Homework 07 – Implementation

Arthur J. Redfern
arthur.redfern@utdallas.edu

---

0 Outline

1 Reading
2 Theory
3 Practice

1 Reading

1. Implementation
   Motivation: understand xNN implementation
   https://github.com/arthurredfern/UT-Dallas-CS-6301-CNNs/blob/master/Lectures/xNNs_070_Implementation.pdf

Complete

2. Efficient processing of deep neural networks: a tutorial and survey
   Motivation: an alternative presentation of xNN hardware circa 2017
   https://arxiv.org/abs/1703.09039

Complete

3. A new golden age for computer architecture
   Motivation: a very nice talk from the Turing award winners on the past and future of hardware and software, available in text and video form
   https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext
   https://www.youtube.com/watch?v=3LVeEjsn8Ts

Complete

2 Theory

4. Write out all of the terms of Strassen based matrix matrix multiplication for **C** = **A B** with BLAS dimensions M = N = K = 4 by applying the Strassen decomposition twice (an initial decomposition then a recursive decomposition). Use the following notation for the scalars in the 3 matrices

[C(0, 0) C(0, 1) C(0, 2) C(0, 3)] =	[A(0, 0) A(0, 1) A(0, 2) A(0, 3)]	[B(0, 0) B(0, 1) B(0, 2) B(0, 3)]
[C(1, 0) C(1, 1) C(1, 2) C(1, 3)]	[A(1, 0) A(1, 1) A(1, 2) A(1, 3)]	[B(1, 0) B(1, 1) B(1, 2) B(1, 3)]
[C(2, 0) C(2, 1) C(2, 2) C(2, 3)]	[A(2, 0) A(2, 1) A(2, 2) A(2, 3)]	[B(2, 0) B(2, 1) B(2, 2) B(2, 3)]
[C(3, 0) C(3, 1) C(3, 2) C(3, 3)]	[A(3, 0) A(3, 1) A(3, 2) A(3, 3)]	[B(3, 0) B(3, 1) B(3, 2) B(3, 3)]

Apologies for the un beautiful matrix formatting, but the above is meant to represent C = A B.

Initial decomposition:

[A(0, 0) A(0, 1)]	[A(0, 2) A(0, 3)]	A'(0,0)	A'(0,1)
[A(1, 0) A(1, 1)]	[A(1, 2) A(1, 3)]		
[A(2, 0) A(2, 1)]	[A(2, 2) A(2, 3)]	A'(1,0)	A'(1,1)
[A(3, 0) A(3, 1)]	[A(3, 2) A(3, 3)]		

[B(0, 0) B(0, 1)]	[B(0, 2) B(0, 3)]	B'(0,0)	B'(0,1)
[B(1, 0) B(1, 1)]	[B(1, 2) B(1, 3)]		
[B(2, 0) B(2, 1)]	[B(2, 2) B(2, 3)]	B'(1,0)	B'(1,1)
[B(3, 0) B(3, 1)]	[B(3, 2) B(3, 3)]		

[C(0, 0) C(0, 1)]	[C(0, 2) C(0, 3)]	C'(0,0)	C'(0,1)
[C(1, 0) C(1, 1)]	[C(1, 2) C(1, 3)]		
[C(2, 0) C(2, 1)]	[C(2, 2) C(2, 3)]	C'(1,0)	C'(1,1)
[C(3, 0) C(3, 1)]	[C(3, 2) C(3, 3)]		

[C'(0, 0), C'(0, 1)]		[A'(0, 0) A'(0, 1)]	[B'(0, 0) B'(0, 1)]
[C'(1, 0), C'(1, 1)]		[A'(1, 0) A'(1, 1)]	[B'(1, 0) B'(1, 1)]

The new matrix becomes with the prime terms:

