Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

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

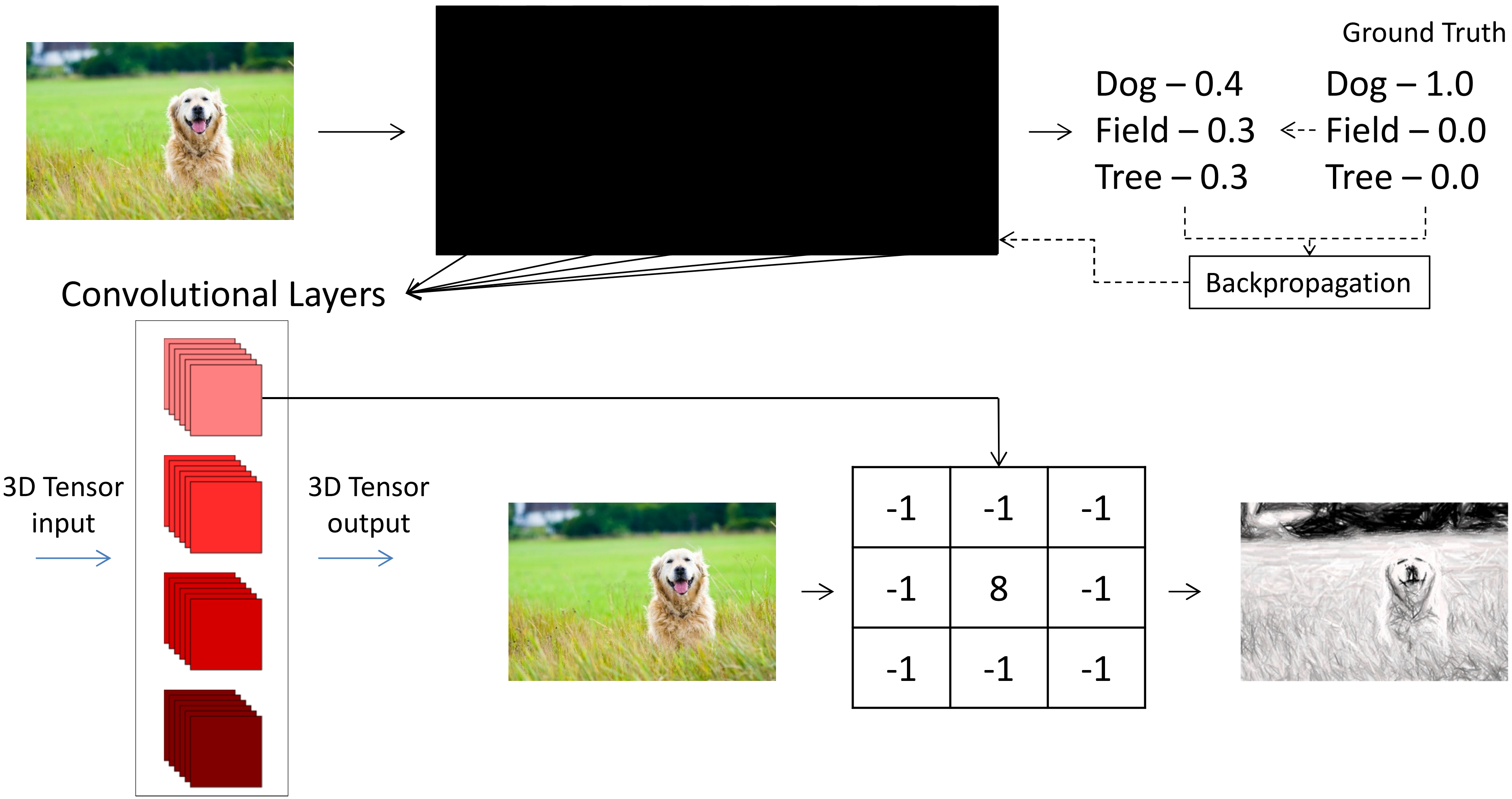
Libraries to exploit parallel hardware with software

- **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 *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



Ground Truth
Dog – 0.4
Field – 0.3
Tree – 0.3
Dog – 1.0
Field – 0.0
Tree – 0.0
Backpropagation
Convolutional Layers
3D Tensor input
3D Tensor output
-1 -1 -1
-1 8 -1
-1 -1 -1

