Low Memory Multi Channel Convolution using General Matrix Multiplication
Small fast methods and how to pick the right ones for a given deep neural network

Andrew Anderson Aravind Vasudevan, Cormac Keane and David Gregg
International Symposium & Workshop on Many-Core Computing
January 17th 2018
Libraries to exploit parallel hardware with software

• How can we exploit parallel hardware?
  • Multiple processors, cores, vector, ILP, GPU

• One solution is to build libraries for key functions
  • E.g. general matrix multiplication (GEMM)
  • Careful manual optimization
  • Also domain specific library generators (e.g. Spiral)

• Libraries have been very successful
  • Especially for deep neural networks
  • “Why GEMM is at the heart of deep learning” – Pete Warden’s blog
Agenda

• Current ways to implement neural network convolution using GEMM libraries
  • Mostly \textit{im2col}

• Improved approaches requiring (much) less memory
  • We avoid the challenge of writing low-level parallel code
  • But we need to jump through some hoops to make it work

• How to select the right approach?
CNN Primer

Convolutional Layers

3D Tensor input

3D Tensor output

Ground Truth
Dog – 0.4   Dog – 1.0
Field – 0.3 ← Field – 0.0
Tree – 0.3   Tree – 0.0

Backpropagation

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th></th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>8</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>
Importance of convolutional layers

- About 89% of forward inference time spent on convolutional layers

Figure: Distribution of forward inference time for AlexNet on CPU. Figure credit [1]

Multiple channel multiple kernel convolution
Convolution as GEMM

- Convolution can be implemented as matrix multiplication with a Toeplitz matrix
  - Decades of work on matrix-matrix multiplication (GEMM)
  - Easy way to quickly get good DNN performance

- Why not just write a fast loop nest?
  - It’s much more difficult than it looks
**im2col**

- **Classical GEMM-based convolution**
  - Widely used in popular deep learning frameworks
  - Based on constructing a Toeplitz matrix
- **Expands the input by a factor of $k^2$**
  - Input tensor has size $C \times H \times W$
  - Im2col requires additional $C \times H \times W \times K^2$ space
    - (Less if the convolution is strided)
- **Additional space can be a big problem on embedded systems**
  - May exceed available memory
  - Poor data locality leading to cache misses and memory traffic
Convolutional layer implementation – *im2col*
**GEMM-based convolution without im2col**

- Im2col needs lots of memory for the patch matrix
  - $C \times H \times W \times K^2$ space
- Could we find another algorithm for convolution that
  - Uses GEMM to achieve high speeds
  - But does not build a patch matrix
- We propose a family of new GEMM-based algorithms
  - Based on sums of convolutions
  - No need for patch matrix
GEMM-based convolution by sum of scaled matrices

• Consider 3x3 convolution with one input channel, one convolution kernel

Convolution becomes addition of sub-matrices, each scaled by one element of the kernel.

To get the right answer we need to shift the matrix entries before adding (next slide)
**GEMM-based convolution by sum of scaled matrices**

- Consider a 3x3 convolution with one input channel, one convolution kernel.

```
  [kernel] * [image] = [result]
```

We need to shift the sub-arrays of to line up the values correctly before adding the corresponding values.
**GEMM-based convolution by sum of scaled matrices**

- **We can extend our sum of scaled matrices algorithm to input with multiple channels**
  - Replace matrix scaling with 1x1 DNN convolution
  - KxK DNN convolution can be computed as the sum of \( K^2 \) 1x1 DNN convolutions

- **1x1 DNN convolution**
  - Can be implemented with matrix-matrix multiplication (GEMM)
  - No extra patch matrices needed

- **Downside is more GEMM calls**
  - We do \( K^2 \) GEMM calls versus just one GEMM call for im2col
1x1 DNN convolution

Matrix-Matrix Multiplication (GEMM)
Accumulating Algorithm

- Compute one 1x1 convolution at a time and add to output

- Temporary space is just one result matrix at a time
  - CHW space for one result matrix
  - Versus \( K^2 CHW \) for im2col
Accumulating Algorithm
**GEMM-accumulating algorithm**

- BLAS GEMM is already an accumulating algorithm
  - Takes an optional matrix parameter to accumulate to
  - So we can do the accumulation as part of the GEMM call
  - Potentially faster than a post-pass loop
- There are *significant* complications
  - We shift the result matrices when accumulating
  - How should we manage pixels at the boundaries of images?
GEMM-accumulating algorithm
Managing boundary pixels with GEMM accumulation

• Convolutions stop at the edge of images
  • All convolution algorithms deal with boundaries as special cases
  • But we are building our sum of 1x1 convolutions with GEMM

• We’re completely misusing the GEMM accumulate
  • At boundaries we spill into and over-write the next row
  • Lots of wrong values in results matrix

• Two strategies
  • Post-pass fix-up of values
  • Dynamically modify input matrix with carefully-placed zeros
Dynamically modifying input matrix – the guillotine
Space complexity of algorithms

