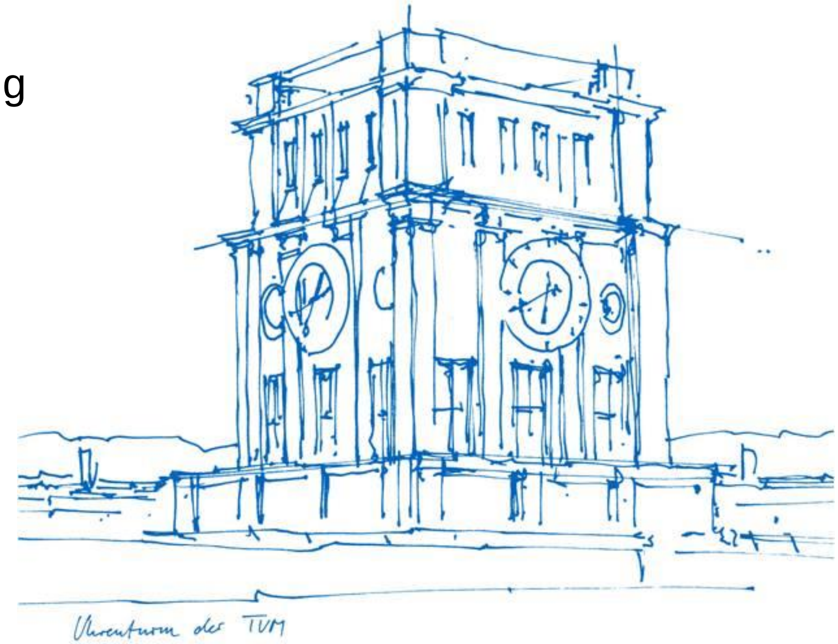
Marcel Mettler

Technische Universität München

Department of Electrical and Computer Engineering

Chair of Electronic Design Automation

25. January 2019



Outline

- Background
  - Introduction to Networks-on-Chip
  - Optimization of Network-on-Chip Topologies

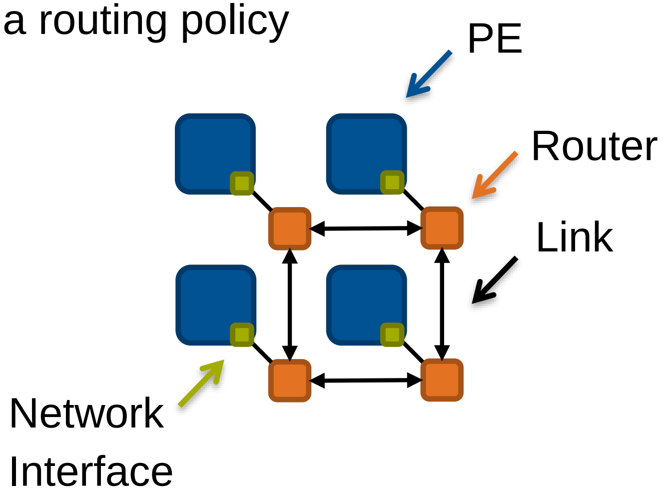
- Application of Neural Networks

- Introduction to the ML framework

Network-on-Chip

On-Chip interconnect inspired by a computer network

- **Processing Element**
  Computation unit e.g. CPU, Video Decoder
- **Network Interface**
  Translates communication protocol between network and PE
- **Router**
  Forward packets through the network according to a routing policy
- **Link**
  Point-to-point connections between rotuers



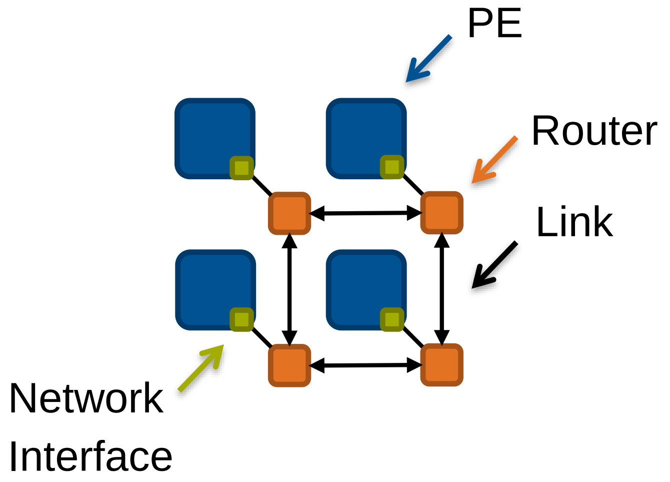
Network-on-Chip

Benefits

- Scalability
  - Overall bandwidth increases with the network size
  - Especially suitable for a high number of PEs
- Short point-to-point connections
  - Enables high clock frequencies
- Segmentation
  - Increases reusability

Drawbacks

- Latency
  - Multiple hops and multiple cycles per hop
- Chip Area



Network-on-Chip Performance Metrics

- **Aggregated Bandwidth**
  The product of the bandwidths per link bw_l and the number of links n_l
  $$BW_{agg} = n_l * bw_l$$

- **Throughput**
  Usable share of the aggregated sending bandwidth

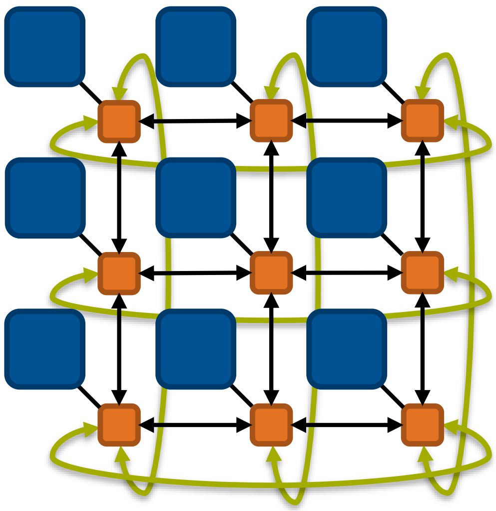
- **Average Latency**
  The average difference between the reception time t_{rx} and the transmit time t_{tx} of all packets $p \in P$
  $$l_{avg} = \frac{1}{|P|} \sum_{p \in P} t_{rx}(p) - t_{tx}(p)$$

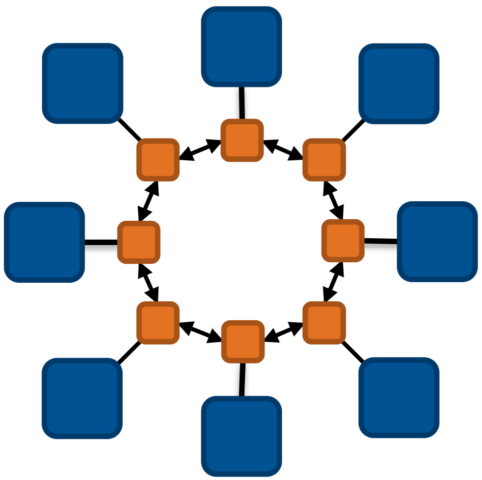
- **Chip Area**

- **Power Consumption**

[1] A. Herkersdorf , “Module 4: Interconnect,” in *Chip Multicore Processors*, 26-Jun-2017.

