MLIR-based offline memory planning and other graph-level optimisations for xcore.ai

Deepak Panickal  Laszlo Kindrat*(Modular)  Scott Roset

*Work done while at XMOS
xcore.ai is a high performance, low latency microcontroller, with 16 logical cores split between two multithreaded processor ‘tiles’, each with 512KB of SRAM.

We have developed an MLIR-based graph compiler(xformer) to optimise TFLite models to deploy on xcore.ai.

I will outline our workflow and focus on three MLIR passes implemented as part of memory usage optimisations.
UNOPTIMISED WORKFLOW

- TensorFlow
  Trained floating point network

- TensorFlow
  Lite convertor

- TensorFlow
  Lite for Microcontrollers Interpreter

- Reference kernels used are very slow as they are not optimised for our Vector Processing Unit

- ONNX
  to TensorFlow convertor

- PyTorch
  Trained network

- Caffe2

- mxnet

- Alternative framework flow

- TFLite
  model
OPTIMISED WORKFLOW WITH GRAPH COMPILER

- **TensorFlow**
  - Trained floating point network

- **TensorFlow**
  - Lite converter

- **TensorFlow Lite Dialect**
  - xcore neural network library

- **xformer**
  - MLIR-based graph compiler

- **ONNX to TensorFlow converter**

- **PyTorch**, **Caffe2**, **mxnet**
  - Trained network

- **Alternative framework flow**

- **Interpreter**
  - TensorFlow Lite for Microcontrollers

- **xcore model**

- **TFLite model**

- **Xcore Dialect**

- **TensorFlow Lite Dialect**
INTERPRETER-LESS WORKFLOW

Why interpreter-less?

- Code size is critical - only include the code for operators used in the model
- Remove unnecessary runtime overhead such as interpreter setup and such code

A tiny open-source project

- Runs tflite-micro interpreter and logs runtime info
- Generates a C++ file for the model

https://github.com/cpetig/tflite_micro_compiler

---

May-23

©2023 XMOS Ltd.
INTERPRETER-LESS WORKFLOW

TensorFlow
Trained floating point network

TensorFlow
Lite convertor

ONNX to TensorFlow convertor

ONNX
Trained network

Alternative framework flow

TFLite model

xformer
MLIR-based graph compiler

+ tflite micro compiler

model as C++ source

xcore neural network library

TensorFlow Lite for Microcontrollers library

Model binary

PyTorch
Caffe2
mxnet

©2023 XMOS Ltd.

EXTERNAL – PUBLIC
### AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

<table>
<thead>
<tr>
<th></th>
<th>Unoptimised</th>
<th>Initial optimisations</th>
<th>Arena planning</th>
<th>Operation splitting</th>
<th>Offloading to flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM used (KB)</td>
<td>1385</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary size (KB)</td>
<td>2590</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)**

<table>
<thead>
<tr>
<th></th>
<th>Unoptimised</th>
<th>Initial optimisations</th>
<th>Arena planning</th>
<th>Operation splitting</th>
<th>Offloading to flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM used (KB)</td>
<td>1385</td>
<td>1258</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary size (KB)</td>
<td>2590</td>
<td>2312</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Plans the tensor arena and allocates offsets for tensors
%2 = "tfl.conv_2d"(%arg0, %0, %1) {dilation_h_factor = 1 : i32, dilation_w_factor = 1 : i32, fused_activation_function = "RELU6", padding = "SAME", stride_h = 2 : i32, stride_w = 2 : i32} :
(tensor<?x160x160x3x!quant.uniform<i8:f32, 0.0039215688593685627:-128>>, tensor<32x3x3x3x!quant.uniform<i8<127:127>:f32:0, 0.0052513480186462402>>, tensor<32x!quant.uniform<i32:f32:0, 2.0593523004208691E-5>>} ->
tensor<?x80x80x32x!quant.uniform<i8:f32, 0.023529412224888802:-128>>

%5 = "tfl.depthwise_conv_2d"(%2, %3, %4) {depth_multiplier = 1 : i32, dilation_h_factor = 1 : i32, dilation_w_factor = 1 : i32, fused_activation_function = "RELU6", padding = "SAME", stride_h = 1 : i32, stride_w = 1 : i32} :
(tensor<?x80x80x32x!quant.uniform<i8:f32, 0.023529412224888802:-128>>, tensor<1x3x3x3x32x!quant.uniform<i8<127:127>:f32:3, 0.021009480580687523>>}, tensor<32x!quant.uniform<i32:f32:0, 4.9434072570875287E-4>>} ->
tensor<?x80x80x32x!quant.uniform<i8:f32, 0.023529412224888802:-128>>
MEMORY PLAN ANALYSIS PASS