- Input image of size CHW
  - C channels, H pixels high, W pixels wide
- Kernels
  - KxK size, C channels, M kernels
  - K is typically 1, 3 or 5

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>#GEMM calls</th>
<th>Ops/GEMM call</th>
<th>Extra space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Im2col</td>
<td>1</td>
<td>$K^2CWHM$</td>
<td>O($K^2CWH$)</td>
</tr>
<tr>
<td>Kernel accumulating</td>
<td>$K^2$</td>
<td>$CWHM$</td>
<td>O($CWH$)</td>
</tr>
<tr>
<td>GEMM accumulating</td>
<td>$K^2$</td>
<td>$CWHM$</td>
<td>O($KW+HC+WC$)</td>
</tr>
</tbody>
</table>
Experiments

- Measurements
  - ARM Cortex A-57
  - Intel Core i5-4570
- Specifically for inference
  - Mini-batchsize = 1
- Multiple implementations of each method

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>#variants implemented</th>
<th>Variant used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Im2col</td>
<td>23</td>
<td>Row-major, copy short from patch</td>
</tr>
<tr>
<td>Kernel accumulating</td>
<td>2</td>
<td>Row-major</td>
</tr>
<tr>
<td>GEMM accumulating</td>
<td>4</td>
<td>Row-major $AB^T$</td>
</tr>
</tbody>
</table>

Intel Core i5-4570 single core
Intel Core i5-4570 multi core

![Chart showing speedup vs. im2col for different kernels: im2col, Kernel-acc, GEMM-acc. The chart compares the performance of different kernels on an Intel Core i5-4570 multi core processor, with the y-axis representing speedup and the x-axis representing different kernel types.]
ARM Cortex A57 single core

![Graph showing speedup vs. im2col with bars for im2col, Kernel-acc, and GEMM-acc]
ARM Cortex A57 multi core

![Graph showing Speedup vs. im2col for different kernels. The graph compares im2col, Kernel-acc, and GEMM-acc.]
Extra memory required
**Key Takeaways**

- DNN convolution can leverage optimized GEMM libraries
  - Im2col is fast but needs lots of extra space
- Our GEMM-accumulating approach offers
  - Similar performance
  - At a fraction of the additional space
- We have these libraries for key operations
  - Despite other advances in parallelization we still build them
  - E.g. BLIS has quite a lot of assembly
  - There can be a different type of software complexity from using the libraries in unintended ways
## Selecting primitive functions to implement layers

### Several different algorithmic approaches to convolution

<table>
<thead>
<tr>
<th>Approach</th>
<th>Good for</th>
<th>Bad for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple loop nest</td>
<td>Memory size</td>
<td>Execution time</td>
</tr>
<tr>
<td>GEMM – im2col</td>
<td>Good all-rounder; strided</td>
<td>Memory size</td>
</tr>
<tr>
<td>GEMM – accumulating</td>
<td>Memory size</td>
<td>Strided; few channels</td>
</tr>
<tr>
<td>MEC algorithm</td>
<td>Strided convolution</td>
<td>Execution time*</td>
</tr>
<tr>
<td>FFT convolution</td>
<td>Large kernels</td>
<td>Memory size</td>
</tr>
<tr>
<td>Winograd convolution</td>
<td>3x3, 5x5 execution time</td>
<td>Various arbitrary</td>
</tr>
</tbody>
</table>

* Based on our implementation; the MEC authors have not released their implementation
Selecting primitive functions to implement layers

- We have library of approx. 70 convolutions
  - Many variants of each main algorithm
  - Many different data formats and layouts

- Given a DNN, how do we select the best one?
  - Each primitive operates on a given data format
  - Primitives using different formats are incompatible

- Legalization pass
  - Can insert data format conversions between incompatible layers
  - But may need a chain of conversions

- We profile execution times of individual layers
  - And find the right combination analytically
Partitioned Boolean quadratic programming (PBQP)
Partitioned Boolean quadratic programming (PBQP)
Intel Haswell one core

Baseline is simple \textit{sum2d} algorithm on one core
Intel Haswell multiple cores

Baseline simple sum2d algorithm on one core
ARM Cortex-A57 one core

Baseline is simple `sum2d` algorithm on one core
Intel Haswell multiple cores

Baseline is simple sum2d algorithm on one core
Key Takeaways

• We have these libraries for key operations
  • But transforming your problem can create additional issues

• There are lots of ways to do convolution with/without GEMM
  • No one best algorithm for all cases
  • Some algorithms are only good in special cases
  • Significant speeds available from with good selection
20th Workshop on Compilers for Parallel Computing

- April 16 – 18 2018
- Trinity College Dublin
- Abstract submission
  - February 15th 2018
- https://cpc2018.scss.tcd.ie/
Thank You

- https://bitbucket.org/STG-TCD/trinnity

This research was supported by Science Foundation Ireland grant 12/IA/1381.

Related research within our group is supported by Science Foundation and ARM, grant 13/RC/2094 (CALCULUS) to Lero – the Irish Software Research Centre, and by the H2020 project number 732204 (BONSEYES).