Regular Network-on-Chip Topologies



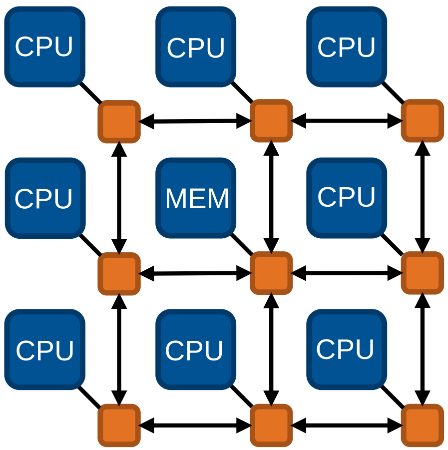


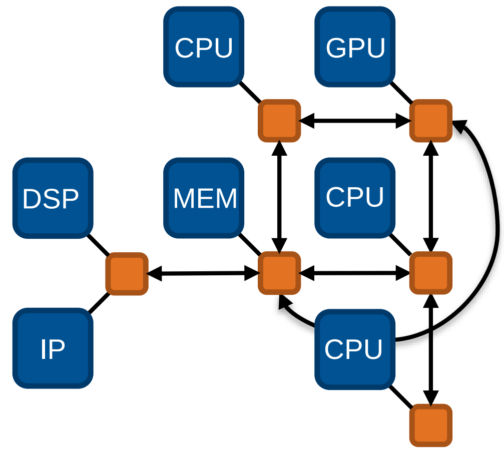
- **High Connectivity**
  - E.g. Mesh, Torus
  - High throughput and low latency
  - High power and area consumption

- **Low Connectivity**
  - E.g. Ring
  - Low power and area consumption
  - Low throughput and high latencies

[1] A. Herkersdorf , “Module 4: Interconnect,” in *Chip Multicore Processors*, 26-Jun-2017.

General-purpose vs. Application-specific NoCs





General-purpose NoCs

- Suited for general-purpose MPSoCs
- Most PEs are not fixed in functionality
  → Traffic requirements are unknown at design time
  → Regular network topologies beneficial

Application-specific NoCs

- Suited for application-specific MPSoCs
- Most PEs are fixed in their functionality
  → Traffic requirements are known at design time
  → Topology must be optimized w.r.t. application

Outline

- Background
  - Introduction to Networks-on-Chip
  - Optimization of Network-on-Chip Topologies
- Application of Neural Networks
- Introduction to the ML framework

Optimization Networks-on-Chip Topologies

Design task: Find the best-suited NoC topology for the application at hand

- The quality of a NoC topology x for an application t

$$Q(x,t) = -1 * \left[w_l l_{avg}(x,t) + w_P P(x) + w_A A(x)\right], \text{ where}$$

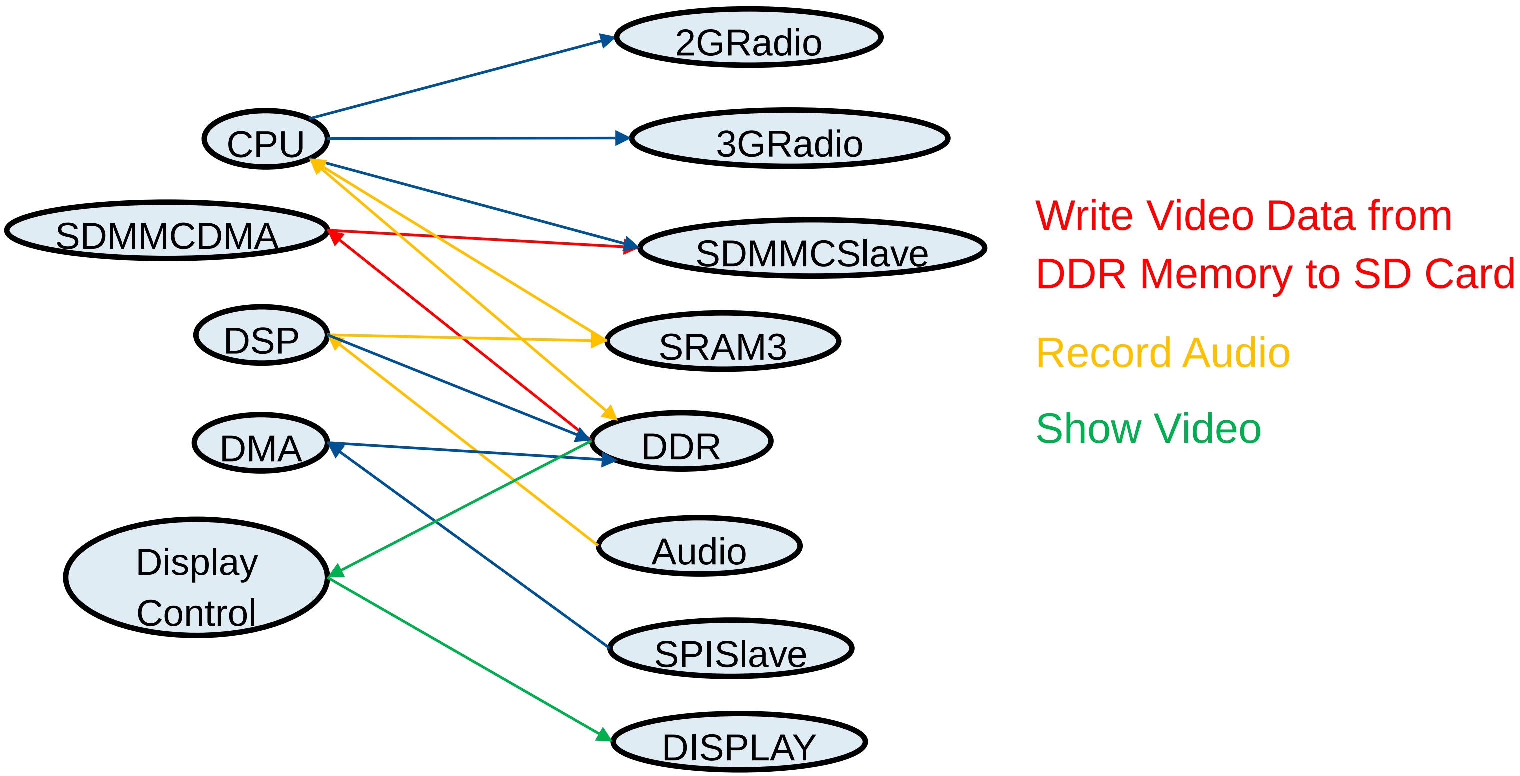
$l_{avg}(x,t) \coloneqq$ average latency
$P(x) \coloneqq$ power dissipation
$A(x) \coloneqq$ chip area
$x \coloneqq$ NoC topology
$t \coloneqq$ application