Walk and calculate non-constant tensor sizes

- Prepare map of firstUsed and lastUsed ops
- Identify ops that can be overlapped
- Greedily allocate offsets for size-sorted tensors
- Prepare the allocation plan
MEMORY PLAN ANALYSIS PASS

Walk and calculate non-constant tensor sizes

1. Prepare map of firstUsed and lastUsed ops
2. Identify ops that can be overlapped
3. Greedily allocate offsets for size-sorted tensors
4. Prepare the allocation plan

Filter and bias are constants

Inputs and outputs are non-constants

Convolutions:
- Filter sizes: 96x1x1x16, 24x1x1x96, 144x1x1x24
- Bias sizes: 96, 96, 24

Depthwise Convolution:
- Weights size: 1x3x3x96
- Bias size: 96

ReLU activation function
MEMORY PLAN ANALYSIS PASS

- Use Liveness Analysis pass in MLIR
- This is used to identify simultaneously alive tensors
- The largest simultaneously alive tensors defines the peak memory usage for the graph
MEMORY PLAN ANALYSIS PASS

- Identify ops that can be overlapped
- Greedily allocate offsets for size-sorted tensors
- Prepare map of firstUsed and lastUsed ops
- Walk and calculate non-constant tensor sizes
- Prepare the allocation plan

Diagram:

- Conv2D
  - filter (96x1x1x16)
  - bias (96)
  - ReLU
  - 1x80x80x96

- Pad
  - paddings (4x2)
  - 1x81x81x96

- DepthwiseConv2D
  - weights (1x3x3x96)
  - bias (96)
  - ReLU
  - 1x40x40x96

- Conv2D
  - filter (24x1x1x96)
  - bias (24)
  - 1x40x40x24

Input and output for Pad can be overlapped
The allocation algorithm is similar to the one used in Tensorflow Lite for Microcontrollers for arena planning.

Doing it in MLIR gives us much more control.

We want to minimise the total memory used while avoiding fragmentation.
The allocation plan is an array of offsets, one for every tensor in the model, written to flatbuffer metadata.

-1 is used for constant tensors.

```
0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,204800,-1,-1,0,-1,1,204800,-1,-1,629856,629856,629856,827136,629856,800256,629856,-1,-1
```
# An Example – MobileNetV2 (160x160x3, Alpha = 1.0)

<table>
<thead>
<tr>
<th></th>
<th>Unoptimised</th>
<th>Initial optimisations</th>
<th>Arena planning</th>
<th>Operation splitting</th>
<th>Offloading to flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM used (KB)</td>
<td>1385</td>
<td>1258</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary size (KB)</td>
<td>2590</td>
<td>2312</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

<table>
<thead>
<tr>
<th></th>
<th>Unoptimised</th>
<th>Initial optimisations</th>
<th>Arena planning</th>
<th>Operation splitting</th>
<th>Offloading to flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM used (KB)</td>
<td>1385</td>
<td>1258</td>
<td>798</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary size (KB)</td>
<td>2590</td>
<td>2312</td>
<td>2312</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Splits operations into multiple ops to reduce peak memory usage
OPERATION SPLIT PASS

- Annotation can be specified by command-line options
- We also have partially working auto-annotation based on Liveness Analysis pass and peak memory usage

Annotate ops to be split
- Insert strided slices and concat
- Calculate overlap and hoist strided slices
- Repeat until desired level
OPERATION SPLIT PASS

1. Annotate ops to be split
2. Insert strided slices and concat
3. Calculate overlap and hoist strided slices
4. Repeat until desired level
Insert strided slices and concat

Annotate ops to be split

Calculate overlap and hoist strided slices

Repeat until desired level

OPERATION SPLIT PASS

May-23

©2023 XMOS Ltd.
OPERATION SPLIT PASS

- Calculate overlap and hoist strided slices
- Annotate ops to be split
- Insert strided slices and concat
- Repeat until desired level

Slices overlap due to change in size
OPERATION SPLIT PASS

Repeat until desired level

Annotate ops to be split
Insert strided slices and concat
Calculate overlap and hoist strided slices

Pad values split across slices
Repeat until desired level

Annotate ops to be split
Insert strided slices and concat
Calculate overlap and hoist strided slices

May-23

EXTERNA–PUBLIC
©2023 XMOS Ltd.
## AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

