NAAS: Neural Accelerator Architecture Search

Yujun Lin, Mengtian Yang, Song Han

MIT
Yujun Lin is a Ph.D. student at MIT EECS, advised by Professor Song Han. He received his M.S. degree from MIT in 2020 and B.Eng degree from Tsinghua University in 2018.

His research focuses on the intersection of efficient deep learning and accelerator architecture design.
Accelerating Deep Learning Computing

- Both neural architecture and accelerator architecture design are important to enable specialization and acceleration.
Data Driven Approach is Desirable

• Given the huge design space, data-driven approach is desirable, where new architecture design evolves as new designs and rewards are collected

• Hardware-aware Neural Architecture Search (NAS) and auto compiler optimization (e.g., autoTVM)
  • Only focus on off-the-shelf hardware
  • Neglect the freedom in the hardware design space
Design Spaces

<table>
<thead>
<tr>
<th>Key Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accelerator</strong></td>
</tr>
<tr>
<td>Local Buffer Size, Global Buffer Size, #PEs</td>
</tr>
<tr>
<td>Compute Array Size, PE Connectivity</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
</tr>
<tr>
<td>Loop Orders, Loop Tiling Size, Dataflow</td>
</tr>
<tr>
<td><strong>Neural Network</strong></td>
</tr>
<tr>
<td>#Layers, #Channels, Kernel Size, Bypass</td>
</tr>
<tr>
<td>(Input / Weight) Quantization Precision</td>
</tr>
</tbody>
</table>
Design spaces are tightly entangled

- Correlation between design spaces is complicated, and *varies from accelerator to accelerator*
Joint Search Accelerator and Neural Network

• Searching accelerator and neural architecture in one optimization loop offers highly matched solutions
Architecture Design Spaces

**Key Dimensions**

- **Accelerator**
  - Local Buffer Size, Global Buffer Size, #PEs
  - Compute Array Size, PE Connectivity
- **Compiler**
  - Loop Orders, Loop Tiling Size, Dataflow
- **Neural Network**
  - #Layers, #Channels, Kernel Size, Bypass
  - (Input / Weight) Quantization Precision

**Architectural Sizing**

**Connectivity Parameters**
Architecture Design Spaces

Key Dimensions

Accelerator
- Local Buffer Size, Global Buffer Size, #PEs
- Compute Array Size, PE Connectivity

Compiler
- Loop Orders, Loop Tiling Size, Dataflow

Neural Network
- #Layers, #Channels, Kernel Size, Bypass
- (Input / Weight) Quantization Precision

Architectural Sizing

Connectivity Parameters

How to embed these design dimensions for searching?
Convolution Loop Nests

- Convolution loop nests can be divided into two parts: \textit{temporal mapping} and \textit{spatial parallelism}

<table>
<thead>
<tr>
<th>Tensor Dimension</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td>N</td>
</tr>
<tr>
<td>Output Channel</td>
<td>K</td>
</tr>
<tr>
<td>Input Channel</td>
<td>C</td>
</tr>
<tr>
<td>Input Row (Output Row)</td>
<td>Y (Y')</td>
</tr>
<tr>
<td>Input Column (Output Column)</td>
<td>X (X')</td>
</tr>
<tr>
<td>Kernel Row</td>
<td>R</td>
</tr>
<tr>
<td>Kernel Column</td>
<td>S</td>
</tr>
</tbody>
</table>

For \_R in range\(\(R / \_R\)\):
For \_S in range\(\(S / \_S\)\):
For \_C in range\(\(C / T_C\)\):
For \_Y' in range\(\(Y' / T_Y'\)\):
For \_X' in range\(\(X' / T_X'\)\):
For \_r in range\(\(R\)\):
For \_s in range\(\(S\)\):
For \_k in range\(\(K / 16\)\):
For \_y' in range\(\(T_Y'\)\):
For \_x' in range\(\(T_X'\)\):
For \_c in range\(\(T_C / 16\)\):
Parallel-For \_m in range\(\(16\)\):
Parallel-For \_n in range\(\(16\)\):
\[\begin{align*}
  c &= \_C \cdot T_C + \_c \cdot 16; \\
  k &= \_k \cdot 16; \\
  y' &= \_Y' \cdot T_Y' + \_y'; \\
  x' &= \_X' \cdot T_X' + \_x'; \\
  y &= y' + r - R; \\
  x &= x' + s - S; \\
  psum[b,k,y',x'] &= acts[b,x,y,x] + wghts[k,c,y,x];
\end{align*}\]
From Computation Loops To Hardware