→ Evaluation toolchain required to evaluate $l_{avg}(x,t)$, $P(x)$ and $A(x)$

NoC Evaluation Toolchain

Application t

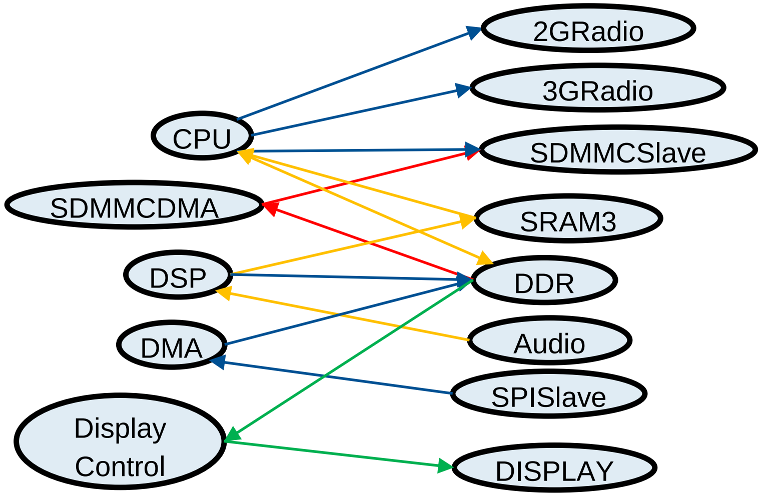
- Described by Core Communication Graph



NoC Evaluation Toolchain

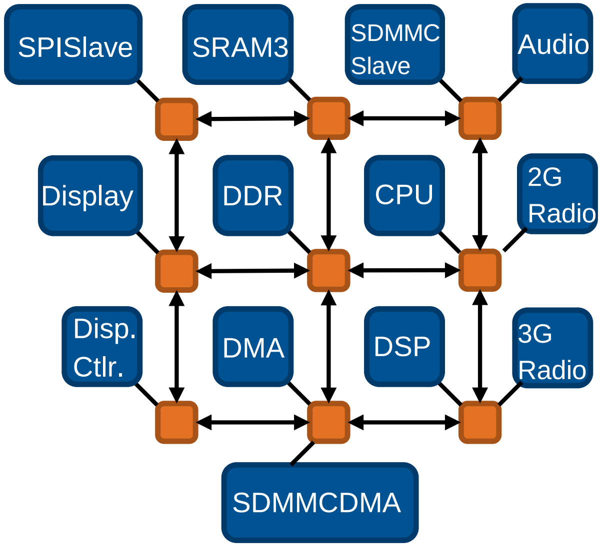
Application t

- Described by Core Communication Graph



NoC Topology x

- Described the topology graph of the network



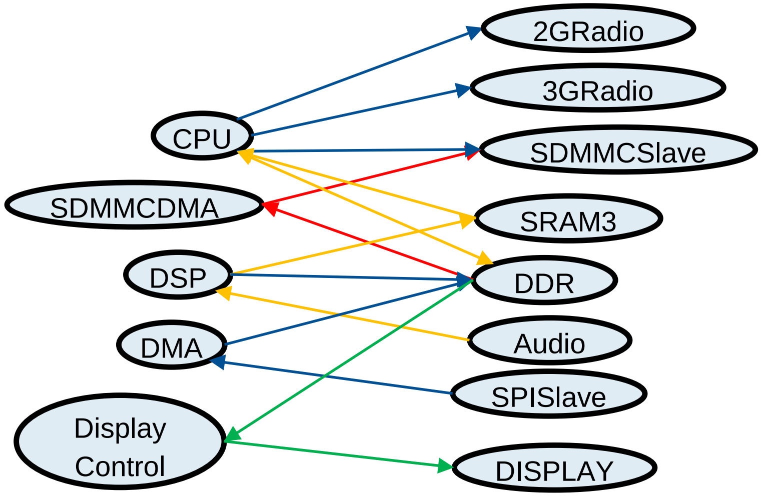
NoC Evaluation Toolchain

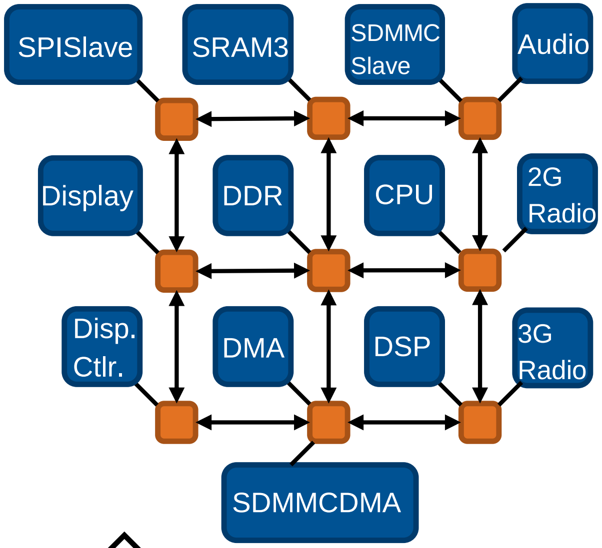
Application t

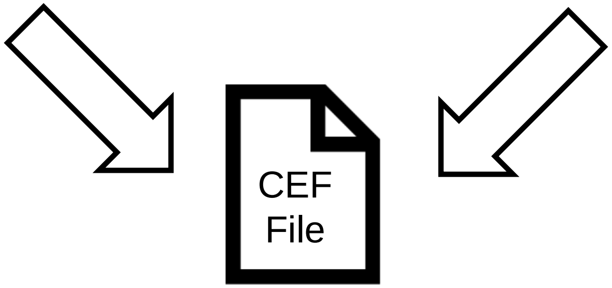
- Described by Core Communication Graph

NoC Topology x

- Described the topology graph of the network





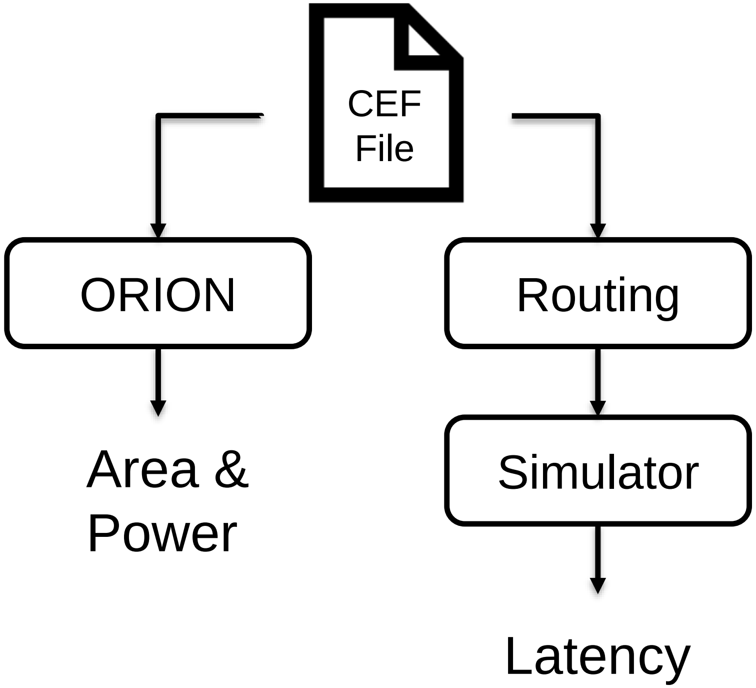


NoC Evaluation Toolchain

ORION[2]:
- Area and power model for NoCs

Routing:
- Shortest path routing algorithm

Simulator:
- Cycle-accurate SystemC simulation



[2] A. B. Kahng, B. Li, L. Peh and K. Samadi, "ORION 2.0: A Power-Area Simulator for Interconnection Networks,“ in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 191-196, Jan. 2012.

