Algorithms and Hardware for AI
Accelerated Discoveries

Deming Chen, Song Han, Philip Harris, Scott Hauck, Pan Li, Dylan Rankin
Roadmap

Overview (16 min)

Pan: Graph NNs (8 min)

Dylan: HLS4ML (8 min)

Deming: PyLog, ScaleHLS (8 min)

Song: PVCNN, SPVNAS, PointAcc (8 min)
Recent boost of AI techniques

Massive Data

AI Algorithms

Computing hardware

Models

Optimization

[The Exponential Growth of Data]
For scientific applications...

- Large-scale scientific data introduce **new challenges**.
For scientific applications...

- Large-scale scientific data introduce **new challenges**.
- Our goal: Prototype new **algorithms and hardware** for domain scientists to deal with such new challenges.
AI Design Modules

Algorithm design
- Training
- Generalization

Hardware design
- Compiler
- Architecture

Co-design
The key question in training: How to learn a model $f_\theta$ that can well and efficiently approximate the target function $f$?
Capacity

The key question in **training**: How to learn a model that can **well and efficiently** approximate the target function $f$?

- Can $f$ be well approximated by $f_\theta$ with a given number of parameters $\theta$?
- The model $f_\theta$ with a small number of parameters $\theta$ indicates good generalization, low memory cost and maybe low latency as well.
Neural Networks have extremely large capacity!

- Shallow NNs approximate continuous functions [Cybenko (1989), Hornik et al (1989)].
- Deep NNs better approximate highly-differentiable functions [Yarotsky (2017)].

\[ x \xrightarrow{Wx + b} \sigma(Wx + b) \xrightarrow{W'\sigma(Wx + b)} y \]

\( \sigma(\cdot) \) is a entry-wise nonlinear function, e.g. \( \sigma(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \)
Neural Networks have extremely large capacity!

- Shallow NNs approximate continuous functions [Cybenko (1989), Hornik et al (1989)].
- Deep NNs better approximate highly-differentiable functions [Yarotsky (2017)].
- Graph (point cloud) NNs with finite neurons have not been well studied yet...

\[
x \xrightarrow{} Wx + b \xrightarrow{} \sigma(Wx + b) \xrightarrow{} W'\sigma(Wx + b)
\]

\(\sigma(\cdot)\) is an entry-wise nonlinear function, e.g. \(\sigma(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}\)
Optimization

The key question in training:
How to learn a model that can **well and efficiently** fit the data we have observed?

- Can we fast compute the parameter $\theta$ such that $f_\theta$ well approximate $f$?

Backpropagation

Rumelhart et al. [1988]
The key question in generalization: How to guarantee the model capture the law that can apply to the data that has not been observed?
Generalization

The key question in generalization: How to guarantee the model capture the law that can apply to the data that has not been observed?

- Data-driven neural network models: Convolutional NNs (CNNs), Graph NNs...
- For A3D3, inject scientific data insights into the model design

Translation Invariance
The key question in compiler design: How to automatically and efficiently map an algorithm $f_\theta$ onto a hardware platform?
Complier --- High-Level Synthesis (HLS)

Levels of Abstraction

- System level
  - Design spec. in high-level languages
    - SW/HW Co-design
  - High-level Synthesis
    - C, C++, SystemC
    - behavioral synthesis
- Behavior level
  - RT level
    - (VHDL, Verilog)
    - controller
    - datapath
- Gate level
  - (netlist)
  - Logic Synthesis
  - Place & Route

- Chip

Levels of Abstraction

- Gate level
  - (netlist)
- Logic Synthesis
- Place & Route
- Chips

Design hardware using C or C++ compared to RTL
- 10X code reduction
- 1000X simulation time reduction

[Source: NEC]
Complier --- High-Level Synthesis (HLS)

Levels of Abstraction

- **System level**: Design spec. in high-level languages
  - SW/HW Co-design
- **Behavior level**: C, C++, SystemC
- **RT level**: (VHDL, Verilog)
- **Gate level**: (netlist)
- **Place & Route**: Chips

Where AI models are designed

Gap...

Where current HLS tools support
Complier --- High-Level Synthesis (HLS)

A3D3 members are now developing the end-to-end compiler support from the python level to the RT level.
The key question in architecture design:
How to design, allocate and arrange computation and communication components for the algorithm $f_\theta$?
GPU outperforms CPU as for the many more ALUs to support parallel computing.

[Cuda C-programming guide 2014]
FPGA allows dedicated data flow control and further enhances parallelism.

Throughput = Batch-Size (BS) / Latency

[Alibaba Clouder]
The key question in co-design: How to jointly design an AI algorithm with its hardware implementation?

Algorithm design
- Training
- Generalization

Hardware design
- Compiler
- Architecture

Co-design

[Hao, et al., 2019]
Co-design

• **Top-down**: evaluate neural networks through architecture template mapping
  --- traditional hardware design for AI models

• **Bottom-up**: architecture template guided neural network model search
  --- Co-design considers both directions

[Hao, et al., 2019]
One example of co-design via differentiable neural architecture search:

- **One direction**: Optimize the model architecture $A$ while keep its hardware implement $I$ fixed.

\[
\min_A \ Loss_{pred}(A) \ast Loss_{hw}(I)
\]

- **Bi-direction**: Optimize both the model architecture $A$ and its hardware implement $I$

\[
\min_{A,I} \ Loss_{pred}(A) \ast Loss_{hw}(I) + \text{Penalty (I)}
\]

The penalty on the implementation according to its hardware resource consumption

[Hao, et al., 2019] [Cai, et al., 2019]
AI Design Modules

$f$: Target func.
$f_0$: Learnable func.

Algorithm design

Training

Generalization

Hardware design

Compiler

Architecture

Co-design

Bottom-up

Bi-directional Co-Design

Top-down

DNN Models

Hardware Accelerators
On Edge Devices

Under-fitting
(too simple to explain the variance)

Appropriate-fitting

Over-fitting
(force-fitting—too good to be true)