ML-on-CPU: should vectorization happen in the LLVM backend or higher up the stack?
The plan

• TVM
• Scheduling
• Vectors
• SVE (Scalable Vector Extension)
What is TVM
What is TVM

• Another end-to-end machine learning compilation stack!
What is TVM

- Another end-to-end machine learning compilation stack!
- Ahead-of-time
What is TVM

• Another end-to-end machine learning compilation stack!
• Ahead-of-time
• Targets wide range of CPUs, GPUs, embedded devices and accelerators
What is TVM

• Another end-to-end machine learning compilation stack!
• Ahead-of-time
• Targets wide range of CPUs, GPUs, embedded devices and accelerators
• Lowers to LLVM
TVM stack

- resnet50.pt
- mobilenet.pb
TVM stack

resnet50.pt

mobilenet.pb

conv2d

conv2d

pooling

softmax
for i in range(0, 16):
    for j in range(0, 16):
for i in range(0, 16):
    for j in range(0, 16):

TIR scheduling:
- split/tile
- unroll
- vectorize
- reorder loops
- compute_at
- ... and many more
for i in range(0, 16):
    for j in range(0, 16):

TIR scheduling:
- split/tile
- unroll
- vectorize
- reorder loops
- compute_at
- ... and many more

LLVM further transformations...
How much scheduling should happen in TVM – example of conv2d
Conv2d

1. Padding

![Input tensor diagram with dimensions 8x8x3](image)
Conv2d

1. Padding

input

pad_input
Conv2d

2. Convolve with weights

```
pad_input
```

```
10 10 3
```

© 2023 Arm
Conv2d

2. Convolve with weights

pad_input

weights

...
Conv2d

2. Convolve with weights
Conv2d

3. Bias add

conv_output

8  8  16
Conv2d

3. Bias add

conv_output

\[
\text{conv\_output} + \text{bias}
\]
Conv2d

4. ReLU

max(

, 0)

conv_bias
A larger conv2d as loops

```python
for ih in range(0, 60):
    for iw in range(0, 60):
        for ic in range(0, 72):
            pad_input[...] = if_then_else(..., input[...], 0)

for oh in range(0, 60):
    for ow in range(0, 60):
        for oc in range(0, 24):
            output[...] = 0
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ic in range(0, 72):
                        output[...] = output[...] + pad_input[...] * weights[...]

for oh in range(0, 60):
    for ow in range(0, 60):
        for oc in range(0, 24):
            output[...] = output[...] + bias[...]

for oh in range(0, 60):
    for ow in range(0, 60):
        for oc in range(0, 24):
            relu[...] = max(output[...], 0)
```
A larger conv2d as loops

```python
for ih in range(0, 60):
    for iw in range(0, 60):
        for ic in range(0, 72):
            pad_input[...] = if_then_else(..., input[...], 0)

for oh in range(0, 60):
    for ow in range(0, 60):
        for oc in range(0, 24):
            output[...] = 0
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ic in range(0, 72):
                        output[...] = output[...] + pad_input[...] * weights[...]

for oh in range(0, 60):
    for ow in range(0, 60):
        for oc in range(0, 24):
            output[...] = output[...] + bias[...]

for oh in range(0, 60):
    for ow in range(0, 60):
        for oc in range(0, 24):
            relu[...] = max(output[...], 0)
```

15 loops 😞
LLVM’s optimisations

```python
for ih_iw in range(0, 60 * 60):
    for ic in range(0, 72):
        pad_input[...] = if_then_else(..., input[...], 0)

for oh_ow in range(0, 60 * 60):
    for oc_outer in range(0, 4):
        for oc_inner in range(0, 6):
            output[...] = 0
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ic_outer in range(0, 18):
                        output[...] = output[...] +
                        pad_input[vec(ic_inner = 4)...] * weights[vec(ic_inner = 4)...

for oh_ow in range(0, 60 * 60):
    for oc in range(0, 24):
        output[...] = max(output[...] + bias[...], 0)
```
LLVM’s optimisations

```python
for ih_iw in range(0, 60 * 60):
    for ic in range(0, 72):
        pad_input[...] = if_then_else(..., input[...], 0)

for oh_ow in range(0, 60 * 60):
    for oc_outer in range(0, 4):
        for oc_inner in range(0, 6):
            output[...] = 0
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ic_outer in range(0, 18):
                        output[...] = output[...] +
                        pad_input[vec(ic_inner = 4)...] * weights[vec(ic_inner = 4)...]
            for oh_ow in range(0, 60 * 60):
                for oc in range(0, 24):
                    output[...] = max(output[...] + bias[...], 0)
```
LLVM’s optimisations