Optimization of NoC topologies

Design task: Find the best-suited NoC topology for the application at hand

Objective:

$$argmax_x \; Q(x,t)$$

Optimization methods:

- Gradient-based optimization **not applicable** as $Q(x,t)$ is not continues
- Possible Heuristics:
  - Simulated Annealing
  - Genetic Algorithms
  - Monte-Carlo Tree Search

Recap: Markov Decision Process

Process description model for **sequential decision problems** in a **fully observable environment**

- **Sequential decision problem**
  The utility of an action does not depend on a single decision but on the whole sequence of decisions

- **Fully observable environment**
  The state of a system is known at all times

- **Definition***
  S: set of possible states
  A: set of possible actions
  $t(s, a)$: state transition function returning state s' for the application of a on s
  $Q(s)$: reward function

  *Note: Definition slightly diverges from the Reinforcement Learning lecture

Monte Carlo Tree Search

- Tree search algorithm for Markov decision processes (MDP)
- Incrementally builds a search tree guided by previous explorations

Node

- State $s \in S$
- Quality of the state $Q(s)$
- Visit count of the state n_s

Edge

- Action $a \in A$

Objective

- Find action a^* in a state s which maximizes the future reward

Monte Carlo Tree Search and NoCs

Transfer NoC optimization problem into MDP

- **States $\boldsymbol{S}$**
  The complete design space of all NoC Topologies

- **Actions $\boldsymbol{A}(\boldsymbol{s})$**
  All valid modifications for a given NoC architecture s

- **Transition function $\boldsymbol{t}(\boldsymbol{s}, \boldsymbol{a})$**
  Defines how an action modifies a NoC architecture
  Formally expressed by graph rewriting

- **Reward function $\boldsymbol{Q}(\boldsymbol{s})$**
  Expresses the quality of a NoC architecture s

$$Q(s) = -1 * \left[w_l l_{avg}(s,t) + w_p P(s) + w_A A(s)\right]$$

DSE of NoC Architectures based on MCTS

Search Tree:

- Node: NoC Architecture s
- Edge: NoC modification a

Process:

Selection:

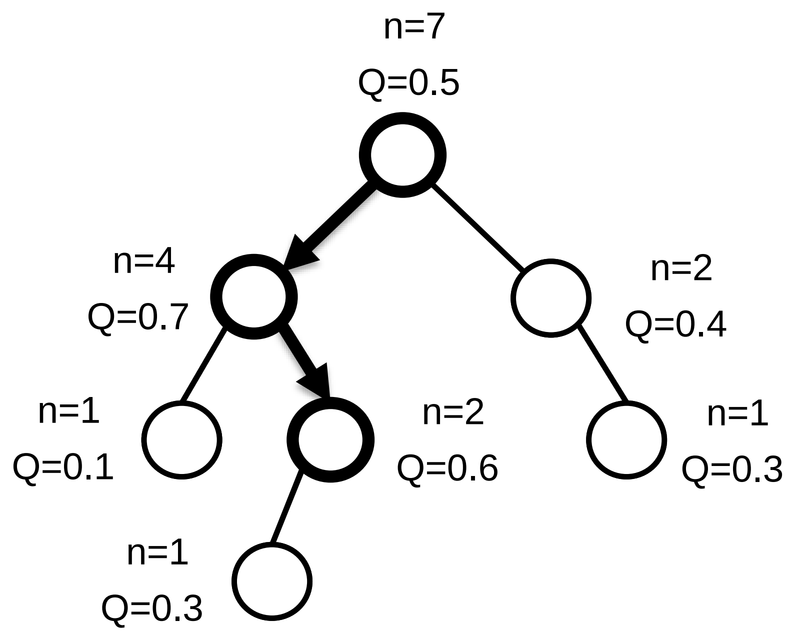
Select a node according to UCT function

$$UCT = Q_{WS}(s) + C_p \sqrt{\frac{2 \ln N(s_{root})}{N(s)}}$$

Exploitation

Exploration

Exploration parameter



DSE of NoC Architectures based on MCTS

Search Tree:

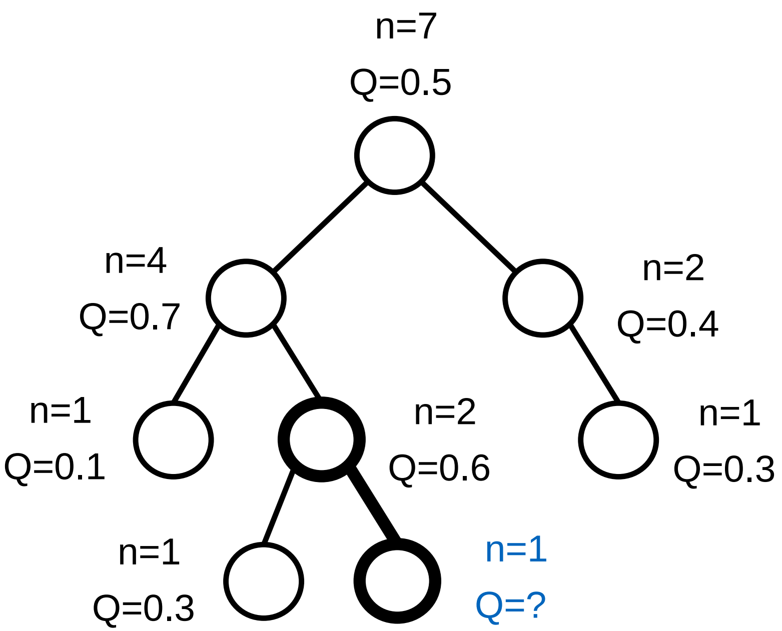
- Node: NoC Architecture s
- Edge: NoC modification a

Process:

Expansion:

Apply n modifications on the selected node:

- Add/remove link
- Switch PEs
- Shift PE
- Merge routers



DSE of NoC Architectures based on MCTS

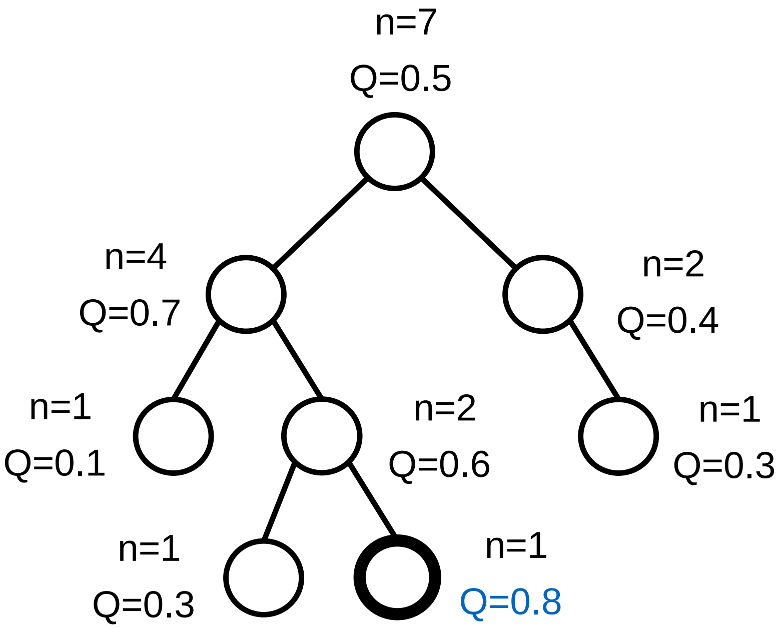
Search Tree:

- Node: NoC Architecture s
- Edge: NoC modification a

Process:

Simulation:
Evaluate the new NoC Architectures:

- Power and Area:
  ORION
- Average Latencies:
  Routing algorithm
  SystemC simulation



