Cadence Tensilica XNNC
Xtensa Neural Network Compiler: Optimizer

Volodymyr Arbatov, Pedro Vaz Artigas, Xianmin Chen, et. al.
Presenter: Maxim Lukyanov
IEEE ML Compiler Workshop
San Jose, CA
March 4, 2019
XNNC: Overview
Xtensa Neural Network Compiler

Neural Net (NN) arch

NN desc

Trained data

Analyzer

Optimizer

Target libs

Vision
CNN/DNA100 kernels & meta information

Target compiler

DNA 100
XNNC: Analyzer
The front-end

• What’s in it?
  – External framework support
  – Tensor manipulations
  – Network level optimizations & device-level pre-partitioning

Quantization flow

- Calibration Set
  - Net
  - Layer wise histograms
  - Accuracy/Range tradeoffs

- Validation Set
  - Net
  - Evaluation Metric
  - Best quantization profile

- Test Set
  - Net
  - Evaluation Metric

Floating Point NN

A framework adapter (e.g. TF)
- Hardware compatibility checks
  - E.g.: Conv+ReLU
    Conv+BatchNorm+Scale
    Mean/Variance Norm+Conv

- Layer merge
  - E.g.: Conv+ReLU
  - Conv+BatchNorm+Scale
  - Mean/Variance Norm+Conv

- Layer datatype connectivity

- Sparsity

- Quantization

- Fixed point emulation

- Accuracy evaluation
  - XNNC IR

- Various modes, parametrizable aggressiveness, influences from connectivity constraints
  - Bit exact with hardware
  - Metrics for different types of nets

Net Evaluation Metric

Calibration Set
Layer wise histograms
Accuracy/Range tradeoffs
Validation Set
Evaluation Metric
Test Set
Net
Best quantization profile
Top1/Top5 mAP
Mean IoU
Pixel accuracy
User provided
Calibration Set
Net
Layer wise histograms
Accuracy/Range tradeoffs
Validation Set
Net
Evaluation Metric
Best quantization profile
Test Set
Net
Evaluation Metric

to Optimizer

© 2019 Cadence Design Systems, Inc. All rights reserved.
XNNC: Optimizer
The middle-end

• Goal: Map a CNN from high-level pre-optimized IR into a lower-level representation with heavy-lifting optimizations

• Targets a family of Xtensa architectures:
  – DSPs: VP6, VC5, VQ6, VQ7, etc.
  – Accelerators: DNA100 and beyond…

• Recursive optimization process
  – Thousands of alternatives evaluated
  – Guided by performance estimations
ASG: alternatives search graph & interim layout selection

- Node – a data object global memory
- Edge – an operation (conv, pool, etc.)

- On construction:
  - Search space is heuristically capped
    - Contains *meaningful transitions* between data object formats
  - Augmented with weights (perf model)
- As we go: lower/expand, redundancy elimination, local edge re-writing

- Goal: Find a fastest performing subgraph among alternative solutions
  - Dijkstra w/ modified relaxation criteria in presence of “meet” nodes
ASG: More on alternatives…

- Ex: *eltWise add* – 2 inputs, each supplied in a number of alternative layouts from predecessor kernels
ASG: pinning objects to local memory

• Initial

![Initial diagram]

• After pinning to a hierarchy

![After pinning diagram]
ASG: pinning objects to local memory

• Extend ASG with nodes representing local memory objects
  – Only consider objects that fully fit
• Note: decision is made via ASG – not a heuristic!
• Take away:
  – Not always efficient to keep intermediate outputs in local memory
  – May lead to:
    – Smaller tiles, more loop overhead and DMA ops
    – Poorer efficiency of computation kernels
    – Kernels – general term can be ISA, fixed function or library functions
    – Overfetch (ex: reloading coefficients)
ASG: redundancy: strength reduction

- Redundant compute:
  - Ex: Propagation of larger strides up the graph
ASG: Redundancy: copies and layouts

- Most spurious copies introduced for connectivity reasons go away
- Ability to utilize non-linear memory operations for copying
  - Ex: depth-wise concat in place
- Fuse aggressively to minimize memory objects in a given path
- Transform layout of data objects to allow more efficient memory transfers and computations
  - Example: Pad a tensor to conform to memory access alignment rules
ASG: Edge rewriting and operations expansion

- **Graph transforms/lowering:**
  - A number of pattern matching rules
  - Ex: deconvolution is expanded into a set of convolutions + an interleave
    - Interleave can be done in local storage or DMA (i.e. may later disappear)
### XNNC IR: 10K foot view

