Building Smart SoCs
Using Virtual Prototyping for the Design and SoC Integration of Deep Learning Accelerators

Holger Keding
Solutions Architect

SYNOPSYS
Silicon to Software

© Accellera Systems Initiative
Agenda

Deep Learning Market and Technology Trends

• How to Design a Deep Learning Accelerator (DLA)
  • Analytical Performance Modeling
  • Shift Left Architecture Analysis and Optimization with Virtual Prototyping

• Example
  • Importing Network Algorithms as prototxt + generate analytical model spreadsheet
  • Find suited configuration and scaling parameters in analytical model
  • Validate first results, and explore architecture for dynamic and power aspects using Virtual Platforms

• Summary
Increasing number of AI Accelerators

By 2025, market ramp of AI in datacenters

$17 Billion

10x Growth

Source: Qualcomm AI Day Speaker Presentation 2019
Deep Learning Technology Trends

New Neural Network algorithms
– Higher accuracy, lower size and less processing
– But: less data re-use, less cycles per byte

Neural Network Compiler optimizations
– Loop-tiling, -unrolling, and -parallelization
– Splitting and fusing of Neural Network layers
– Memory layout optimization across layers
– Optimized code generation to utilize available hardware accelerators

Deep Learning Accelerator optimizations
– Schedule workload on parallel hardware engines
– Optimize and reduce data transfers to and from memory
AI SoC Design Challenges

Brute-force Processing of Huge Data Sets

• Choosing the right algorithm and architecture: CPU, GPU, FPGA, vector DSP, ASIP
  – CNN graphs evolving fast, need short time to market, cannot optimize for one single graph
  – Joint design of algorithm, compiler, and target architecture
  – Joint optimization of power, performance, accuracy, and cost

• Highly parallel compute drives memory requirements
  – High on-chip and chip to chip bandwidth at low latency
  – High memory bandwidth requirements for parameters and layer to layer communication

• Performance analysis requires realistic workloads to consider dynamic effects
  – Scheduling of AI operators on parallel processing elements
  – Unpredictable interconnect and memory access latencies

Large Design Space drives Differentiation by AI Algorithm & Architecture
Agenda

• Deep Learning Market and Technology Trends

How to Design a Deep Learning Accelerator (DLA)
  • Analytical Performance Modeling
  • Shift Left Architecture Analysis and Optimization with Virtual Prototyping

• Example
  • Importing Network Algorithms as prototxt + generate analytical model spreadsheet
  • Find suited configuration and scaling parameters in analytical model
  • Validate first results, and explore architecture for dynamic and power aspects using Virtual Platforms

• Summary
How to design a DLA?

Analytical Models
+ Good first order
+ Results within minutes
- Omits dynamic effects

High-Level Architecture
+ Good for hardware exploration
+ Simulations in minutes/hours
  ~ Varying Accuracy

RTL Simulation
+ Perfect accuracy
- High computational needs
- High turn-around costs

Functional LT Model (VDK)
+ Good for SW development
+ Simulations in minutes/hours
+ Trace Ops, Memory accesses
- Low Timing Accuracy
Analytical Performance Models

Simple Example: Amdahl’s Law [1]

\[ S_{\text{speedup}} = \frac{1}{(1 - p) + \frac{p}{s}} \]

\[ \lim_{s \to \infty} S_{\text{speedup}} = \frac{1}{1 - p} \]


- **Simple insightful formula, with restricted applicability, though.**
- “**All models are wrong but some are useful**” (George Box, 1978)
Analytical Models – Roofline Models (1)

Roofline: an insightful visual performance model for multicore architectures (Williams, Waterman, Patterson, 2009)

\[ \text{pp}(\text{freq}_{\text{clk}}, \#\text{resources}) \]

- Theoretical maximum compute power
- ILP or SIMD
- Only Thread-Level Parallelism

\[ \frac{2 \text{ operations}}{8 \text{ bytes fetched}} = 0.25 \frac{\text{ops}}{\text{byte}} \]
Analytical Models – Roofline Models (2)