• Spatial parallelism determines the PE connectivity
  • \textit{e.g.}, \( C \) (in channels) indicates reduction of partial sum registers
  • \textit{e.g.}, \( K \) (out channels) indicates forward of input feature registers

```python
For _R in range(R / R):
    For _S in range(S / S):
        For _C in range(C / T_C):
            For _Y’ in range(Y’ / T_Y’):
                For _X’ in range(X’ / T_X’):
                    For _c in range(T_C / 16):
                        Parallel-For _m in range(16):
                            Parallel-For _n in range(16):
                                c = _C * T_C + _c * 16;
                                k = _k * 16;
                                y’ = _Y’ * T_Y’ + _y’;
                                x’ = _X’ * T_X’ + _x’;
                                y = y’ + r - R;
                                x = x’ + s - S;
                                psum[b, k, y’, x’] += acts[b, x, y, x] * wgts[k, c, y, x];
```

Mapping

HW Design

Encoding Accelerator and Mappings

Architectural Sizing
- L2 Buffer Size
- L1 Buffer Size
- Number of PEs
- Bandwidth

Connectivity Parameters
- Array Dimension
- Array Dim. Sizes
- PE Connection (Parallel Dim)

Hardware Encoding Vector
- L2 Buffer Size
- L1 Buffer Size
- #PEs
- Bandwidth

Mapping Encoding Vector
- Loop Orders
- Tiling Sizes

To/From DRAM
- Shared Buffer (L2 Scratch Pad)
- Network-On-Chip (NoC)
- Partial Sum
- Private Buffer (L1 Scratch Pad)
- ALU (MAC Unit)

Forward

Parallel Reduction

For _R in range(R / R):
  For _S in range(S / S):
    For _C in range(C / T_C):
      For _Y’ in range(Y’ / T_Y’):
        For _X’ in range(X’ / T_X’):
          For _k in range(K / 16):
            For _y’ in range(T_Y’):
              For _x’ in range(T_X’):
                For _c in range(T_C / 16):
                  Parallel-For _m in range(16):
                    Parallel-For _n in range(16):
                      c = _C * T_C + _c * 16;
                      k = _k * 16;
                      y’ = _Y’ * T_Y’ + _y’;
                      x’ = _X’ * T_X’ + _x’;
                      y = y’ + r - R;
                      x = x’ + s - S;
                      psum[b,k,y’,x’] += acts[b,x,y,x] * wgts[k,c,y,x];
Neural Accelerator Architecture Search

Benchmarks

Update Sample Distribution

Select Best Fits: Low EDP

Evolution

Best Architecture

Mapping Search Space

Sample

Mapping Population

Update Sample Distribution

Select Best Fits: Low EDP

Evolution

Best Mapping

HW Desc.

HW Perf. Estimation (MAESTRO)

EDP

Hardware Evaluation Environment

Accelerator Architecture Search Space

Sample

Accelerator Population

Update Sample Distribution

Select Best Fits: Low EDP

Evolution

Best Mapping
Neural Accelerator Architecture Search

Accelerator Architecture Search
1. *Random sample* accelerator candidates based on multivariate normal distribution $N(\mu_A, \sigma_A, \Sigma_A)$
Neural Accelerator Architecture Search

Compiler Mapping Search
1. Determine mapping space for each accelerator candidates from NN benchmarks
Compiler Mapping Search
2. **Random sample** mapping candidates based on multivariate normal distribution $N(\mu_M, \sigma_M, \Sigma_M)$
Compiler Mapping Search
3. Decode encoding vectors to hardware description;
4. Evaluate Energy-Delay-Product (EDP) for each pair of accelerator candidate and its mapping candidate
Neural Accelerator Architecture Search

Compiler Mapping Search
5. **Select best fits** with lowest EDP

Diagram:
- Mapping Search Space
- Sample
- Decode
- HW Description
- HW Perf. Estimation (MAESTRO)
- EDP
- Hardware Evaluation Environment
- Select Best Fits: Low EDP
- Accelerator Architecture Search Space
- Sample
- Mapping Population
- Evolution
Compiler Mapping Search

5. Update $\mu_M, \sigma_M, \Sigma_M$ to increase the likelihood around best fits
Neural Accelerator Architecture Search

Compiler Mapping Search (iteratively optimizing mappings)
Accelerator Architecture Search

2. Select best fits based EDP using corresponding searched mappings

---

Neural Accelerator Architecture Search
Accelerator Architecture Search

3. Update $\mu_A, \sigma_A, \Sigma_A$ to *increase the likelihood* around best fits
Neural Accelerator Architecture Search

Compiler Mapping Search (iteratively optimizing mappings)

Accelerator Architecture Search (iteratively optimizing accelerator)
Encoding Non-numerical Parameters

- **Index-based Encoding**
  - Increment/Decrement of index value does not convey any physical information

<table>
<thead>
<tr>
<th>Non-Numerical Parameter</th>
<th>Numerical Encoding Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Orders</td>
<td>Index</td>
</tr>
<tr>
<td>CRXKYS</td>
<td>0</td>
</tr>
<tr>
<td>CXYRSK</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Hardware Encoding Vector**

- L2
- L1
- #PE
- BW
- #Dim
- Dim Sizes
- Parallel Dims

**Mapping Encoding Vector**

- Loop Orders
- Tiling Sizes
- Loop Orders
### Importance-based Encoding

1. Fix the dimension position in the encoding vectors
2. Optimizer assigns numerical *importance* to these dimensions
   - by random sampling based on multivariate normal distribution, the same as other numerical parameters such as array sizes
Encoding Non-numerical Parameters

<table>
<thead>
<tr>
<th>L2</th>
<th>L1</th>
<th>#PE</th>
<th>BW</th>
<th>#Dim</th>
<th>Dim Sizes</th>
<th>Parallel Dims</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mapping Encoding Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Orders</td>
</tr>
<tr>
<td>K</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware Encoding Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Orders</td>
</tr>
<tr>
<td>K</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

- Importance-based Encoding
  3. **Sort** the dimensions by the importance value in decreasing order
### Encoding Non-numerical Parameters

#### Hardware Encoding Vector

<table>
<thead>
<tr>
<th>Dim</th>
<th>L2</th>
<th>L1</th>
<th>#PE</th>
<th>BW</th>
<th>#Dim</th>
<th>Dim Sizes</th>
<th>Parallel Dims</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parallel Dims</th>
<th>K</th>
<th>C</th>
<th>Y'</th>
<th>X'</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Mapping Encoding Vector

<table>
<thead>
<tr>
<th>Loop Orders</th>
<th>Tiling Sizes</th>
<th>Loop Orders</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loop Order</th>
<th>K</th>
<th>C</th>
<th>Y'</th>
<th>X'</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tiling Sizes</th>
<th>K</th>
<th>C</th>
<th>Y'</th>
<th>X'</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

#### Other Connectivity Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
<td>16</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Loop Order

<table>
<thead>
<tr>
<th>K</th>
<th>C</th>
<th>Y'</th>
<th>X'</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Decode

```python
Parallel-For c in range(16):
    For r in range(3):
        For x' in range(7):
            Parallel-For k in range(16):
                For y' in range(7):
                    For s in range(3):
```

#### Mapping Encoding Vector

<table>
<thead>
<tr>
<th>Loop Order</th>
<th>C</th>
<th>R</th>
<th>X'</th>
<th>K</th>
<th>Y'</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Other Connectivity Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
<td>16</td>
<td>-</td>
</tr>
</tbody>
</table>
Evaluation

• Design Spaces of NAAS
  • 4 resource constraints: EdgeTPU, NVDLA, Eyeriss, ShiDianNao
  • NAAS searches #PEs at stride of 8, buffer sizes at stride of 16B, array sizes at stride of 2

• CNN Benchmarks
  • Classic large-scale networks: VGG16, ResNet50, UNet
  • Light-weight mobile networks: MobileNetV2, SqueezeNet, MNasNet

• Evaluation Settings
  • Large-scale NN with more hardware resources (EdgeTPU, NVDLA with 1024 PEs)
  • Light-weight NN with limited hardware resources (ShiDianNao, Eyeriss, NVDLA with 256 PEs)
Learning Curves

- As the optimization continues, the EDP mean of NAAS candidates decreases.
- NAAS gradually improves the range of hardware selections.
Search Beyond Architecture Sizing