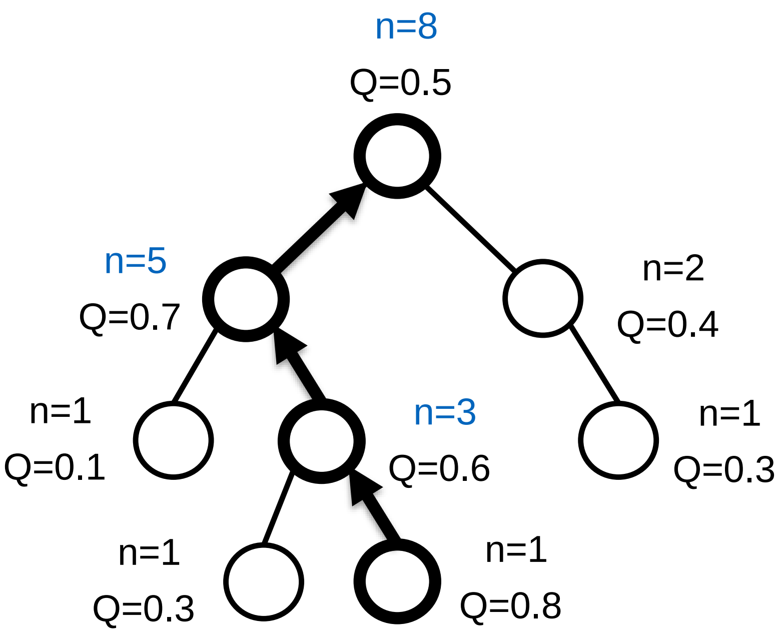
DSE of NoC Architectures based on MCTS

Search Tree:

- Node: NoC Architecture s
- Edge: NoC modification a

Process:

Backpropagation:
Update the visit count of the ancestor architectures



DSE of NoC Architectures based on MCTS

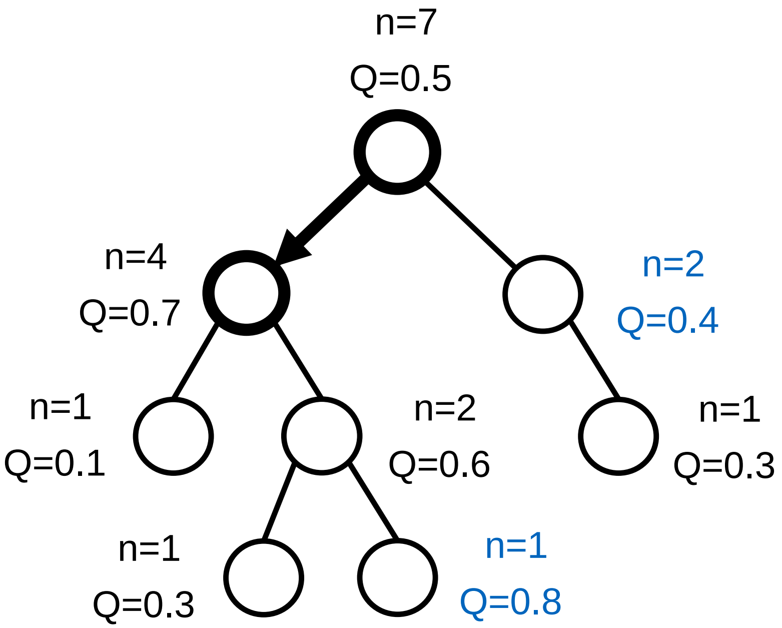
Search Tree:

- Node: NoC Architecture s
- Edge: NoC modification a

Process:

Repeat until termination criteria is fulfilled
Time budget or simulation limit

Update root node
Select child with the best successor



DSE of NoC Architectures based on MCTS

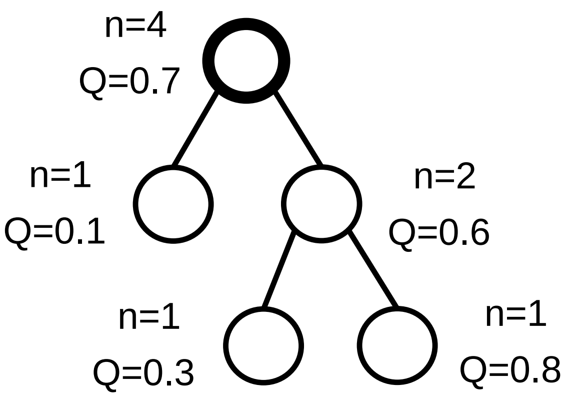
Search Tree:
- Node: NoC Architecture s
- Edge: NoC modification a

Process:

Repeat till termination criteria is fulfilled

Update root node:
Select child with the best successor

Continue with the exploration
Expansion
Simulation
Backpropagation



DSE of NoC Architectures based on MCTS

Conclusion:

✓ **No domain knowledge required**

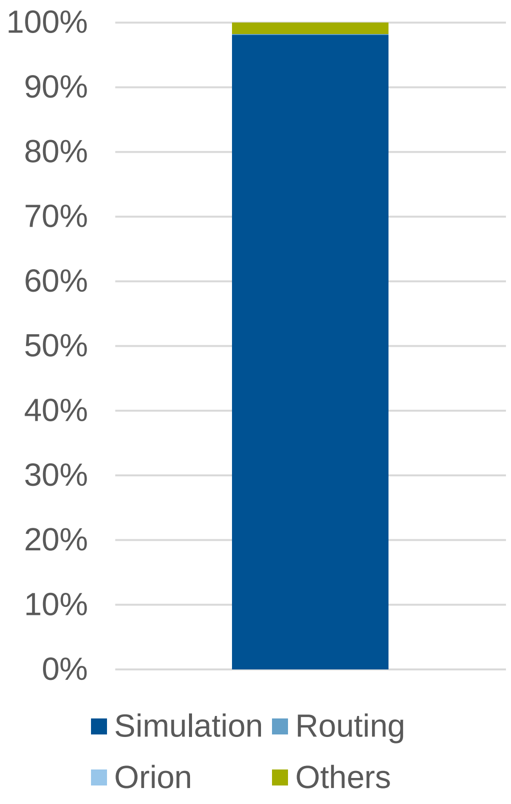
✓ **Traceability of the modifications**
Designers are able to trace the modifications and validate the result
Increases trust in the optimization tool

✗ **Low convergence speed**
Convergence to an optimized design is not given within reasonable time