- **Multistage IR**
  - Operations are subject to stage-based rule rewriting
  - A rule is an transformation or annotation of the IR
  - Explicit semantics for high level operations
    - Rules applied “gnostically”
  - Loop abstractions – first class citizens
  - Converges to straightforward unambiguous code generation
XNNC IR: At the start of compilation

```plaintext
BATCH(
    input = [ax:DATATYPE0:whd:[227:227:3] pitch:[1:227:51529]],
    count = 4,
)

## Formula ##
dagraph( name = 'formula',
    input = [ax:DATATYPE0:whd:[227:227:3] pitch:[1:227:51529]],
)

op( name = 'conv1',
    input = [ax:DATATYPE0:whd:[227:227:3] pitch:[1:227:51529],
             conv1_coeff:DATATYPE0:whdn:[11:11:3:96] pitch:[1:11:121:363],
             conv1_bias:DATATYPE4:bias:[96] pitch:[1],
             DATA: dRandom(2, DATATYPE4:bias:[96] pitch:[1], -128, 127)],
)

## Formula ##
cnnConv3D((0, 0, 0, 0), [11, 11, 3], (4, 4), 1, 1, True, {'PARAM1' ... 'PARAM10'}, None, False, None),

op( name = 'norm1',
             datastructure0:DATATYPE1:[1024] pitch:[1],
             DATA: dRandom(3, DATATYPE1:[1024] pitch:[1], 0, 32767)],
)

## Formula ##
cnnLRN((5, 5), cnnLRN, {'PARAM1', ... 'PARAM5'}),

op( name = 'pool1',
    input = [t13:DATATYPE1:dwh:[1000:1:1] pitch:[1:1000:1000],
             datastructure2:DATATYPE2:[9875] pitch:[1],
             DATA: dRandom(19, DATATYPE2:[9875] pitch:[1], 0, 65535)],
)

## Formula ##
cnnSoftmax(D, {'PARAM1', 'PARAM2'}),
```

XNCC IR: Local transformations and rules

• Applicable to a single operation in the search graph and applied recursively:
  – Each local transformation has a set of rules associated with it
  – Top level transforms are invoked in-order each examining degrees of freedom per rule
    – A degree of freedom is typically associated with a parameter, e.g. tile size
  – If Transform $i$ explored $N$ degrees of freedom applied to a single parameter, it'll yield $N$ variants to Transform $i+1$
  – Given $M$ parameters per rule, with each having a freedom degree it’s a vast space, obviously growing exponentially
  – When exploring the search space various heuristics kick in to cap traversal of the space

• Although order is initially fixed, each transformation may yield some or no opportunities for subsequent ones
XNCC IR: Introducing loop nests

• Tiling – introduces an initial loop nest structure with default index maps.

• Index mapping function:
  – Basically, an affine map for data and iteration spaces
    
    ```
    [ ## Gather ##
      H(  domain: DATATYPE0:align=X:ndwh:[((ind(2, 7) < 7-1)?(14):(12)):3:11:11]  # Ex: Local memory
          range: DATATYPE0:align=X:ndwh:[14:3:11:77]  # Ex: Global memory
          [ # offs dom.n dom.d dom.w dom.h ..loops..
            [0, 1, 0, 0, 0, 0, 0, 0] # rng.n, pitch: 1
            [0, 0, 1, 0, 0, 0, 0, 0] # rng.d, pitch: 64
            [0, 0, 0, 1, 0, 0, 0, 0] # rng.w, pitch: 192
            [0, 0, 0, 0, 1, 0, 11, 0] # rng.h, pitch: 2112
          ]
        ])
    [ ## Scatters ##
      H(  domain: DATATYPE3:dwh:[((ind(2, 7) < 7-1)?(14):(12)):55:((ind(1, 2) < 2-1)?(28):(27))]
          [ # offs dom.d dom.w dom.h ..loops..
            [0, 1, 0, 0, 0, 0, 14, 96] # rng.d, pitch: 1
            [0, 0, 1, 55, 0, 0, 0, 0] # rng.w, pitch: 96
            [0, 0, 0, 1, 28, 0, 0, 0] # rng.h, pitch: 5280
          ]
        ])
    ]]
    ```
    ## Loops ##
    [1, 2, 7, 1]
    attribute loop_order: ALoopOrder(2, 1, 0, 3)
XNNC IR: Example of an index mapping function*

**Strided index mapping:**

\[ f(x_1, \ldots, x_n) \rightarrow y_1, \ldots, y_m \]

\[ y_i = P_i + \sum_{j=1}^{n} x_j S_{ij} \]

**Strided index mapping composition (at compile-time):**

\[ g(z_1, \ldots, z_l) \rightarrow x_1, \ldots, x_n \]

\[ f \circ g \rightarrow f(g(z_1, \ldots, z_l)) \rightarrow y_1, \ldots, y_m \]

---

XNNGC IR: Kernel selection

- Kernels – generic term for us to describe an atomic entity or a primitive
  - Can be an ISA intrinsic, a fixed function unit or a library function

- Kernel descriptions carry attributes
  - represented either computationally or via a composition of index mapping functions

- We narrow a set of kernels applicable to an operation with a number parameters
  - data layout, data types, I/O sizes, strides/dilations, etc.