<table>
<thead>
<tr>
<th></th>
<th>Unoptimised</th>
<th>Initial optimisations</th>
<th>Arena planning</th>
<th>Operation splitting</th>
<th>Offloading to flash</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RAM used (KB)</strong></td>
<td>1385</td>
<td>1258</td>
<td>798</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Binary size (KB)</strong></td>
<td>2590</td>
<td>2312</td>
<td>2312</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

©2023 XMOS Ltd.
## AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

<table>
<thead>
<tr>
<th></th>
<th>Unoptimised</th>
<th>Initial optimisations</th>
<th>Arena planning</th>
<th>Operation splitting</th>
<th>Offloading to flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM used (KB)</td>
<td>1385</td>
<td>1258</td>
<td>798</td>
<td>363</td>
<td></td>
</tr>
<tr>
<td>Binary size (KB)</td>
<td>2590</td>
<td>2312</td>
<td>2312</td>
<td>2337</td>
<td></td>
</tr>
</tbody>
</table>

©2023 XMOS Ltd.
Offload weights in the model to a flash image which can then be streamed in at runtime
FLASH IMAGE PASS

Conditionally insert XC_LoadConstant ops

- Change execution order
- Combine loads and lower to XC_LoadFlash op

1x160x160x3

Conv2D
- filter (32x3x3x3)
- bias (32)

1x80x80x32

DepthwiseConv2D
- weights (1x3x3x32)
- bias (32)

1x80x80x32

Relu6

1x80x80x16

Conv2D
- filter (16x1x1x32)
- bias (16)

Filter and bias are constants

1x80x80x96

Conv2D
- filter (96x1x1x16)
- bias (96)

Relu6
Conditionally insert XC_LoadConstant ops

- Change execution order
- Combine loads and lower to XC_LoadFlash op

Constant is too small
FLASH IMAGE PASS

Change execution order

Conditionally insert XC_LoadConstant ops
Combine loads and lower to XC_LoadFlash op

We want these loads done just before it is needed by Conv2D

Combine loads and lower to XC_LoadFlash op
Combine loads and lower to XC_LoadFlash op

Conditionally insert XC_LoadConstant ops

Change execution order

Two loads are to the same user op
Combine loads and lower to XC_LoadFlash op

Conditionally insert XC_LoadConstant ops

Change execution order

©2023 XMOS Ltd.
### AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

<table>
<thead>
<tr>
<th></th>
<th>Unoptimised</th>
<th>Initial optimisations</th>
<th>Arena planning</th>
<th>Operation splitting</th>
<th>Offloading to flash</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RAM used (KB)</strong></td>
<td>1385</td>
<td>1258</td>
<td>798</td>
<td>363</td>
<td></td>
</tr>
<tr>
<td><strong>Binary size (KB)</strong></td>
<td>2590</td>
<td>2312</td>
<td>2312</td>
<td>2337</td>
<td></td>
</tr>
</tbody>
</table>
AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

<table>
<thead>
<tr>
<th></th>
<th>Unoptimised</th>
<th>Initial optimisations</th>
<th>Arena planning</th>
<th>Operation splitting</th>
<th>Offloading to flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM used (KB)</td>
<td>1385</td>
<td>1258</td>
<td>798</td>
<td>363</td>
<td>384</td>
</tr>
<tr>
<td>Binary size (KB)</td>
<td>2590</td>
<td>2312</td>
<td>2312</td>
<td>2337</td>
<td>488</td>
</tr>
</tbody>
</table>
CHALLENGES AND FUTURE PLANS

- **TensorFlow**
  - Trained floating point network

- **TensorFlow Lite convertor**

- **xcore neural network library**

- **TFLite model**

- **xformer**
  - MLIR-based graph compiler
  - + tflite micro compiler

- **ONNX to TensorFlow convertor**

- **Caffe2 & mxnet**
  - trained network

- **Alternative framework flow**

- **TensorFlow Lite for Microcontrollers library**

- **Model binary**

- **PyTorch**

- **MLIR-based graph compiler**

- **ONNX**

- **tflite micro compiler**

- **model as C++ source**

- **External - Public**

©2023 XMOS Ltd.

36
CHALLENGES AND FUTURE PLANS

Challenges
- Identify prior art to reuse
- Find what is the “correct” way, what idioms to use

Future plans
- Adapt memory plan analysis to add page in/out ops for handling larger models
- Better execution order
EXPERIENCE WITH MLIR

The MLIR framework made it easy for us to quickly add optimisations and productise our AI tools.

We reuse a lot of code from TensorFlow and the MLIR project.

Being able to work at the right level of abstraction is intuitive.
Thank you!

- deepakpanickal@xmos.com

All code is available publicly at
- https://github.com/xmos/ai_tools
- https://github.com/xmos/lib_tflite_micro