Roofline: an insightful visual performance model for multicore architectures (Williams, Waterman, Patterson, 2009)

\[ \text{slope} = \text{maximum memory bandwidth} \]

\[ \text{op}_{\text{intensity}} \cdot \text{mem\_bandwidth}_{\text{peak}} \]

\[ \frac{2 \text{ operations}}{8 \text{ bytes fetched}} = 0.25 \frac{\text{ops}}{\text{byte}} \]
Analytical Models – Roofline Models (3)

Roofline: an insightful visual performance model for multicore architectures (Williams, Waterman, Patterson, 2009)
Analytical Models – Roofline Models

\[ \text{performance} = \min \left( \text{performance}_{\text{peak}}, \quad \text{opintensity} \cdot \text{membandwidth}_{\text{peak}} \right) \]

\[ \text{opintensity} = \frac{\text{operations}}{\text{bytes}} \]

Roofline: an insightful visual performance model for multicore architectures (Williams, Waterman, Patterson, 2009)
Example: Analytical Model for CNN Convolutional Layer (1)

Conv1 of AlexNet

Maths Textbook

Convolution algorithm

\[ n_{MAC} = o_h \cdot o_w \cdot o_c \cdot k_w \cdot k_h \cdot i_c \]

\[ = 55 \cdot 55 \cdot 96 \cdot 11 \cdot 11 \cdot 3 \]

\[ = 105,415,200 \]
Example: Analytical Model for CNN Convolutional Layer (2)

Conv1 of AlexNet

\[ d_{MAC} = d_{ifmap} + d_{kernel} = (i_w \cdot i_h \cdot i_c + k_w \cdot k_h \cdot i_c \cdot k) \cdot B_t \approx 0.38\text{MiB} \]

\[ n_{MAC} = o_h \cdot o_w \cdot o_c \cdot k_w \cdot k_h \cdot i_c = 55 \cdot 55 \cdot 96 \cdot 11 \cdot 11 \cdot 3 = 105,415,200 \]

⇒ Operational Intensity \( I = \frac{n_{MAC}}{d_{MAC}} \approx 278 \text{ ops/B} \)

But: here we assume unlimited amount of local memory
Example: Analytical Model for CNN Convolutional Layer (3)

Conv1 of AlexNet

\[ n_{MAC} = o_h \cdot o_w \cdot o_c \cdot k_w \cdot k_h \cdot i_c = 55 \cdot 55 \cdot 96 \cdot 11 \cdot 11 \cdot 3 = 105,415,200 \]

\[ d_{MAC} = 2 \cdot n_{MAC} \cdot B_i \approx 420MiB \]

\[ \Rightarrow \text{Operational Intensity } I = \frac{n_{MAC}}{d_{MAC}} \approx \frac{1}{4} \text{ ops/B} \]
Conv1 of AlexNet

**Example: Analytical Model for CNN Convolutional Layer (4)**

**Maths Textbook**

Convolution algorithm

```
for(row=0; row<oh; row++) {
  for(col=0; col<ow; col++) {
    for(k=0; k<oc; k++) {
      for(ti=0; ti<ic; ti++) {
        for(i=0; i<kh; i++) {
          for(j=0; j<kw; j++) {
            L : outputfm[k][row][col] += 
            kernels[k][ti][i][j] * inputfm[t][sw*row+i][sh*col+j];
          }
        }
      }
    }
  }
}
```
Example: Analytical Model for CNN Convolutional Layer (5)

Conv1 of AlexNet – with very simple tiling

Practical setup: limited amount of local memory

Width + Height + Channel + Kernel Tiling

input cube  *  filter kernels  =  output cube

119x227x3
115x227x3

96x11x11x3

27x55x96
28x55x96
Example: Analytical Model for CNN Convolutional Layer (6)