for ih_iw in range(0, 60 * 60):
    for ic in range(0, 72):
        pad_input[...] = if_then_else(..., input[...], 0)

for oh_ow in range(0, 60 * 60):
    for oc_outer in range(0, 4):
        for oc_inner in range(0, 6):
            output[...] = 0
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ic_outer in range(0, 18):
                        output[...] = output[...] + pad_input[vec(ic_inner = 4)...] * weights[vec(ic_inner = 4)...]

for oh_ow in range(0, 60 * 60):
    for oc in range(0, 24):
        output[...] = max(output[...] + bias[...], 0)
LLVM’s optimisations

```python
for ih_iw in range(0, 60 * 60):
    for ic in range(0, 72):
        pad_input[...] = if_then_else(..., input[...], 0)

for oh_ow in range(0, 60 * 60):
    for oc_outer in range(0, 4):
        for oc_inner in range(0, 6):
            output[...] = 0
        for kh in range(0, 3):
            for kw in range(0, 3):
                for ic_outer in range(0, 18):
                    output[...] = output[...] + pad_input[vec(ic_inner = 4)...] * weights[vec(ic_inner = 4)...]

for oh_ow in range(0, 60 * 60):
    for oc in range(0, 24):
        output[...] = max(output[...] + bias[...], 0)
```

- outer axes fused
- split output channels $oc = 24 \rightarrow oc_{outer} = 4$ and $oc_{inner} = 6$
- two loop nests merged
LLVM’s optimisations

```python
for ih_iw in range(0, 60 * 60):
    for ic in range(0, 72):
        pad_input[...] = if_then_else(..., input[...], 0)

for oh_ow in range(0, 60 * 60):
    for oc_outer in range(0, 4):
        for oc_inner in range(0, 6):
            output[...] = 0
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ic_outer in range(0, 18):
                        output[...] = output[...] +
                                        pad_input[vec(ic_inner = 4)] * weights[vec(ic_inner = 4)]

for oh_ow in range(0, 60 * 60):
    for oc in range(0, 24):
        output[...] = max(output[...] + bias[...], 0)
```

- **outer axes fused**
- **split output channels** oc = 24 -> oc_outer = 4 and oc_inner = 6
- **unroll oc_outer**
- **two loop nests merged**
LLVM’s optimisations

```python
for ih_iw in range(0, 60 * 60):
    for ic in range(0, 72):
        pad_input[...] = if_then_else(..., input[...], 0)

for oh_ow in range(0, 60 * 60):
    for oc_outer in range(0, 4):
        for oc_inner in range(0, 6):
            output[...] = 0
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ic_outer in range(0, 18):
                        output[...] = output[...] + pad_input[vec(ic_inner = 4)...] * weights[vec(ic_inner = 4)...]

for oh_ow in range(0, 60 * 60):
    for oc in range(0, 24):
        output[...] = max(output[...] + bias[...], 0)
```