S_1 = (A'(0,0) + A'(1,1)) * (B'(0,0) + B'(1,1))
S_2 = (A'(1,0) + A'(1,1)) * (B'(0,0))
S_3 = (A'(0,0)) * (B'(0,1) – B'(1,1))
S_4 = (A'(1,1)) * (B'(1,0) – B'(0,0))
S_5 = (A'(0,0) + A'(0,1)) * (B'(1,1))
S_6 = (A'(1,0) – A'(0,0)) * (B'(0,0) + B'(0,1)))
S_7 = (A'(0,1) – A'(1,1)) * (B'(1,0) + B'(1,1))

Re-decomposition:
S_1 = (A'(0,0) + A'(1,1)) * (B'(0,0) + B'(1,1))

C1'(0,0)	C1'(0,1)	=	A(0,0) + A(2,2)	A(0,1) + A(2,3)	*	B(0,0) + B(2,2)	B(0,1) + B(2,3)
C1'(1,0)	C1'(1,1)		A(1,0) + A(3,2)	A(1,1) + A(3,3)		B(1,0) + B(3,2)	B(1,1) + B(3,3)

S_1	=	C1'(0,0)	C1'(0,1)
		C1'(1,0)	C1'(1,1)

Similarly, when calculating, S_2, S_3, S_4, S_5, S_6 and S_7 all multiplications are recalculated following the Strassen's multiplication rules the second time for second decomposition.

Finally after the all second decomposition results, the final values are put into the following values:
C(0,0) = S1 + S4 – S5 + S7
C(0,1) = S3 + S5
C(1,0) = S2 + S4
C(1,1) = S1 – S2 + S3 + S6

3 Practice

5. We've seen the importance of quantization to reducing memory, reducing data movement and increasing compute (with the same resources). To gain more experience with quantization, read the following PyTorch quantization introduction work through the following example (you'll need to write some of your own code to download models etc.):

- https://pytorch.org/docs/stable/quantization.html
- https://pytorch.org/tutorials/advanced/static_quantization_tutorial.html

6. Complete

7. [Optional] Creating tools to automate work is a common activity when working with xNNs. For this problem, you will build a tool to predict performance using a simplified network specification for a simplified hardware architecture specification.

The network specification can come from scraping the graph of the network created by PyTorch (preferred), or it can be hand specified as follows. Specify a network as a text file using the following to describe each layer (ID is a unique identifier like a unique number)

Operation IDx
    Parameters
        Operation type, parameter 0, parameter 1, …
    Input tensors
        Previous layer IDa, precision, pointer, dimension 0, dimension 1, …
        Previous layer IDb, precision, pointer, dimension 0, dimension 1, …
        …
    Output tensors
        Precision, pointer, dimension 0, dimension 1, …
        Precision, pointer, dimension 0, dimension 1, …
        …

Note that this format encodes all tensor dimensions directly into the network description. While suboptimal from an experimentation perspective (e.g., changing the input size to the network requires changing parameters in all layers), this is convenient for a deployment perspective as it allows for better optimization with respect to memory placement and data movement. Pointer

is a memory address, but we won't use it here so you can ignore it (I'm not going to make you figure out optimal memory placement from a buffer reuse perspective).

Previous layer IDa, IDb, … are included with the input tensors to indicate the previous layer that needs to complete before this input is available, a convenient value to track for graph optimization. Input tensors can be input feature maps, filter coefficients, … basically any value that's not a parameter encoded with the operation. If the input tensor is an external edge to the graph (e.g., the network input, a filter weight, …) the previous layer ID can be set to -1 indicating that there is no previous layer.

For a CNN style 2D convolution operation, example parameters include:

Input feature map grouping
Input feature map row pad
Input feature map col pad
Filter row stride
Filter col stride
Filter row dilation
Filter col dilation

Using a strategy similar to that of CNN style 2D convolution, additional operations can be specified for bias addition, ReLU, max pooling, average pooling, global average pooling, matrix multiplication, … and other layer components as necessary.