Conv1 with tiling

\[
T_c = s_w T_w + k_w - s_w \\
T_h = s_h T_h + k_h - s_h \\
T_k = k_0, k_1, k_2, k_3 \\
T_w \\
T_h
\]

Now it gets more tricky: Taking into account non-integer relations of tiling parameters and channel dimensions:

\[ d_{\text{input}} = i_c \cdot \left( \frac{k}{T_k} \cdot \left( \frac{o_w}{T_w} \cdot \frac{o_h}{T_h} \cdot (T_c \cdot (S_h T_h + k_h - S_h) \cdot (S_w T_w + k_w - S_w)) \right) \right) \]

\[ d_{\text{weight}} = i_c \cdot \frac{k}{T_k} \cdot \left( \frac{o_w}{T_w} \cdot \frac{o_h}{T_h} \cdot (T_c T_k \cdot k_w k_h) \right) \]

Tiling also brings the operational intensity closer to the optimum HW utilization point.
Example: Analytical Model, Mapping Conv to HW Resources

Tiling parameters and MAC Cell number and depth should match tiling parameters.

# MAC cells can be configured to scale up/down peak performance.
4.7.1 Convolution

<table>
<thead>
<tr>
<th>MAC operations</th>
<th>$o_w \cdot o_h \cdot o_c \cdot k_w \cdot k_h \cdot \frac{i_k}{l_{group}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC cycles</td>
<td>$l_{group} \left[ \frac{i_k}{T_c \cdot l_{group}} \right] \left[ \frac{k}{T_k \cdot l_{group}} \right] \cdot o_h \cdot o_w \cdot k_w \cdot l$</td>
</tr>
<tr>
<td>$d_{input} = d_{CDMA,DAT}$</td>
<td>$i_h \cdot i_w \cdot \frac{i_c \cdot o_k}{\text{atom}<em>{DMA} \cdot l</em>{group}} \cdot \text{atom}<em>{DMA} \cdot l</em>{gate}$</td>
</tr>
</tbody>
</table>

Operational Intensity (Operations/Byte)
### Roofline model

#### 4.7.1 Convolution

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC operations</td>
<td>$o_w \cdot o_h \cdot o_c \cdot k_w \cdot k_h \cdot \frac{i_c}{l_{group}}$</td>
</tr>
<tr>
<td>MAC cycles</td>
<td>$l_{group} \left[ \frac{i_c}{T_c \cdot l_{group}} \right] \left[ \frac{k}{T_k \cdot l_{group}} \right] \cdot o_h \cdot o_w \cdot k_w \cdot l$</td>
</tr>
<tr>
<td>$d_{input} = d_{CDMA_DAT}$</td>
<td>$i_h \cdot i_w \cdot \frac{i_c \cdot b_y}{atom_{DMA} \cdot l_{group}} \cdot atom_{DMA} \cdot l_{gate}$</td>
</tr>
</tbody>
</table>

Operational Intensity (Operations/Byte)
Analytical Model as Python Generated Spreadsheet

Expressions represent both Algorithmic and HW -> calculate attainable performance

|   | A  | B     | C | D   | E   | F | G   | H    | I    | J    | K | L | M  | N   | O   | P   | Q   | R   | S   | T   | U   | V   | W   | X   | Y   | Z   | AA  | AB  |
|---|----|-------|---|-----|-----|---|-----|------|------|------|----|---|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1 |    | num_mac_cell | depth | data_size | dma_ato | edd | ndwidth | d1ute_size | elem_pe | elem_pe | clock | sp | mem | ba |
| 2 | 16 | 64     | 64  | 64  | 32  | 1000 | 12000   | 512       | 16    | 4    |      |    |     |    |