DSE of NoC Architectures based on MCTS

- Analysis of the execution time identifies SystemC simulation as bottleneck

- **Objective:**
  Estimate the SystemC simulation results
  → Neural Network



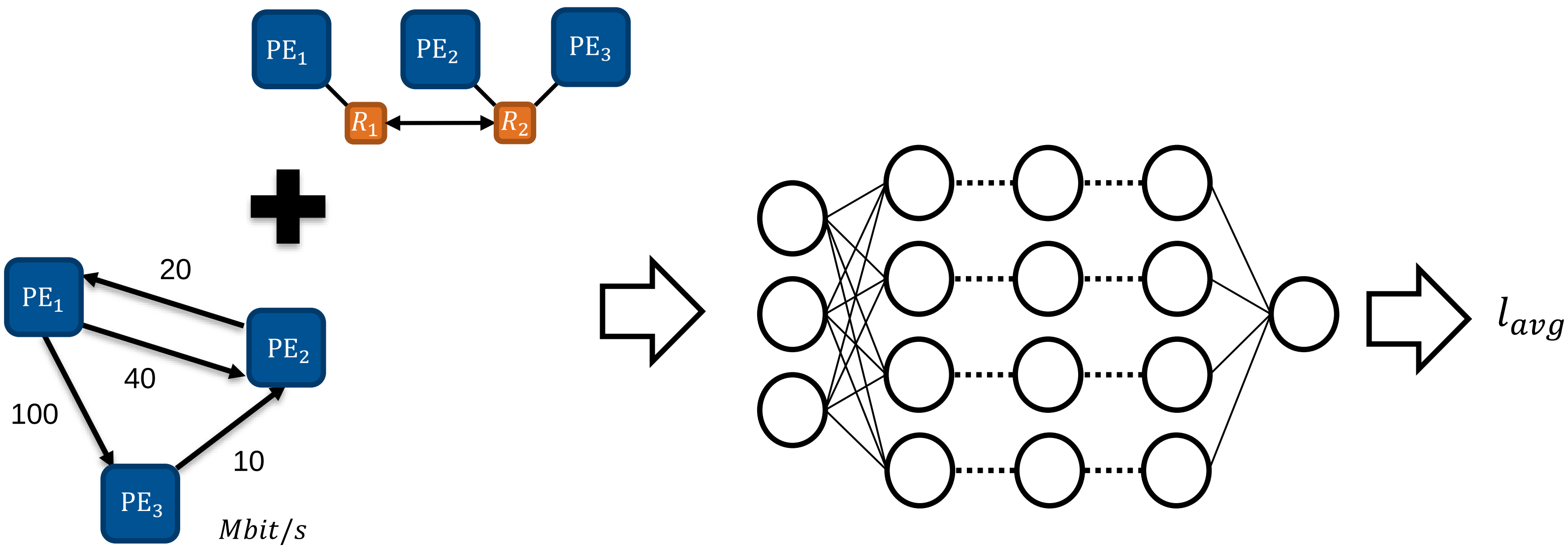
Outline

- Background

- Application of Neural Networks
    - Data Representation
    - Exploration of further neural network architectures

- Introduction to the ML framework

Data Representation Problem



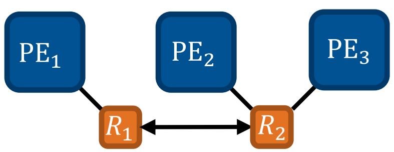
Fully-connected neural network requires a vector as input data.

Hands-on exercise:

For the chosen problem, define the input vector of the neural network.

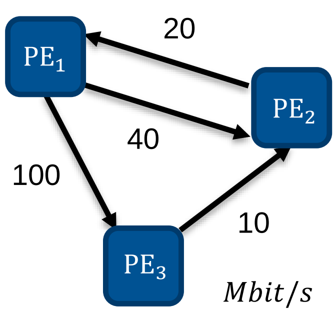
Mathematical Representation of NoCs

NoC Architecture



Adjacency Matrix

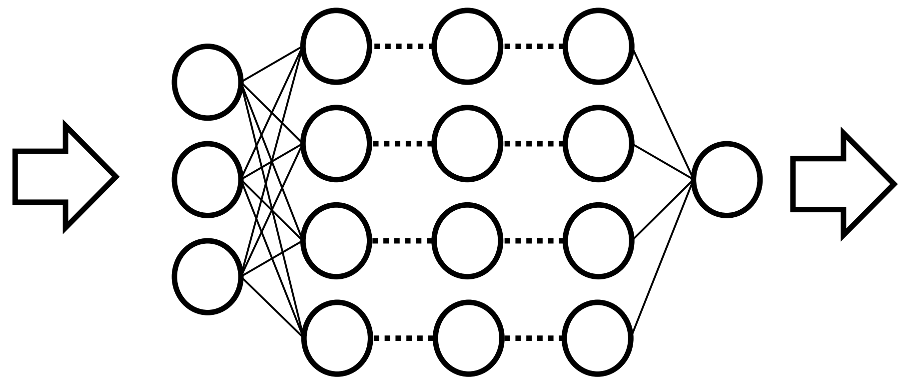
$$
\begin{array}{c|ccccc|}
 & PE_1 & PE_2 & PE_3 & R_1 & R_2 \\
\hline
PE_1 & 0 & 0 & 0 & 1 & 0 \\
PE_2 & 0 & 0 & 0 & 0 & 1 \\
PE_3 & 0 & 0 & 0 & 0 & 1 \\
R_1 & 1 & 0 & 0 & 0 & 1 \\
R_2 & 0 & 1 & 1 & 1 & 0
\end{array}
$$



$$
\begin{array}{c|ccc|}
 & PE_1 & PE_2 & PE_3 \\
\hline
PE_1 & 0 & 40 & 100 \\
PE_2 & 20 & 0 & 0 \\
PE_3 & 0 & 10 & 0
\end{array}
\quad Mbit/s
$$

Data Representation Problem

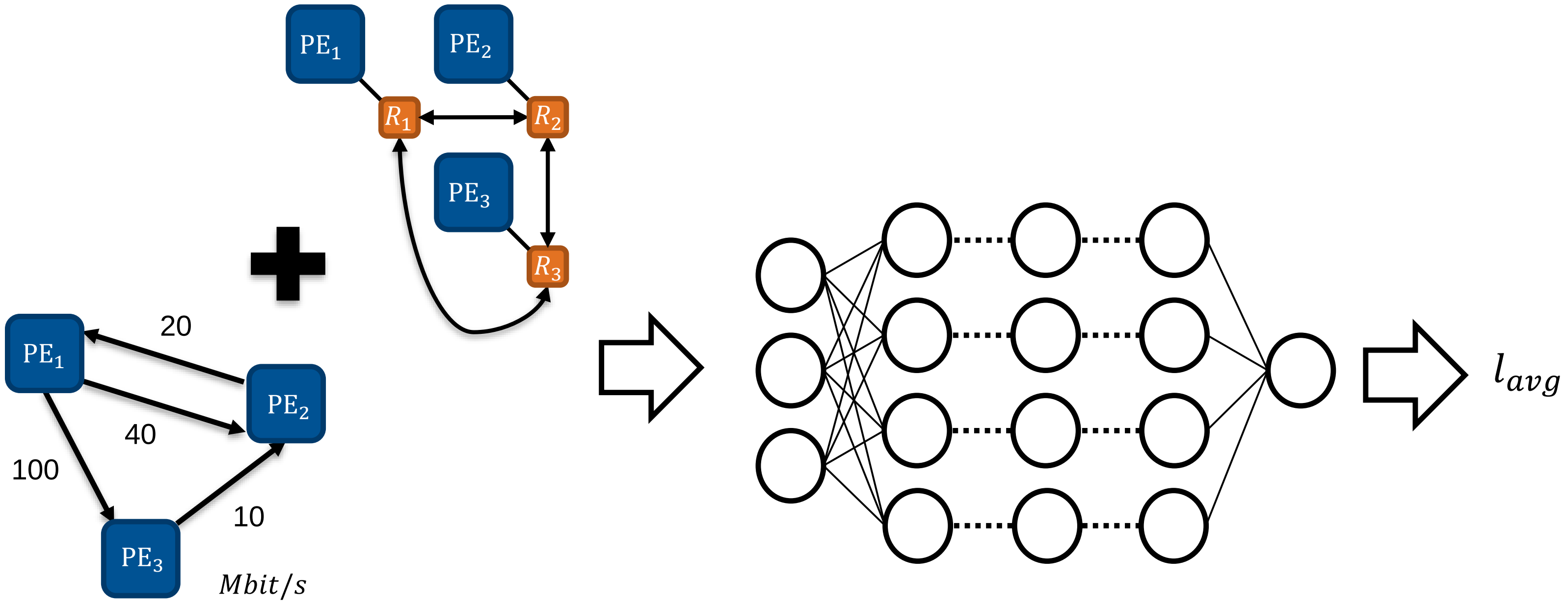
$$
\begin{array}{ll}
\text{NoC Topology} & \left\{ \begin{array}{c} PE_1 \rightarrow PE_1 \\ PE_1 \rightarrow PE_2 \\ PE_1 \rightarrow PE_3 \\ PE_1 \rightarrow R_1 \\ \vdots \\ R_2 \rightarrow R_2 \end{array} \right. \\
\text{Core Communication Graph} & \left\{ \begin{array}{c} PE_1 \rightarrow PE_1 \\ PE_1 \rightarrow PE_2 \\ PE_1 \rightarrow PE_3 \\ \vdots \\ PE_3 \rightarrow PE_3 \end{array} \right.
\end{array}
\quad
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 40 \\ 100 \\ \vdots \\ 0 \end{bmatrix}
$$



l_{avg}

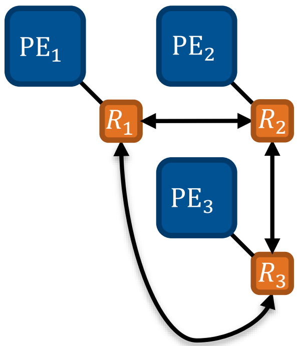
What if the network size changes?