For the network to specify in the above format, choose 1 or more of: ResNet-50, ResNeXt-50, MobileNet V2, MobileNet V3, Inception V4, NASNet, MnasNet or AmoebaNet. Set the input tensor for the network to 1 x 3 x 512 x 1024 at 16 bit precision. Also assume all filter coefficients are quantized to 16 bit precision.

Why am I having you go through all this trouble? I want you to think of networks as graphs of operations controlled by parameters that map input tensors to output tensors. This is a key for understanding performance and optimal implementation strategies.

Next, specify the hardware as a text file using the following parameters (an example value is provided for each parameter in <>)

Data movement	
DDR bits	<64 b / 8 B>
DDR frequency	<3200 MHz>
DDR availability	<0.50>
DDR efficiency	<0.80>
Memory	
Off device	<¥>
On device	<4 MB>

Compute

Frequency (cycles per second)	<1 GHz>
Matrix M, N, K at 8 bit precision	<32, 32, 32>
Matrix M, N, K at 16 bit precision	<32, 16, 16>
Matrix M, N, K at 32 bit precision	<32, 8, 8>
Cycles per matrix tile	<32>
Vector N at 8 bit precision	<32>
Vector N at 16 bit precision	<16>
Vector N at 32 bit precision	<8>
Cycles per vector op	<1>

The matrix op will tile CNN style 2D convolution lowered to matrix matrix multiplication. If the M_{conv}, N_{conv} and K_{conv} values of CNN style 2D convolution are not an integer multiple of the M, N and K values of the matrix operation, then use M_{tiles} = ceil(M_{conv}/M), N_{tiles} = ceil(N_{conv}/N) and K_{tiles} = ceil(K_{conv}/K), note that this represents a potential computational inefficiency. The matrix operation will also be used for matrix matrix multiplication with the same tiling strategy. The matrix compute time for the CNN style 2D convolution and matrix matrix multiplication operations is

Matrix time = (M_{tiles} * N_{tiles} * K_{tiles}) * (cycles per matrix tile) / (frequency)

All other operations will assumed to be at vector speed and their vector time is approximated as

Vector time = ceil(number of ops / vector N) * (cycles per vector op) / (frequency)

The total compute time for an operation is

Compute time = matrix time + vector time

In order to determine data movement time, locations for each of the tensors needs to be determined. As a first order approximation, mark all filter coefficient tensors as off device. Also mark the network input image tensor and output prediction tensor as off device. For all other feature map tensors, mark them as on device if their size is smaller than that of the on device memory and mark them off device otherwise.

For all tensors marked off device, in order for them to be used in an operation, they need to be moved on device. The amount of time that it takes to move a tensor from off device to on device is

Tensor move time = ((precision / 8) * dimension 0 * dimension 1 * …) / (DDR bandwidth)

where DDR bandwidth = (DDR bits / 8) * (DDR frequency) * (DDR availability) * (DDR efficiency). The data movement time for an operation is the data movement time for all tensors that need to be moved from off device to on device for that operation

Data movement time = tensor move time 1 + tensor move time 2 + …

Now, the total time for an operation can be bound as

Serial operation time = (compute time) + (data movement time)
Parallel operation time = max(compute time, data movement time)

Repeat the serial operation time and parallel operation time calculation for all operations in the graph. Then sum these results for an overall bound on the network time

Serial network time = serial operation time 1 + serial operation time 2 + …
Parallel network time = parallel operation time 1 + parallel operation time 2 + …

So to summarize, you've generated

1. A text file describing all operations in a chosen network
2. A performance prediction program for
    a. Estimating the compute time for each operation
    b. Determining a memory placement for each tensor
    c. Estimating the data movement time for each operation
    d. Estimating a serial and parallel time bound for each operation
    e. Estimating a serial and parallel time bound for the full network

The final step is to have the performance prediction program write the results to a CSV file for each layer and the overall network.

Optional not attempted here

Submitted by – Kapil Gautam