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>type</th>
<th>fmap_w</th>
<th>fmap_h</th>
<th>kernel_w</th>
<th>kernel_h</th>
<th>stride_w</th>
<th>stride_h</th>
<th>pad_w</th>
<th>pad_h</th>
<th>group</th>
<th>bias</th>
<th>output_w</th>
<th>output_h</th>
<th>output_c</th>
<th>surf</th>
<th>num</th>
<th>needs_tilling</th>
<th>d</th>
<th>input_d</th>
<th>d_kgroup</th>
<th>d_output_map_to</th>
<th>sdtp_mode</th>
<th>comp</th>
<th>elem_per</th>
<th>packing_1</th>
<th>packing_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>conv1</td>
<td>CONV</td>
<td>224</td>
<td>224</td>
<td>7</td>
<td>7</td>
<td>64</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1005632</td>
<td>4704</td>
<td>1005632</td>
<td>CONV, X1, ALU</td>
<td>PER_KERNEL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>bn_conv1</td>
<td>BN</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>1005632</td>
<td>0</td>
<td>1005632</td>
<td>X1_ALU</td>
<td>X1_MUL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>scale_conv1</td>
<td>SCALE</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>1005632</td>
<td>0</td>
<td>1005632</td>
<td>X1_MUL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>conv1_relu</td>
<td>RELU</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>1005632</td>
<td>0</td>
<td>1005632</td>
<td>X1_RELU</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>pool1</td>
<td>POOL</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>1005632</td>
<td>0</td>
<td>387200</td>
<td>POP</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>res2a_branch1</td>
<td>CONV</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>19498</td>
<td>387200</td>
<td>CONV, X1, ALU</td>
<td>PER_KERNEL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>bn2a_branch1</td>
<td>BN</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>0</td>
<td>387200</td>
<td>X1_ALU</td>
<td>X1_MUL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>scale2a_branch1</td>
<td>SCALE</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>0</td>
<td>387200</td>
<td>X1_MUL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>res2a_branch2a</td>
<td>CONV</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>18492</td>
<td>387200</td>
<td>CONV, X1, ALU</td>
<td>PER_KERNEL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>bn2a_branch2a</td>
<td>BN</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>0</td>
<td>387200</td>
<td>X1_ALU</td>
<td>X1_MUL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>scale2a_branch2a</td>
<td>SCALE</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>0</td>
<td>387200</td>
<td>X1_MUL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>res2a_branch2a_relu</td>
<td>RELU</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>0</td>
<td>387200</td>
<td>X1_RELU</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>res2a_branch2b</td>
<td>CONV</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>18492</td>
<td>387200</td>
<td>CONV, X1, ALU</td>
<td>PER_KERNEL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>bn2a_branch2b</td>
<td>BN</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>0</td>
<td>387200</td>
<td>X1_ALU</td>
<td>X1_MUL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>scale2a_branch2b</td>
<td>SCALE</td>
<td>55</td>
<td>55</td>
<td>64</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>112</td>
<td>112</td>
<td>64</td>
<td>4</td>
<td>FALSE</td>
<td>387200</td>
<td>0</td>
<td>387200</td>
<td>X1_MUL</td>
<td>LOOSE</td>
<td>LOOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exploring different numbers of MAC cells and their depth

![Graph showing the performance of ResNet18 and AlexNet with varying MAC cell numbers and depths.](image-url)
Analytical Model Summary

What is achieved and what comes next?

What we have seen:
+ Good first order analysis of static effects
+ Results within minutes
~ Requires deep understanding of both algorithm and architecture

What is not covered
- Implementation overhead is hard to predict and not ‘priced in’ in first round
- Omits dynamic effects
- Joint performance and power is difficult
How to design a DLA?