Data Representation Problem



Data Representation Problem

NoC Architecture



Adjacency Matrix

$$\begin{array}{c c}
 & \begin{array}{cccccc} PE_1 & PE_2 & PE_3 & R_1 & R_2 & R_3 \end{array} \\
\begin{array}{c} PE_1 \\ PE_2 \\ PE_3 \\ R_1 \\ R_2 \\ R_3 \end{array} &
\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}
\end{array}$$

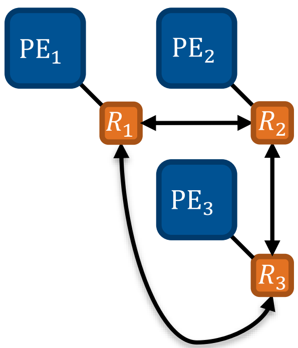
Matrix becomes larger while input vector size is fixed.

How can we address this problem?

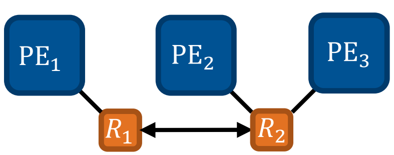
Zero Padding

NoC Architecture

Adjacency Matrix



$$
\begin{array}{c|cccccc|}
 & PE_1 & PE_2 & PE_3 & R_1 & R_2 & R_3 \\
\hline
PE_1 & 0 & 0 & 0 & 1 & 0 & 0 \\
PE_2 & 0 & 0 & 0 & 0 & 1 & 0 \\
PE_3 & 0 & 0 & 0 & 0 & 0 & 1 \\
R_1 & 1 & 0 & 0 & 0 & 1 & 1 \\
R_2 & 0 & 1 & 0 & 1 & 0 & 1 \\
R_3 & 0 & 0 & 1 & 1 & 1 & 0
\end{array}
$$



$$
\begin{array}{c|cccccc|}
 & PE_1 & PE_2 & PE_3 & R_1 & R_2 & R_3 \\
\hline
PE_1 & 0 & 0 & 0 & 1 & 0 & 0 \\
PE_2 & 0 & 0 & 0 & 0 & 1 & 0 \\
PE_3 & 0 & 0 & 0 & 0 & 1 & 0 \\
R_1 & 1 & 0 & 0 & 0 & 1 & 0 \\
R_2 & 0 & 1 & 1 & 1 & 0 & 0 \\
R_3 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

- Chose a matrix size which fits the maximal supported number of routers and Pes
- Set the rows and columns of non-present routers/PEs to 0

  → matrices might become very sparse for large

Conclusion

- × The number of features highly depends on the NoC size
  - → Extensive zero padding would be required
- × Large NoCs require very big input vector ($> 10^3$)
  - → High number of trainable parameters ($> 10^6$)
- × Sparse input vectors are difficult to train
- × Spatial invariance of the features is not considered

- ➢ Not suited for the problem at hand

Outline

- Background
- Application of Neural Networks
  - Data Representation
  - Exploration of further neural network architectures
- Introduction to the ML framework

Neural Network Architecture Exploration

Convolutional Neural Network

- ✓ Processes patterns in the input vector

- × Designed for data in the Euclidean domain (i.e. not for graphs)
- × The number of features highly depends on the NoC size
  → Extensive zero padding would be required
- × Locality between neighboring routers vanishes after transformation into matrices

- ➢ Not suited for the problem at hand

Neural Network Architecture Exploration

Fully-Convolutional Neural Network

- ✓ Processes patterns in the input vector
- ✓ Processes arbitrary input vector sizes

- × Designed for data in the Euclidean domain (i.e. not for graphs)
- × Locality between neighboring routers vanishes after transformation into matrices

- ➢ Not suited for the problem at hand

Neural Network Architecture Exploration

Recurrent Neural Networks

- ✗ Designed for sequences of data
- ➢ Not suited for the problem at hand

Neural Network Architecture Exploration

Geometric Deep Learning[3]

- summarizes a set of neural network architectures for manifolds and graphs
- Still open field of research

✓ Designed for non-Euclidean domains i.e. graphs

× Graphs of arbitrary sizes are not supported

➢ Not suited for the problem at hand

[3] Federico Monti, “Geometric Deep Learning,” *Geometric Deep Learning*.
[Online]. Available: http://geometricdeeplearning.com/. [Accessed: 18-Jan-2019].

Outline

- Background
- Application of Neural Networks
  - Data Representation
  - Neural Network Architecture Exploration
  - Rewrite Problem
- Introduction to the ML framework

Rewrite Optimization Problem

Objective:

Estimate the average latency in a NoC

$$l_{avg} = \frac{1}{|P|} \sum_{p \in P} T_{rx}(p) - T_{tx}(p)$$

Rewritten Objective:

Estimate the average latency of each flow $f \in F$ individually

$$l_{avg} = \frac{1}{|F|} \sum_{F} \frac{bw_f}{BW_{NoC}} l_f$$

$$BW_{NoC} = \sum_{F} bw_f$$

Flows with higher BW send more packets and thus must be given a stronger weight

Design of the Recurrent Neural Network

Output:

The average latency of a flow f^*

Input:

Sequence of vectors describing the contentions along the routing path of flow f^*

$$\mathrm{S}_{\mathrm{f}^*} = \left(\underline{c_0}, \underline{c_1}, \dots, \underline{c_n}, \dots, \underline{c_{N-1}}\right);\ N = \text{number of hops}$$