Importance of convolutional layers

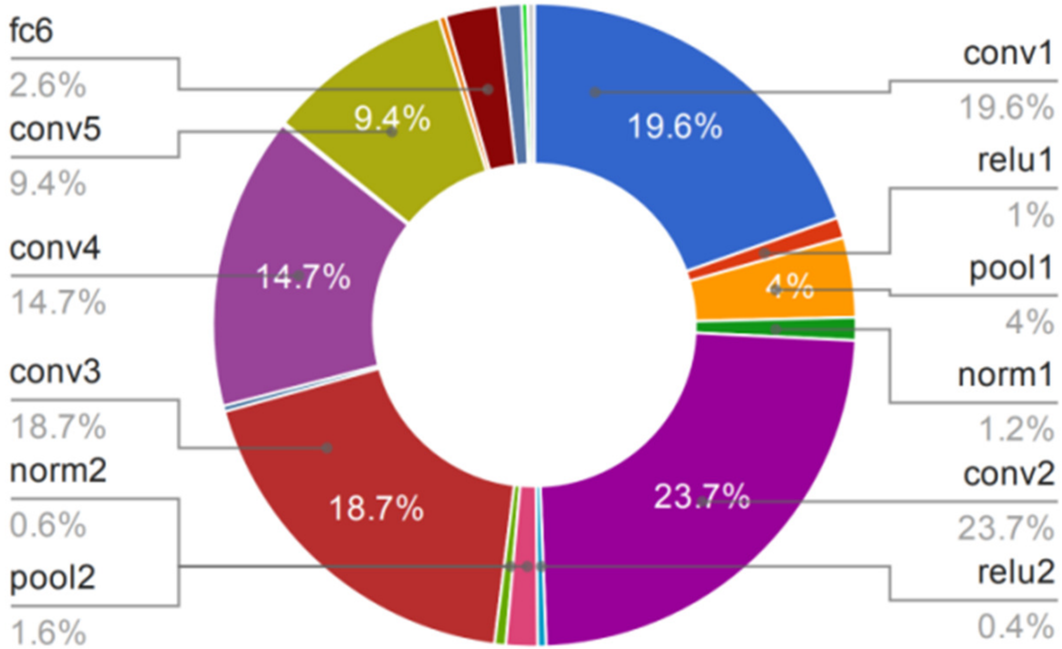
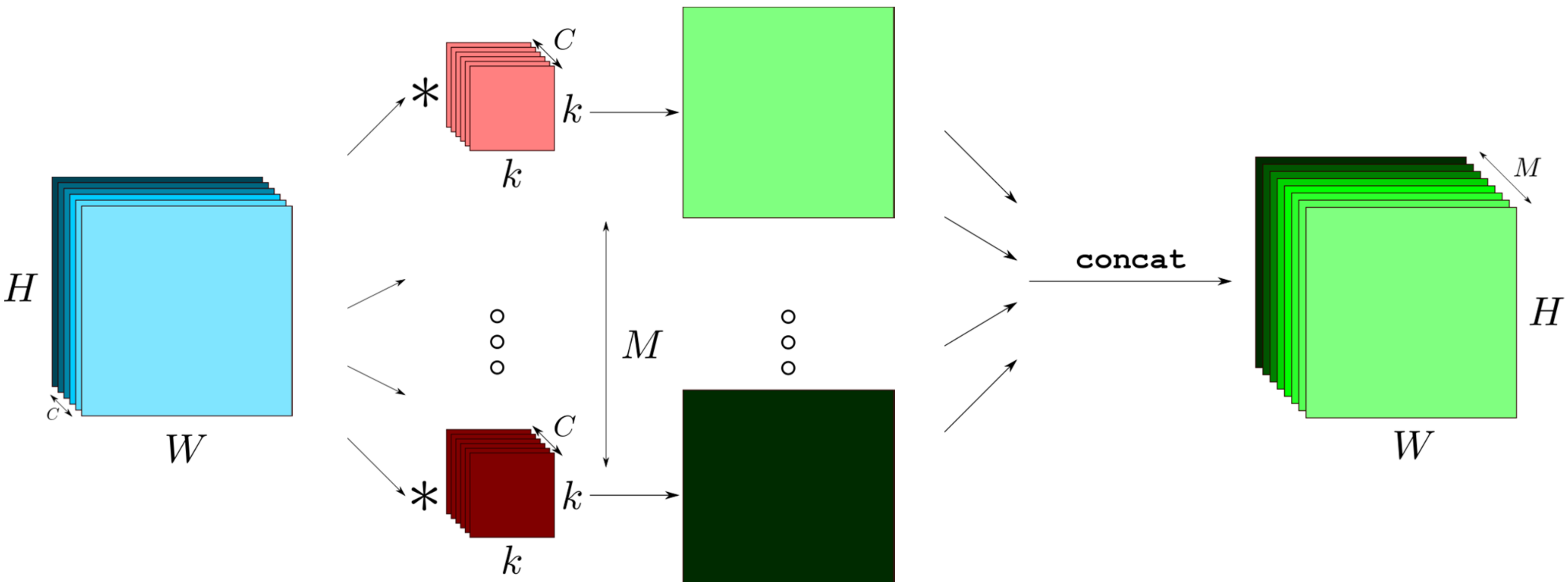