- **Outer axes fused**
- **Split output channels** $oc = 24 \rightarrow oc_{\text{outer}} = 4$ and $oc_{\text{inner}} = 6$
- **Unroll** $oc_{\text{outer}}$
- **Vectorize across input channels**
- **Two loop nests merged**
TVM’s scheduling

```python
def schedule(input, weights, bias):
    for oh in parallel(0, 60):
        for ow_outer in range(0, 15):
            for ow_inner0 in range(0, 4):
                for ic in range(0, 72):
                    pad_input[...] = input[...]

        for oc_outer in range(0, 6):
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ow_inner1 in range(0, 4):
                        conv[vec(oc_inner)[...]] = broadcast(...)
                    for ic in range(0, 72):
                        for ow_inner2 in range(0, 4):
                            conv[vec(oc_inner)[...]] = pad_input[...] + weights[vec(oc_inner)[...]] * conv[vec(oc_inner)[...]]

        for oc_outer in range(0, 6):
            for ow_inner3 in range(0, 4):
                relu[vec(oc_inner)] = max(conv[...]+bias[vec(oc_inner)[...]])
```
TVM’s scheduling

```python
for oh in parallel(0, 60):
    for ow_outer in range(0, 15):
        for ow_inner0 in range(0, 4):
            for ic in range(0, 72):
                pad_input [...] = input [...]  

    for oc_outer in range(0, 6):
        for kh in range(0, 3):
            for kw in range(0, 3):
                for ow_inner1 in range(0, 4):
                    conv[vec(oc_inner)] = broadcast(...)
                for ic in range(0, 72):
                    for ow_inner2 in range(0, 4):
                        conv[vec(oc_inner)] = pad_input [...] + weights[vec(oc_inner)] * conv[vec(oc_inner)]

    for oc_outer in range(0, 6):
        for ow_inner3 in range(0, 4):
            relu[vec(oc_inner)] = max(conv [...] + bias[vec(oc_inner)])
```

compute nests merged!
for oh in parallel(0, 60):
    for ow_outer in range(0, 15):
        for ow_inner0 in range(0, 4):
            for ic in range(0, 72):
                pad_input[…] = input[…]
        for oc_outer in range(0, 6):
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ow_inner1 in range(0, 4):
                        conv[vec(oc_inner)…] = broadcast(…)
                    for ic in range(0, 72):
                        for ow_inner2 in range(0, 4):
                            conv[vec(oc_inner)…] = pad_input[…] + weights[vec(oc_inner)…] * conv[vec(oc_inner)…]
        for oc_outer in range(0, 6):
            for ow_inner3 in range(0, 4):
                relu[vec(oc_inner)] = max(conv[…] + bias[vec(oc_inner)…])
TVM’s scheduling

```python
for oh in parallel(0, 60):
    for ow_outer in range(0, 15):
        for ow_inner0 in range(0, 4):
            for ic in range(0, 72):
                pad_input[...] = input[...]

    for oc_outer in range(0, 6):
        for kh in range(0, 3):
            for kw in range(0, 3):
                for ow_inner1 in range(0, 4):
                    conv[vec(oc_inner)...] = broadcast(...)

                for ic in range(0, 72):
                    for ow_inner2 in range(0, 4):
                        conv[vec(oc_inner)...] = pad_input[...] + weights[vec(oc_inner)...] * conv[vec(oc_inner)...]

        for oc_outer in range(0, 6):
            for ow_inner3 in range(0, 4):
                relu[vec(oc_inner)] = max(conv[...] + bias[vec(oc_inner)...])
```

- `oc_inner` is a vector
- Width is split into two and loops have been reordered
- Compute nests merged!
TVM’s scheduling

```python
for oh in parallel(0, 60):
    for ow_outer in range(0, 15):
        for ow_inner0 in range(0, 4):
            for ic in range(0, 72):
                pad_input[...] = input[...]
        for oc_outer in range(0, 6):
            for kh in range(0, 3):
                for kw in range(0, 3):
                    for ow_inner1 in range(0, 4):
                        conv[vec(oc_inner)...] = broadcast(...)
                    for ic in range(0, 72):
                        for ow_inner2 in range(0, 4):
                            conv[vec(oc_inner)...] =
                            pad_input[...] + weights[vec(oc_inner)...] * conv[vec(oc_inner)...

    for oc_outer in range(0, 6):
        for ow_inner3 in range(0, 4):
            relu[vec(oc_inner)] = max(conv[...] +
            bias[vec(oc_inner)...])

```

- `oc_inner` is a vector
- Width is split into two and loops have been reordered
- Compute nests merged!
- 5.8x faster
Do we need all TIR scheduling primitives?