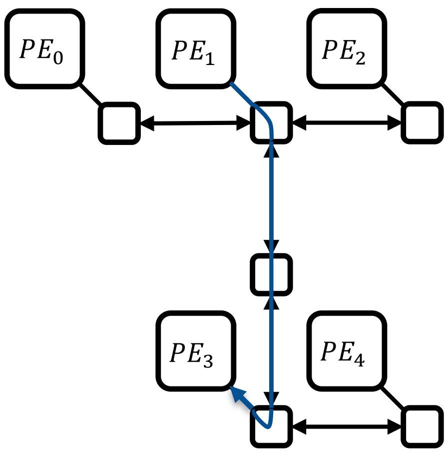
Each vector $\underline{c_n}$ collects the link utilization of all conflicting flows $F_{c_{f^*}}$ at the n^{th} hop of f^*:

$$\underline{c_n} = \left(u_{f^*}, u_{f_0}, u_{f_1}, \dots, u_{f_M}, 0, \dots, 0\right)^T;\ f_m \in F_{c_{f^*}} \text{and } \left|\underline{c_n}\right| = 10$$

$$u_f := \frac{BW_f}{BW_{max}}$$

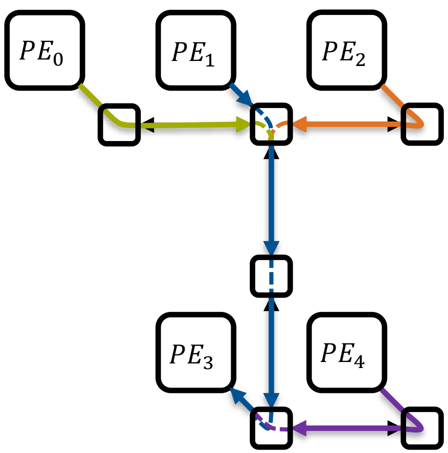
upper bound of conflicting flows

Design of the Recurrent Neural Network

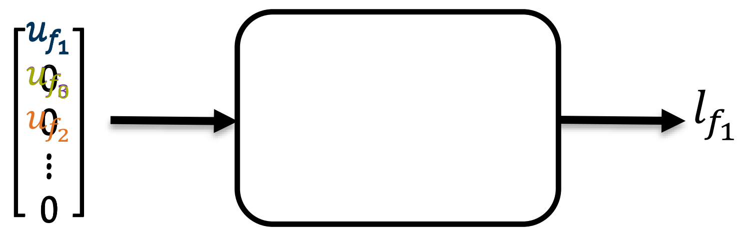


$$
\begin{array}{c}
\\
PE_0 \\ PE_1 \\ PE_2 \\ PE_3 \\ PE_4
\end{array}
\begin{array}{c}
\begin{array}{ccccc} PE_0 & PE_1 & PE_2 & PE_3 & PE_4 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & bw_{f_0} & 0 \\
0 & 0 & 0 & bw_{f_1} & 0 \\
0 & 0 & 0 & bw_{f_2} & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & bw_{f_3} & 0
\end{bmatrix}
\end{array}
$$

Design of the Recurrent Neural Network



$$
\begin{array}{c} \\ PE_0 \\ PE_1 \\ PE_2 \\ PE_3 \\ PE_4 \end{array}
\begin{array}{c}
\begin{array}{ccccc} PE_0 & PE_1 & PE_2 & PE_3 & PE_4 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & {\color{olive} bw_{f_0}} & 0 \\
0 & 0 & 0 & {\color{blue} bw_{f_1}} & 0 \\
0 & 0 & 0 & {\color{orange} bw_{f_2}} & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & {\color{purple} bw_{f_3}} & 0
\end{bmatrix}
\end{array}
$$



Recurrent Neural Network

Outline

- Background
- Application of Neural Networks
- Introduction to the Machine Learning Framework

Python IDE - Pycharm

Start Pycharm

cd /usr/local/labs/ML/

./bin/ml pycharm &

Open Project

File → Open…

Change directory to */usr/local/labs/ML/ab12cde/ml_eda*

Ok

Run or Debug Project

Run → Run…/Debug… →TaskX

Open a Terminal in Pycharm

View → Tool Windows → Terminal or Alt+F12

Open the Python Console in Pycharm

View → Tool Windows → Python Console

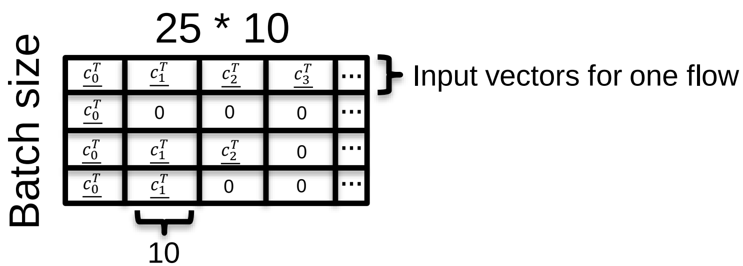
Project Structure

ml_eda

- data
  - features.npy
  - labels.npy

Training Data

- Features
  - 2d-array (batch size x concatenated input vectors)
  - Max number of contentions: 10
  - Max number of hops: 25



- Labels
  - 1d-array (batch size)
  - Average latency in cycles (from SystemC)

Note: First half of the data is for lowly utilized NoCs while the other half is for highly utilized NoCs!

Project Structure

ml_eda

- data
  - features.npy
  - labels.npy
- logs

Log-directory

- Output directory for all log-files generated during the training process including
  - Hyperparameters
  - Neural Network weights
  - Event files for Tensorboard

Project Structure

ml_eda

- data
  - features.npy
  - labels.npy
- logs
- models
  - genericRNN.py
  - simpleLSTM.py

Neural Network Models

- SimpleLSTM.py
  - Simple RNN model using LSTM cells
  - Used in Task 1

- GenericRNN.py
  - Parameterizable RNN
  - Used for the hyperparameter optimization in Task 2

Project Structure

ml_eda

- data
  - features.npy
  - labels.npy
- logs
- models
  - genericRNN.py
  - simpleLSTM.py
- venv

Virtual Python Environment

- Isolated python environment
- Stores ML libraries (e.g. keras and tensorflow)
- Do not modify this directory!

Project Structure

ml_eda

- data
  - features.npy
  - labels.npy
- logs
- models
  - genericRNN.py
  - simpleLSTM.py
- venv
- HyperparameterOptimization.py

Hyperparameter Optimization Class

- Wrepper Class for the scikit-optimize library
- Utilized in Task 2

Project Structure

ml_eda

- data
  - features.npy
  - labels.npy
- logs
- models
  - genericRNN.py
  - simpleLSTM.py
- venv
- HyperparameterOptimization.py
- NeuralNetwork.py

Neural Network Base Class

- Abstract base class for all models in *models*
- Main Methods:
  - *fitness(cls, hyperparameters)*
    Trains a neural network for 5 epochs and returns the mean squared validation error

  - *split_data_set(cls, features, labels)*
    Splits data set into training, validation and test data

Project Structure

ml_eda

- data
  - features.npy
  - labels.npy
- logs
- models
  - genericRNN.py
  - simpleLSTM.py
- venv
- HyperparameterOptimization.py
- NeuralNetwork.py
- Task1.py
- Task2.py

Task1

- Main class for the first task

Task2

- Main class for the second task

Recurrent Neural Networks in Keras

LSTM & GRU Class

- Implementation not suited for GPUs
- Important parameters
  - Units
  - Return_sequence

Masking Class

- Supports masking for input data of variable number of time steps
- Important parameters:
  - Mask_value
  - Input_shape

CuDNNLSTM & CuDNNGRU Class

- GPU only implementation of LSTM and GRU
- Do not support masking layer

Further Information: https://keras.io/layers/recurrent/

Task 1

a) Data pre-processing
   - Load data set from the files „data/features.npy“ and „data/labels.npy“
   - Separates the data set into 70% training, 15% validation and 15% test set using the method split_data_set in NeuralNetwork.py

b) Train the RNN
   - Implement a LSTM network in the models/SimpleRNN.py class
   - Tune the hyperparameters to obtain a mean absolute percentage error below 5%

Task 2

a) Hyperparameter Optimization
   - Define a set of hyperparameters for your RNN.
   - Implement a parameterizable RNN in GenericRNN.py.

b) Random Search
   - Run a random search optimization for 15 Iterations.
   - Plot the results using Tensorboard and find the best network.

c) Bayesian Optimization
   - Run a Bayesian optimization for 15 iterations (modify log directory).
   - Plot the reuslts using Tensorboard and find the best network.