- Ultimate choice is based on a set of evaluations based on the XNNGC performance model (using the same recursive scheme)

```plaintext
XNNC IR: Kernel selection

- Kernels – generic term for us to describe an atomic entity or a primitive
  - Can be an ISA intrinsic, a fixed function unit or a library function

- Kernel descriptions carry attributes
  - represented either computationally or via a composition of index mapping functions

- We narrow a set of kernels applicable to an operation with a number parameters
  - data layout, data types, I/O sizes, strides/dilations, etc.

- Ultimate choice is based on a set of evaluations based on the XNNGC performance model (using the same recursive scheme)

```
XNNC IR: Parallelization/batching & kernel fusion

• Parallelization and batching:
  – **Parallelization** – effectively picking a loop in a loop nest and performing a loop interchange to make it outermost
    – Can be nested, if hierarchy’s necessary, i.e. vectorization
  – **Batching** introduces a loop over batch of elements
    – Merge it with a loop nest for an operation
    – Easy, just an extra loop dimension!
    – Can also be mapped onto a kernel that handles batching

• Kernel fusion
  – Ex: Fusing 2 convolutions
    – Tile the last convolution first
    – The index function that gathers elements for the bottom convolution will determine the shape of the top convolution output
    – Continue propagate information up to the input of an operation as a whole
    – Intermediate data is kept in local memory
    – The rest is identical to forming any other loop nest
XNNC IR: Software pipeliner

- Main goals:
  - Execution of DMA transfers & heterogenous kernels pipelined
  - Local memory partitioning and buffering
  - Using multi-buffering schemes

- Takeaway:
  - Gets funky very fast in presence of fused kernels due to large ILs and increased buffer pressure
ASG: Final touches

• ASG scheduling:
  – Given a dataflow subgraph of ASG – schedule it
  – The criteria for a topological sort is reduction of storage pressure
    – I.e. minimize the number of buffers that are simultaneously live

• Storage coalescing:
  – Decide which storage is shared by which data objects
  – A typical allocation problem
  – Uses a technique similar to linear scan register allocation sans spilling
Performance model

- >500 DSP kernels used by the compiler to select from
  - More if we consider accelerators
- Each has parameterized model derived from its code structure
  - Covers all DMA transfers, loop nest structure, latency, access patterns for all kernels
- Very high accuracy

<table>
<thead>
<tr>
<th>Network</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>2.5%</td>
</tr>
<tr>
<td>Inception V3</td>
<td>1.6%</td>
</tr>
<tr>
<td>ResNet 50</td>
<td>3.7%</td>
</tr>
<tr>
<td>MobileNet V1</td>
<td>1.4%</td>
</tr>
</tbody>
</table>

- Why do we need an accurate performance model?
  - To generate code, compile, and perform a cycle accurate simulation run: usually takes about 10 seconds per measure per sample
  - Very slow if thousands and thousands of samples!
- Parameters of the performance model of a kernel are built automatically from large number of samples taken during search.
XNNC: Automatic verification

• Simplified reference is generated along with a net binary:
  – high level, bit exact reference functions

• Automated testing of optimized code against reference
  – all necessary harness generated by the compiler

• We heavily use an option to apply transforms randomly
  – Doesn’t use perf model, just randomly transforms the code
  – Ex: random graph transformations, rules, attributes, tile sizes etc.
  – Produces valid implementation, verified against a reference
    – I.e. it “classifies” or “object detects” just may take a while
  – Greatly improves code coverage, may create unanticipated graph topologies.
XNNC Optimizer: what was our goal here?

• To mention the obvious:
  – Powerful framework with state-of-the-art techniques… sure!

• …but, most importantly:
  – High level of abstraction
    – Do not lower too soon, hard or impossible to backtrack
    – Semantic ties with multiple targets – know what you’re compiling for!
  – Unambiguous and straight-forward code generation
  – Ultra precise performance modelling
    – Critical for IP, don’t need HW to tune