- Compared to searching the architectural sizing only (e.g., NASAIC, NHAS), searching the connectivity parameters and mapping strategies as well achieves considerable EDP reduction.
NAAS offers better solution than baseline
Jointly Optimize NN, Mapping, Accelerator

For epoch_naas in range(max_naas_epochs):
    accelerators = NAAS_generate_hardware()
    For hw in accelerators:
        For epoch_ofa in range(max_ofa_epochs):
            networks = OFA_generate_networks(accuracy)
            For nn in networks:
                map = NAAS_optimize_mappings(hw, nn)
                edp = NAAS_get_edp(hw, nn, map)
                OFA_update_optimizer(nn, edp)
                best_nn, best_map, best_edp = OFA_update_best(nn, map, edp)
        NAAS_update_optimizer(hw, best_nn, best_map, best_edp)
Evaluation

• Design Space of NAS
  • Once-For-All ResNet NAS
  • 3 width multiplier choices: 0.65, 0.8, 1.0
  • 18 residual blocks at maximum
  • 3 reduction ratios in each residual block: 0.2, 0.25, 0.35
  • Input image size ranges from 128 to 256 at strid of 16
Top-1 Accuracy vs. Normalized EDP

- Eyeriss
- NAAS (accelerator-compiler co-search)
- NAAS (accelerator-compiler-NN co-search)

ResNet-50 on ImageNet

- +2.7% improvement
- 4.42× improvement

Graph showing Top-1 Accuracy vs. Normalized EDP with various data points and the best performance marked.

Diagram of network architecture including:
- Global Buffer
- Network-On-Chip (NoC)
- Output Buffer
- PE
- Local Accumulate
- To/From DRAM

Table:

<table>
<thead>
<tr>
<th>Array Size</th>
<th>18 x 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataflow</td>
<td>K-X' Parallel</td>
</tr>
<tr>
<td>L1 Buffer</td>
<td>496 B</td>
</tr>
<tr>
<td>L2 Buffer</td>
<td>107 KB</td>
</tr>
</tbody>
</table>

Legend:

- BroadCast
- K0
- K1
- K9
- X0
- X1
- X17
- X2
- Array Size 18 x 10
- Dataflow K-X’ Parallel
- L1 Buffer 496 B
- L2 Buffer 107 KB
Compared to NASAIC

<table>
<thead>
<tr>
<th>Search Approach</th>
<th>Arch</th>
<th>Cifar-10 Accuracy</th>
<th>Latency (cycles)</th>
<th>Energy (nJ)</th>
<th>EDP (cycles-nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NASAIC</td>
<td>NVDLA</td>
<td>93.2</td>
<td>3e5</td>
<td>1e9</td>
<td>3e14</td>
</tr>
<tr>
<td></td>
<td>ShiDianNao</td>
<td>91.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAAS</td>
<td>NVDLA</td>
<td>93.2</td>
<td>8e4</td>
<td>2e9</td>
<td>2e14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Search Approach</th>
<th>Co-Search Cost (Gds)</th>
<th>NN Training Cost (Gds)</th>
<th>Total Cost (Gds)</th>
<th>AWS Cost</th>
<th>CO2 Emission</th>
</tr>
</thead>
<tbody>
<tr>
<td>NASAIC</td>
<td>6000N</td>
<td>16 N</td>
<td>6000N</td>
<td>$ 441,000N</td>
<td>41,000N lbs</td>
</tr>
<tr>
<td>NHAS</td>
<td>12+4N</td>
<td>16 N</td>
<td>12+20N</td>
<td>$ 1,500N</td>
<td>150N lbs</td>
</tr>
<tr>
<td>NAAS</td>
<td>&lt;0.25N</td>
<td>50</td>
<td>&lt; 50 + 0.25N</td>
<td>&lt; $ 18N</td>
<td>&lt; 2N lbs</td>
</tr>
</tbody>
</table>

- Gds: GPU days. N: the number of deployment scenarios.
- AWS cost $75/Gd, CO2 emission is 7.5 lbs/Gd.
Neural Accelerator Architecture Search

• Design spaces of hardware, compiler, and neural networks are tightly entangled, joint-optimization is better than separate optimization.

• Optimize both numerical parameters and non-numerical parameters, such as PE connectivity and loop order. Importance-based encoding helps optimize non-numerical parameters.

https://tinyml.mit.edu