Figure: Distribution of forward inference time for AlexNet on CPU. Figure credit [1]

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

[1] – Jia, Yangqing. *Learning semantic image representations at a large scale*. University of California, Berkeley, 2014.

Multiple channel multiple kernel convolution



Convolution as GEMM

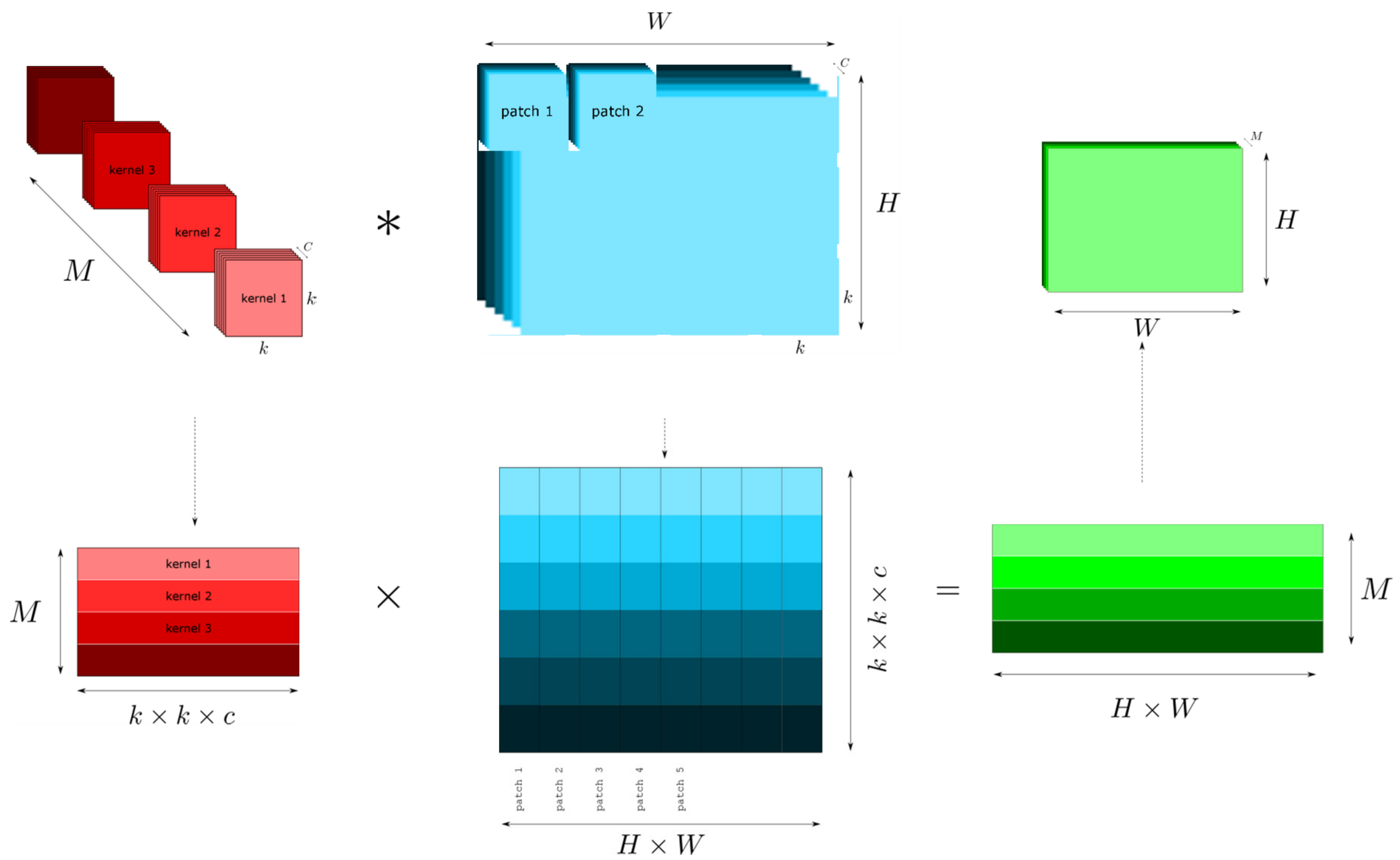
- **Convolution can be implemented as matrix multiplication with a Toeplitz matrix**
  - Decades of work on matrix-matrix multiplication (GEMM)
  - Easy way to quickly get good DNN performance