**Analytical Models**
- Good first order
- Results within minutes
- Omits dynamic effects

**High-Level Architecture**
- Good for hardware exploration
- Simulations in minutes/hours
- Varying Accuracy

**Functional LT Model (VDK)**
- Good for SW development
- Simulations in minutes/hours
- Trace Ops, Memory accesses
- Low Timing Accuracy

**RTL Simulation**
- Perfect accuracy
- High computational needs
- High turn-around costs

- Validate back-annotate
- Refine
- Validate back-annotate
Shift Left Architecture Analysis and Optimization

- **NN Workload Model**
- **Deep Learning Accelerator**
- **Multi-core CPU**
- **Interconnect**
- **AI SoC Model**
- **Power, Performance**
- **Translate**
- **Explore**
- **Map**
- **Results**
- **Neural Network**
- **Neural Network Compiler**
- **Deep Learning Accelerator**
- **Multi-core CPU**
- **Interconnect**
- **AI SoC**

Accellera Systems Initiative

Design and Verification Conference and Exhibition European Edition 2019
Platform Architect Ultra
Providing a Comprehensive Library of Generic and Vendor Specific Models

Capture Workload Model

Capture Architecture Model

Interconnect Models
Generic:
- SBL-TLM2-FT (AXI)
- SBL-GCCI (ACE, CHI)

IP Specific:
- Arteris FlexNoC & Ncore
- Arm AHB/APB
- Arm PL300
- Arm SBL-301
- Arm SBL-400
- Synopsys DW AXI

Memory Subsystems
- Generic multiport memory controller (GMPMC)
- DesignWare uMCTL2 memory controller
- DesignWare LPDDR5 memory controller
- Co-simulate with RTL

Traffic, Processors, RTL
- Task-based and trace-based workload models
- Cycle accurate processor for ARM, ARC, Tensilica, CEVA
- RTL Co-simulation/emulation

Analyze Power & Performance
Workload Modeling and Mapping

- **Workload Model**
  - Task level parallelism and dependencies
  - Characterized with processing cycles and memory accesses

- **SoC Platform Model**
  - Accurate SystemC Transaction level models of processing elements, interconnect and memory

- Map workload to platform
- Analyze performance metrics
  - End-to-end constraints
  - Workload activity
  - Utilization of resources
  - Interconnect metrics
    - Latency, Throughput, Contention
    - Outstanding transactions
    - ...
System Level Power Modeling

• Workload Model
  – Task level parallelism and dependencies
  – Characterized with processing cycles and memory accesses

• SoC Platform Model
  – Accurate SystemC Transaction level models of processing elements, interconnect and memory

• System-level Power Overlay Model
  – Define power state machine per component
  – Bind power models to Virtual Prototype
  – Measure power and performance based on real activity and utilization
Automated generation of workloads from AI frameworks
- AI Operator Library for Neural Network modeling
  - E.g. Convolution, Matmul, MaxPool, BatchNorm etc.
- Example workload model of ResNet50 Neural Network
- Utility to convert prototxt description to workload model using AI operator library

AI centric HW architecture model library
- VPUs configured to represent AI compute and DMA engines
- Interconnect and memory subsystem models
- Example performance model of NVIDIA Deep Learning Accelerator (NVDLA)

AI centric analysis views: memory + processing utilization
Workload Model of One Convolution Layer

Scaling parameters reflect the DLA architecture – can be taken from analytical model.
Agenda

• Deep Learning Market and Technology Trends
• How to Design a Deep Learning Accelerator (DLA)
  • Analytical Performance Modeling
  • Shift Left Architecture Analysis and Optimization with Virtual Prototyping

Example
• Importing Network Algorithms as prototxt + generate analytical model spreadsheet
• Find suited configuration and scaling parameters in analytical model
• Validate first results, and explore architecture for dynamic and power aspects using Virtual Platforms

• Summary
Example: Resnet-18 (Inference) with NV-DLA

Goals:
① 100 ms latency, ② minimize power, ③ minimize energy

Optimize Hardware configuration:
– SIMD width
– Burst size, outstanding transactions
– speed of DDR memory and of data path
ResNet-18 Workload model generated with AI-XP
Example: Brief Overview of NVDLA

Convolution Engine (CONV_CORE)
- Works on two sets of data: offline-trained kernels (weights) and input features (images)
- Configurable MAC units and convolutional buffer (RAM)
- Executes operations such as `tf.nn.conv2d`

Single Data Point Processor (SDP)
- Applies linear and non-linear (activation) functions onto individual data points.
- Executes e.g. `tf.nn.batch_normalization`, `tf.nn.bias_add`, `tf.nn.elu`, `tf.nn.relu`, `tf.sigmoid`, `tf.tanh`, and more.