YES
1. split
2. reorder
3. compute_at

MAYBE
1. unroll
2. vectorize
Vectorize

```python
for i in range(0, 10):
    for j in Vectorize(range(0, 4)):
        A[i * 4 + j] = B[i * 4 + j]
```
Vectorize

```python
for i in range(0, 10):
    for j in Vectorize(range(0, 4)):
        A[i * 4 + j] = B[i * 4 + j]

for i in range(0, 10):
```
Vectorize

```python
for i in range(0, 10):
    for j in Vectorize(range(0, 4)):
        A[i * 4 + j] = B[i * 4 + j]

for i in range(0, 10):
```

vector node TIR

\[ A[\text{base} : \text{base} + \text{extent}] \]
for i in range(0, 10):
    for j in Vectorize(range(0, 4)):
        A[i * 4 + j] = B[i * 4 + j]

for i in range(0, 10):
What to vectorize in conv2d
What to vectorize in conv2d

```python
for ic in range(0, 72):
    output[...] = output[...] + pad_input[...] * weights[...]
```
What to vectorize in conv2d

```python
for ic in range(0, 72):
    output[...] = output[...] + pad_input[...] * weights[...]
```

vectorize the innermost loop!
What to vectorize in conv2d

```python
for ic in range(0, 72):
    output[...] = output[...] + pad_input[...] * weights[...]
```

vectorize the innermost loop!

72 / 4 = 18 vector instructions
Different kind of vectors – scalable vectors
SVE (Scalable Vector Extension)

scalable vector

%105 = load <vscale x 4 x float>, ptr %Aptr
SVE (Scalable Vector Extension)

scalable vector

\[ %105 = \text{load} \ <\text{vscale} \times 4 \times \text{float}> , \text{ptr} \ %Aptr \]

compile time
unknown
SVE (Scalable Vector Extension)

scalable vector
%105 = load <vscale x 4 x float>, ptr %Aptr

In hardware:

128 bits vscale = 1
SVE (Scalable Vector Extension)

scalable vector
%105 = load <vscale x 4 x float>, ptr %Aptr

In hardware:

128 bits  vscale = 1

256 bits  vscale = 2

compile time unknown
SVE (Scalable Vector Extension)

scalable vector
%105 = load <vscale x 4 x float>, ptr %Aptr

In hardware:
128 bits vscale = 1
256 bits vscale = 2

Should TVM care about this?
Vectorize for SVE

```python
for ic in range(0, 72):
    output[...] = output[...] + pad_input[...] * weights[...]
```
Vectorize for SVE

```
for ic in range(0, 72):
    output[...] = output[...] + pad_input[...] * weights[...]
```

still vectorize the innermost loop!
Vectorize for SVE

for ic in range(0, 72):
    output[...] = output[...] + pad_input[...] * weights[...]

still vectorize the innermost loop!

SVE:
72 / (4 * vscale) vector instructions
That awful loop nest again

for oh in parallel(0, 60):
    for ow_outer in range(0, 15):
        for ow_inner0 in range(0, 4):
            for ic in range(0, 72):
                pad_input[...] = input[...]
        for oc_outer in range(0, 6):
            for kw in range(0, 3):
                for ow_inner1 in range(0, 4):
                    conv[vec(oc_inner)...] = broadcast(...)
            for ic in range(0, 72):
                for ow_inner2 in range(0, 4):
                    conv[vec(oc_inner)...] = pad_input[...] + weights[vec(oc_inner)...] * conv[vec(oc_inner)...]  
                for oc_outer in range(0, 6):
                    for ow_inner3 in range(0, 4):
                        relu[vec(oc_inner)] = max(conv[...] + bias[vec(oc_inner)...])]
That awful loop nest again

```python
for oh in parallel(0, 60):
    for ow_outer in range(0, 15):
        for ow_inner0 in range(0, 4):
            for ic in range(0, 72):
                pad_input[...] = input[...]
        for oc_outer in range(0, 6):
            for kw in range(0, 3):
                for kh in range(0, 3):
                    for ow_inner1 in range(0, 4):
                        conv[vec(oc_inner)...] = broadcast(...)  
                        for ic in range(0, 72):
                            for ow_inner2 in range(0, 4):
                                conv[vec(oc_inner)...] =
                    pad_input[...] + weights[vec(oc_inner)...] * conv[vec(oc_inner)...]
                                for oc_outer in range(0, 6):
                                    for ow_inner3 in range(0, 4):
                                        relu[vec(oc_inner)] = max(conv[...] +
                                bias[vec(oc_inner)...])
```
That awful loop nest again