- **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×H×W
  - Im2col requires additional C×H×W×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*

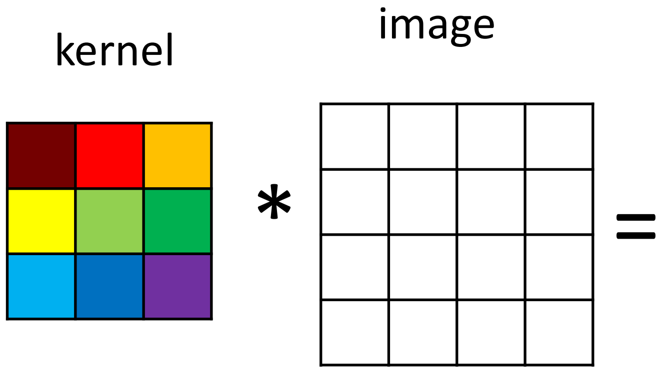


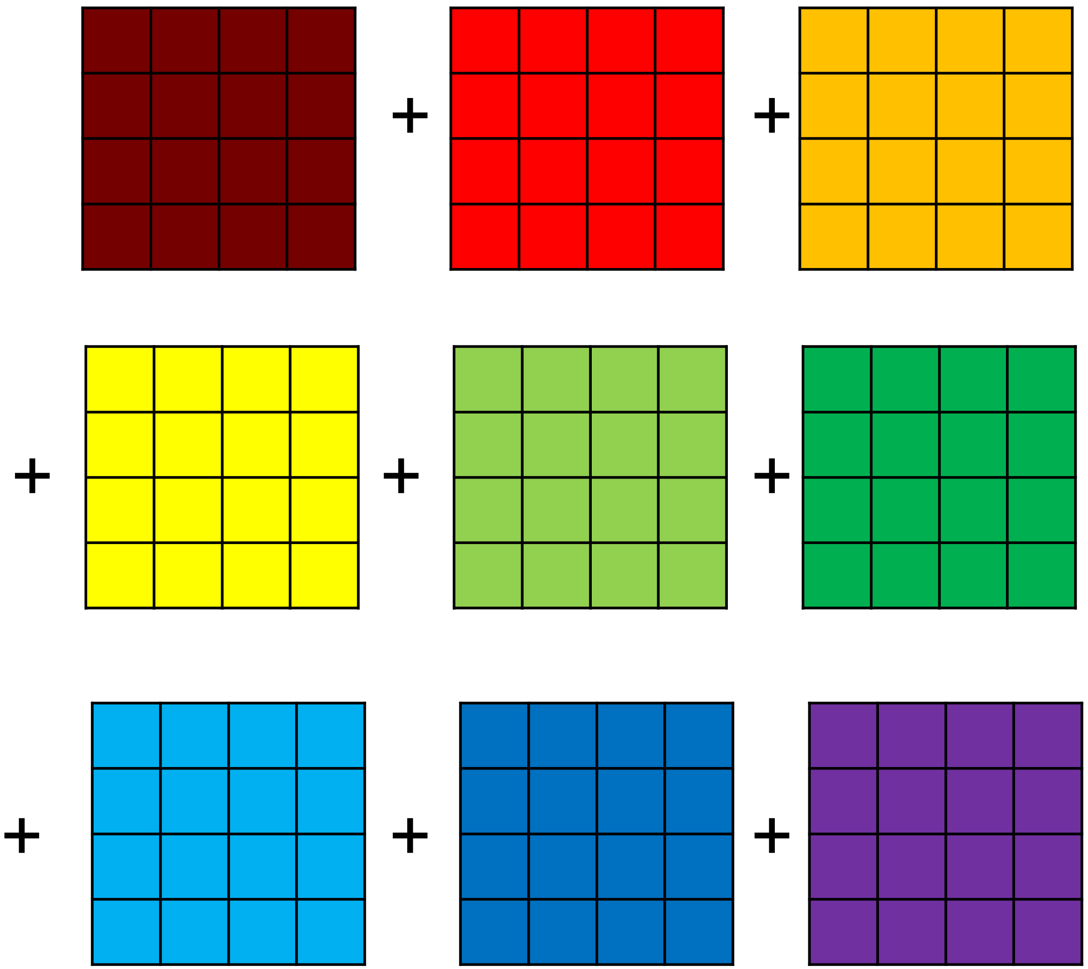