Planar Data Processor (PDP)
- Applies common CNN spatial operations such as min/max/avg pooling
- Executes e.g. `tf.nn.avg_pool`, `tf.nn.max_pool`, `tf.nn.pool`.

Cross-channel Data Processor (CDP)
- Processes data from different channels/features, e.g. local response normalization (LRN) function
- Executes e.g. `tf.nn.local_response_normalization`

Data Reshape Engine (RUBIK)
- Performs data format transformations (splitting, slicing, merging, ...)
- Executes e.g. `tf.nn.conv2d_transpose`, `tf.concat`, `tf.slice`, etc.
VP Simulation Results of Initial Configuration

Performance limited by processing, use wider SIMD data path.
Simulation Reveals Implementation Effects... (1)

Differences between calculated and measured data read/write amount

AlexNet (Norm1):
Expected: 580,800 Bytes
Measured: 654,720 Bytes

Inflation by ~12.72%

Surface 0
Addr: 0x00

Surface 1
Addr: 0x120

$\Rightarrow$ “Dark Bandwidth”
Simulation Reveals Implementation Effects... (2)

Differences between calculated and measured execution time

Convolutional Layers 1&2 of LeNet on NVDLA
Back-Annotate Simulation Findings To Analytical Model

Caffe .prototxt

Platform Architect / Simulation Model

Spreadsheet / Analytical Model
Impact of SIMD Width on Performance

Resource Utilization of CONV Datapath (yellow), CONV DMA (red) and other components
DDR Memory Bandwidth and Power Improvement

- **Resource Utilization**
  - SIMD-128
  - SIMD-64

- **Power Consumption**
  - Conv PE
  - DDR Power

- **Percentage Improvements**
  - 25% faster
  - 10% lower total energy
  - 20% lower average power
Resnet 18 Example Sweep

Goal: 100 ms latency, minimize power & energy

Sweep parameters
- Burst size: 16, 32
- Outstanding transactions: 4, 8
- DDR memory speed: DDR4-1866, DDR4-2400
- Clock frequency of data path: 1, 1.33, 2GHz
- SIMD width: 64, 128 operations per cycle

Sensitivity

Root-Cause Analysis
Sweep Over Hardware Parameters, Latency

- **Outstanding transactions**
- **GHz**
- **SIMD**
- **DDR4 speed**
- **Burst size**
## Power/Performance/Energy Trade-off Analysis

### Optimal Solution

<table>
<thead>
<tr>
<th>Datapath GHz</th>
<th>Outstanding Tx</th>
<th>Burst size</th>
<th>DDR</th>
<th>SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR4-1866M</td>
<td>32</td>
<td>1</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>DDR4-2400</td>
<td>64</td>
<td>2</td>
<td>128</td>
<td>2</td>
</tr>
</tbody>
</table>

### Values
- simtime [ms]
- sum energy [uJ]
- avg power [mW]
Example: Resnet-18 with NV-DLA

Goal:
– 100 ms latency, minimize energy

Optimize Hardware configuration:
– SIMD width: 128 operations per cycle
– Burst size: 32 bytes
– outstanding transactions: 8
– speed of DDR memory: DDR4-1866
– speed of data path: 1GHz
Summary

• Be fast and get it right!
• Shift Left with Virtual Prototyping
• Joint Optimization of Algorithm, Architecture, and Compiler
Thank You

Questions