for oh in parallel(0, 60):
    for ow_outer in range(0, 15):
        for ow_inner0 in range(0, 4):
            for ic in range(0, 72):
                pad_input[...] = input[...]
        for oc_outer in range(0, 6):
            for kw in range(0, 3):
                for kh in range(0, 3):
                    for ow_inner1 in range(0, 4):
                        conv[vec(oc_inner)...] = broadcast(...)
                    for ic in range(0, 72):
                        for ow_inner2 in range(0, 4):
                            conv[vec(oc_inner)...] = pad_input[...] + weights[vec(oc_inner)...] * conv[vec(oc_inner)...
                            for oc_outer in range(0, 6):
                                for ow_inner3 in range(0, 4):
                                    relu[vec(oc_inner)] = max(conv[...] + bias[vec(oc_inner)...])

potential scalable vector
weights[base : base + 4 * vscale]
That awful loop nest again

for oh in parallel(0, 60):
  for ow_outer in range(0, 15):
    for ow_inner0 in range(0, 4):
      for ic in range(0, 72):
        pad_input[...] = input[...]
     for oc_outer in range(0, 6):
       for kw in range(0, 3):
         for kh in range(0, 3):
           for ow_inner1 in range(0, 4):
             conv[vec(oc_inner)...] = broadcast(...)
           for ic in range(0, 72):
             for ow_inner2 in range(0, 4):
               conv[vec(oc_inner)...] =
               pad_input[...] + weights[vec(oc_inner)...] * conv[vec(oc_inner)...]
             for oc_outer in range(0, 6):
               for ow_inner3 in range(0, 4):
                 relu[vec(oc_inner)] = max(conv[...] + bias[vec(oc_inner)...])

matching outer loop

potential scalable vector

weights[base : base + 4 * vscale]
That awful loop nest again

```python
for oh in parallel(0, 60):
    for ow_outer in range(0, 15):
        for ow_inner0 in range(0, 4):
            for ic in range(0, 72):
                pad_input[...] = input[...]
        for oc_outer in range(0, 6):
            for kw in range(0, 3):
                for ow_inner1 in range(0, 4):
                    conv[vec(oc_inner)...] = broadcast(...)  
                    for ic in range(0, 72):
                        for ow_inner2 in range(0, 4):
                            conv[vec(oc_inner)...] = 
                                          pad_input[...] + 
                                          weights[vec(oc_inner)...] * conv[vec(oc_inner)...] 
                        for oc_outer in range(0, 6):
                            for ow_inner3 in range(0, 4):
                                relu[vec(oc_inner)] = max(conv[...] + 
                                          bias[vec(oc_inner)...]])
```

matching outer loop

```python
for oc_outer in range(0, 24 / (4 * vscale))
```

potential scalable vector

```python
weights[base : base + 4 * vscale]
```
Vsate in TIR - summary
Vscale in TIR - summary

• Helps to include scalable vectors into complex loop structures
Vsacle in TIR - summary

- Helps to include scalable vectors into complex loop structures
- Enables targeting gather load and scatter store from the schedules
Vscale in TIR - summary

• Helps to include scalable vectors into complex loop structures
• Enables targeting gather load and scatter store from the schedules
• Paves way to supporting extensions like SME
Vscale in TIR - summary

- Helps to include scalable vectors into complex loop structures
- Enables targeting gather load and scatter store from the schedules
- Paves way to supporting extensions like SME
- Complexity and maintenance cost
Thank You
Danke
Gracias
Grazie
谢谢
ありがとうございます
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה