EDA for AI Chip Designs

Bei Yu
Chinese University of Hong Kong
byu@cse.cuhk.edu.hk

July 30, 2023
Convolutional Neural Network (CNN)

Autonomous drive

Image recognition

- bouquet of red flowers
- tablet
- bottle of water
- glass of water with ice and lemon
- cup of coffee
- dining table with breakfast items
- plate of fruit
- banana slices
- fork
- a person sitting at a table
Renaissance of Deep Learning (2006 – )

• A fast learning algorithm for deep belief nets. [Hinton et.al 1996]
• Data + Computing + Industry Competition
• NVidia’s GPU, Google Brain (16,000 CPUs)
• Speech: Microsoft [2010], Google [2011], IBM
• Image: AlexNet, 8 layers [Krizhevsky et.al 2012] (26.2% -> 15.3%)
Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)

11x11 conv, 96, /4, pool/2
5x5 conv, 256, pool/2
3x3 conv, 384
3x3 conv, 384
3x3 conv, 256, pool/2
fc, 4096
fc, 4096
fc, 1000
Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)

VGG, 19 layers (ILSVRC 2014)

GoogleNet, 22 layers (ILSVRC 2014)
Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)

VGG, 19 layers (ILSVRC 2014)

ResNet, 152 layers (ILSVRC 2015)
Some Recent Classification Architectures

- **AlexNet** (Krizhevsky, Sutskever, and E. Hinton 2012) 233MB
- **Network in Network** (Lin, Chen, and Yan 2013) 29MB
- **VGG** (Simonyan and Zisserman 2015) 549MB
- **GoogleNet** (Szegedy, Liu, et al. 2015) 51MB
- **ResNet** (He et al. 2016) 215MB
- **Inception-ResNet** (Szegedy, Vanhoucke, et al. 2016)
- **DenseNet** (Huang et al. 2017)
- **Xception** (Chollet 2017)
- **MobileNetV2** (Sandler et al. 2018)
- **ShuffleNet** (Zhang et al. 2018)
Some Recent Classification Architectures

- **AlexNet** (Krizhevsky, Sutskever, and E. Hinton 2012) 233MB
- **Network in Network** (Lin, Chen, and Yan 2013) 29MB
- **VGG** (Simonyan and Zisserman 2015) 549MB
- **GoogleNet** (Szegedy, Liu, et al. 2015) 51MB
- **ResNet** (He et al. 2016) 215MB
- **Inception-ResNet** (Szegedy, Vanhoucke, et al. 2016) 23MB
- **DenseNet** (Huang et al. 2017) 80MB
- **Xception** (Chollet 2017) 22MB
- **MobileNetV2** (Sandler et al. 2018) 14MB
- **ShuffleNet** (Zhang et al. 2018) 22MB
Why AlexNet is large in size, but small in operations?

- Special FC layers
- Special Conv layers
- More channels
- Some redundant operators
Convolutional Neural Network (CNN)

Convolutional layers account for over 90% computation

Flexibility vs. Efficiency

- **CPU** (Raspberry Pi3)
- **GPU** (Jetson TX2)
- **FPGA** (UltraZed)
- **ASIC** (Movidius)

[Diagram showing the comparison between CPU, GPU, FPGA, and ASIC in terms of flexibility and power/efficiency]
Convolution layer is one of the most expensive layers

- Computation pattern
- Emerging challenges

More and more end-point devices with limited memory

- Cameras
- Smartphone
- Autonomous driving
1st Challenge: Model Size

Hard to distribute large models through over-the-air update

AlphaGo: 1920 CPUs and 280 GPUs, $3000 electric bill per game

on mobile: drains battery
on data-center: increases TCO

---

## Application Category

<table>
<thead>
<tr>
<th>Both</th>
<th>Datacenter</th>
<th>Edge</th>
</tr>
</thead>
</table>
DNN Deployment Flow
Deployment Flow: A Naive Approach

DNN model

PyTorch
ONNX
TensorFlow
Caffe

DNN model

(?)

AI Chip
Deployment Flow: A Naive Approach
Deployment Flow: A Naive Approach

[Diagram showing the deployment flow with DNN model, Compiler, Hardware Description Language, and AI Chip]

- DNN model
- Compiler
- Hardware Description Language
- AI Chip
Deployment Flow: A Naive Approach

Question:
Why NOT using such deployment approach?
Deployment Flow: AI Chip Generation

Design Specifications

Hardware Description Language

EDA Flow

AI Chip

PyTorch

TensorFlow

Caffe

module test
  input in[3];
  ...
end module

RTL Design

Logic Synthesis

Place & Route

Testing & Verification

Packaging

Deployment Flow: AI Chip Generation
Deployment Flow: DNN Model Compilation

**Design Specifications**

**Hardware Description**

**Language**

**AI Chip**

**EDA Flow**

**Compiler**

**Instructions**

**DNN model**

**Manual Template**

```python
for i.0 in range( ):
    for j.0 in range( ):
        for k.0 in range( ):
            for i.1 in range( ):
                for j.1 in range( ):
                    C[...] += A[...] * B[...]
            for i.2 in range( ):
                for j.2 in range( ):
                    D[...] = max(C[...], 0.0)
```

**Parameter Search**
## Multi-Level Intermediate Representation (MLIR)\(^4\)

### LLVM
- **Pro**: Target-independent representation for optimization
- **Con**: Operating low-level operations is tricky

### MLIR
- A tool for multi-level IR design (MLIR dialects)
- Enable different levels of abstraction

\(^4\) [https://mlir.llvm.org/](https://mlir.llvm.org/)
Torch-MLIR
The Torch-MLIR project aims to provide first class compiler support from the PyTorch ecosystem to the MLIR ecosystem.

Other MLIR Based DL Compiler:

1. OpenXLA
2. StableHLO
3. Triton
4. OneFlow

The overall architecture of Torch-MLIR
TPU-MLIR is an open-source machine-learning compiler based on MLIR for TPU.

**Top Dialect:**
- graph optimization
- quantization and inference
- ...

**TPU Dialect:**
- weight reordering
- operator slicing
- address assignment
- ...

---

Why MLIR?

- Stable LLVM community
- Easy to implement
- Efficient optimization in multi-level abstract

☆: You can simply combine other MLIR projects by dialect conversion.
- Circuit IR Compilers and Tools (CIRCT)
- apply MLIR and the LLVM development methodology
- to the domain of hardware design tools.
Calyx

Calyx is a compiler infrastructure for languages that target hardware accelerators.

Firrtl

Firrtl is an intermediate representation (IR) for digital circuits designed as a platform for writing circuit-level transformations.

Calyx and Firrtl has been integrated with the LLVM CIRCT infrastructure and is available as a dialect within it.

---

8 https://github.com/chipsalliance/firrtl
9 https://circt.llvm.org/
Update on Chisel\textsuperscript{10}

**Chisel v3.6.0:**

- The primary change in Chisel v3.6.0 is the transition from the Scala FIRRTL Compiler to the new MLIR FIRRTL Compiler (CIRCT/FIRTOOL).

- "Meaningless intermediate variable" problem is solved.

**Chisel v5.0.0:**

- Scala FIRRTL Compiler (SFC) is deprecated.

Today We Focus on EAD Part

- Design Specifications
- Hardware Description
- Language
- AI Chip
- EDA Flow

module test
  input in[3];
  ...
end module

- RTL Design
- Logic Synthesis
- Place & Route
- Testing & Verification
- Packaging
Outline

2. Arithmetic Unit Synthesis

3. Datapath Driven Placement

4. Wafer-Scale Floorplan
Arithmetic Unit Synthesis
Design Specifications

Hardware Description

Language

AI Chip

EDA Flow

RTL Design

Logic Synthesis

Place & Route

Testing & Verification

Packaging

module test
  input in[3];
  ...
end module
Logic synthesis v.s. Physical synthesis

Constraints mapping between two synthesis stages is difficult.
Question:

Why We Need to Optimize Arithmetic Circuits (Adder & Multiplier)?
<table>
<thead>
<tr>
<th>mfo</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>node size s</td>
<td>180</td>
<td>190</td>
<td>200</td>
<td>210</td>
<td>220</td>
<td>230</td>
<td>240</td>
</tr>
</tbody>
</table>

**Front-End Team Perspective:**

**Back-End Team Perspective:**

- Run design tools with all solutions is time-consuming. => **Enumeration is not feasible!**
- For 3K solutions, running time is $3000 \times 5 = 15K$ mins.
- What we care: **Pareto Frontier Curve**
Pareto Frontier

- All the points are not dominated by any other point.
- Evaluation: Hyper-volume.
  - Size of the region bounded by the Pareto frontier and reference point.
  - Each dimension of the reference point is the maximum value on that dimension.
General flow;

- Use a joint output Power-Delay function (PD) as the regression output rather than using any single output;
- Select different features in different applications.

$\alpha$-sweep learning

---

4S. Roy, Y. Ma, J. Miao and B. Yu, "A learning bridge from architectural synthesis to physical design for exploring power efficient high-performance adders", ISLPED’17.
Can we do more?

Directions

• Can we improve the quality of the Pareto-Frontier?

• We use 3K samples to cover the solution space. Can we use less labelled data for training?

Solution: Active Learning

• Select representative samples to learn the property of the Pareto-Frontier;

• Fewer samples are needed, and the cost for labeling data is reduced.
Regression

- Gaussian process model;
- A prediction consists of a mean and a variance;
- Off-the-shelf library for implementation.

\[
f(x)\]

\[
\text{Prediction}
\]

\[
\text{95% confidence interval}
\]

---

Uncertainty

- Given the prediction \((m, \sigma)\), a hyper-rectangle is defined as
  \[
  HR(x) = \{y : m_i(x) - \beta \sigma_i(x) \leq y_i \leq m_i(x) + \beta \sigma_i(x)\}
  \]

- The uncertainty region is defined as:
  \[
  R_{t+1}(x) = R_t(x) \cap HR(x)
  \]
Classification

\[
x \in \begin{cases} 
P, & \text{if } \max(R_t(x)) \leq \min(R_t(x')) + \delta, \\
N, & \text{if } \max(R_t(x')) \leq \min(R_t(x)) + \delta, \\
U, & \text{otherwise.}
\end{cases}
\]
Sampling

• Pick the one with largest uncertainty among the Pareto-optimal designs and unclassified designs.

\[ w_t(x) = \max_{y, y' \in R_t(x)} ||y - y'||_2. \]
Pareto-Frontier Results
Pareto-Frontier Results

![Area vs Critical Delay](image1)

![Power vs Critical Delay](image2)
A neural process is exploited to reduce computational complexity.

- A variational graph autoencoder is built to extract features from prefix adder structures automatically;
- A neural process is exploited to reduce computational complexity.

---

Our solution: Overview (I) [TCAD’21]
Our solution: Overview (II) [TCAD’21]

Graph Construction

Node Feature Matrix:

\[
\begin{pmatrix}
1 & 0 & 0.1 & 0.4 \\
1 & 0 & 1 & 0.4 \\
0 & 1 & 0.1 & 0.4 \\
0 & 1 & 0.2 & 0.1 \\
2 & 0 & 0.1 & 0.4
\end{pmatrix}
\]

Adjusted Adjacency Matrix:

\[
\begin{pmatrix}
10001000 \\
01001900 \\
00100110 \\
00001101
\end{pmatrix}
\]
• A partial product generator (PPG)
• A compressor tree (CT), which is the most critical part.
• A carry propagation adder
Deep Q-learning (DQN) based framework

- ResNet-18 as the agent
- A state $s$ refers to a structure.
- An action $a$ refers to modification on current structure $s$
- Pareto-driven Reward

---

Datapath Driven Placement
module test
  input in[3];
  ...
end module
### Dataflow Optimization and Physical Synthesis

#### Datapath Placement

<table>
<thead>
<tr>
<th>APM ICCAD00</th>
<th>SAPT ISPD12</th>
<th>PADE DAC12</th>
<th>MFI ISPD13</th>
<th>FPGA PEs ISCAS19</th>
<th>ASIC PEs ISPD22</th>
</tr>
</thead>
</table>

![Dataflow Diagram](image1.png)  
![Physical Synthesis Diagram](image2.png)

(a)  
(b)
• Dataflow optimization becomes essential for AI chips
  • Schedules operation by data availability
  • Exposes opportunity for parallelism and data reuse
• Datapath regularity gives rise to new physical synthesis approaches
  • Directly determines system performance!

(a) Our flow 200 iteration
(b) Our flow detailed placement
• Datapath driven physical synthesis is not new!

• Placer has little control of exact locations if datapath is generated separately

• Abstract physical model
  • Bit-sliced abstraction
  • Compiled from HDL
  • Blocks are placed abutted

• Two-step heuristic for linear placement
  • quadratic assignment
  • sliding window optimization

---

1Ye, T. Tao, and Giovanni De Micheli. "Data path placement with regularity", ICCAD’00.
Datapath Driven Placement: SAPT [ISPD’12]²

- Structure aware placement obtains better steiner wire length → better routed wire length
- Apply bit-slice alignment constraints for force-directed placer
- Regularity driven detailed placement

Machine learning techniques are involved

- Graph-based (e.g., automorphism) and physical features (e.g., cell area) are analyzed and extracted from the netlist
- Features are fed to SVMs and NNs to classify and evaluate datapath patterns
- Maximize the evaluation accuracies of datapath and non-datapath pattern
- Proposed new placer: PADE

PADE effectively handles datapath in placement.

---

Systolic arrays are a popular choice to support neural network computations

Current FPGA CAD tools cannot synthesize them in high quality

One solution: restrict fixed locations for PEs

- Sufficient DSPs, close to used I/O banks

---

Datapath Driven Systolic Array Placement [ISCAS’19]\(^4\)

---

Simultaneously place the PEs and random logic cells for better solution quality

Analytical global placement with PE array regularization

---

• **Datapath main frame (DMF)**
  
  • a set of $n$ disjoint paths from input to output
  • maximize the number of datapath gates on these paths

• Can be optimally identified by the min-cost max-flow algorithm

DMF identification can be transformed to a network flow problem

---

Wafer-Scale Floorplan
module test
  input in[3];
  ...
end module
The Cerebras CS-1’s Wafer-Scale Engine (WSE) is the largest and most powerful processor ever built.

- Consisting of 800 by 1060 identical processing elements.
- Keep all memory and all computation together on a single monolithic chip.

Die photograph of the Cerebras Wafer-Scale Engine (WSE; at left). For comparison, the largest GPU is shown on the right to scale.

---

Cerebras: Largest Chip Ever Built

- 46,225 mm$^2$ silicon
- 2.6 Trillion transistors, 7nm TSMC
- 850,000 processor cores
- 40 Gigabytes of On-chip Memory
- 9 PByte/s memory bandwidth
- 100 Pbit/s fabric bandwidth
CS-1 Supercomputer Hardware
- **conv**: basic convolution kernel


- **4 execution arguments**: \((h, w, c, k)\) ⇒ variables to be determined.
Problem Formulation

- Determine the **execution parameters** and the **locations** for all kernels. [ISPD’20]\(^8\)
  - Blocks are soft: kernel sizing
  - Floorplanning
  - Optimize performance, wirelength, etc.

---

 Kernel floorplanning. Figure adopted from [ICCAD’20]\(^9\)

---

\(^8\)Michael James, et.al., "Physical Mapping of Neural Networks on a Wafer-Scale Deep Learning Accelerator", ISPD’20.

\(^9\)Bentian Jiang, et.al., CU.POKer: Placing DNNs on Wafer-Scale AI Accelerator with Optimal Kernel Sizing, ICCAD’20.
Initialize best_solution and best_time.
Set lower_bound = 0, upper_bound = MAX_INT.

lower_bound + min_gap \leq upper_bound

\[ targe\_time = \frac{((lower\_bound + upper\_bound)/2) \leq upper\_bound}{binary\_search} \]

Have a legal solution?

Update best_solution and best_time if needed.
Set upper_bound = targe_time.

Set lower_bound = targe_time.

Neighbor-range search based on best_time.

Data-path aware kernel placement under targe_time.

Kernel candidate generation under targe_time.

Post refinement on best_solution.

Output best_solution.

Two-steps Search

- Binary search
- Neighbor-range search
- Post refinement

Searching under Target Time

- Kernel candidates generation with optimal shapes under given target time
- Data-path aware placement

---

Bentian Jiang, et.al., CU.POKer: Placing DNNs on Wafer-Scale AI Accelerator with Optimal Kernel Sizing, ICCAD’20.
Overall Flow

- Given a target time $T$, generate all the kernel candidates with optimal shapes and execution times under $T$.
- According to the connectivity graph, generate the topological order of the kernels for placement.
- Place the kernels compactly row by row in the topological order.
Comparisons with Conventional Floorplanning Heuristics

**Simulated Annealing Placer:**
Max_time: 76698  
Wire_length: 3237  
Adapter_cost: 15  
Score: 110478

**Divide and Conquer Placer:**
Max_time: 65106  
Wire_length: 2650.5  
Adapter_cost: 18  
Score: 93321

**CU.POKer:**
Max_time: 65170  
Wire_length: 1489.5  
Adapter_cost: 12  
Score: 81265

Performance comparisons with SA and DC placers on 8 public benchmarks.
Conclusion
- Arithmetic Unit Synthesis
- Datapath Driven Placement
- Wafer-Scale Floorplan
THANK YOU!