GEMM-based convolution without im2col

- **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**



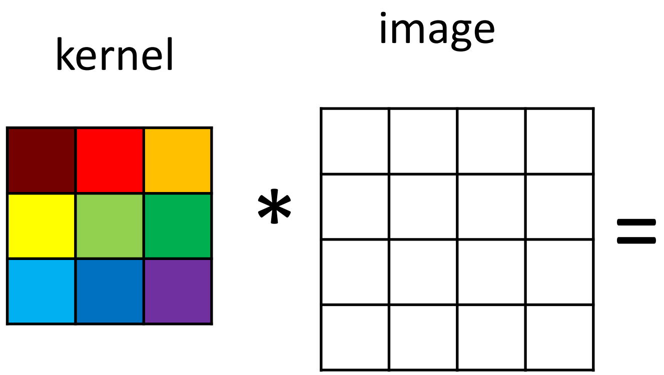


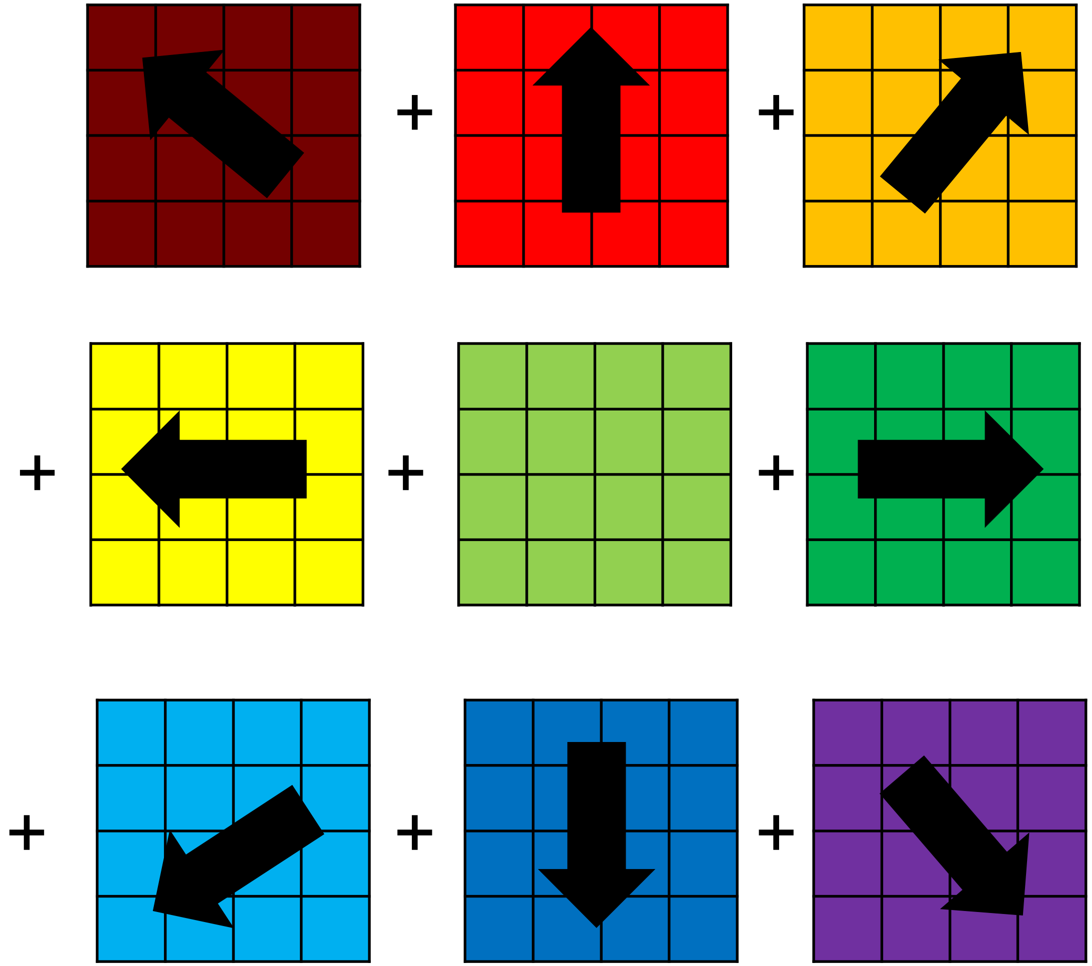
Convolution becomes addition of sub-matrices, each scaled by one element of the kernel.

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 3x3 convolution with one input channel, one convolution kernel**



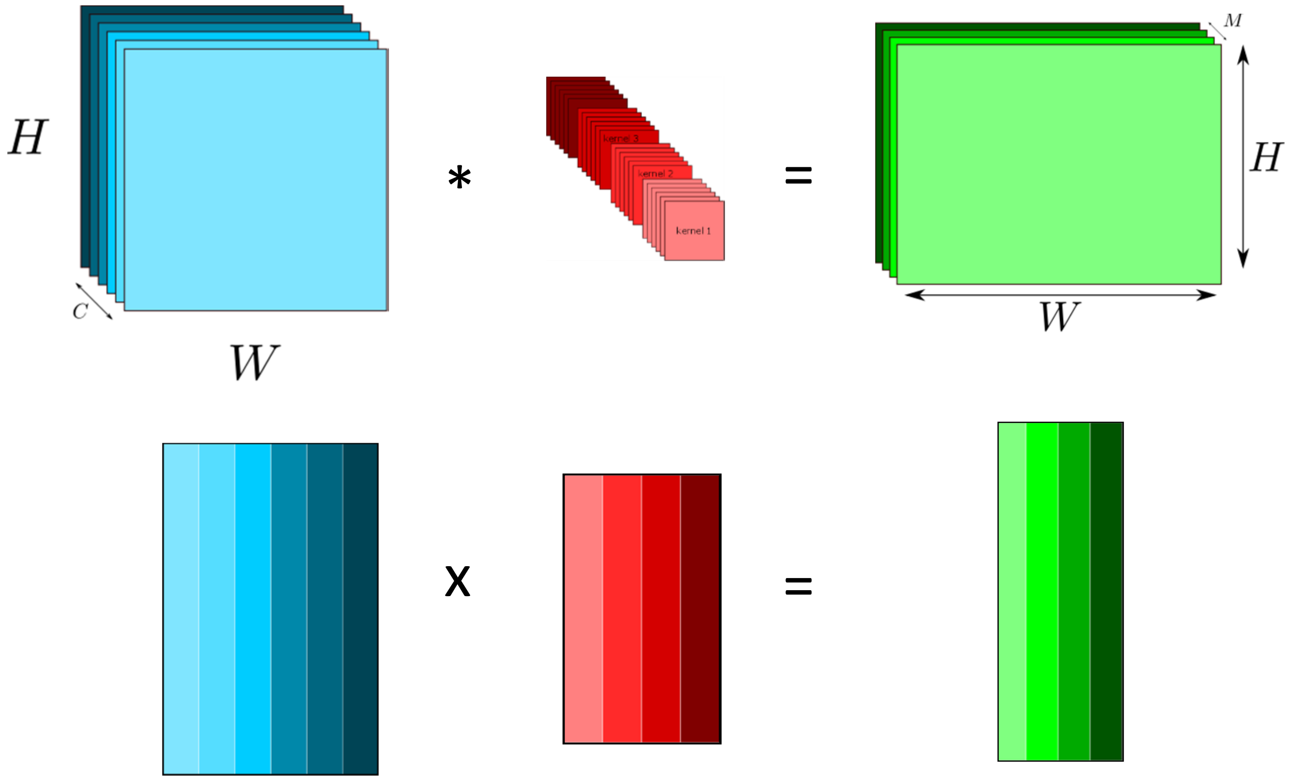


We need to shift the sub-arrays of to line up the values correctly before adding the corresponding values

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

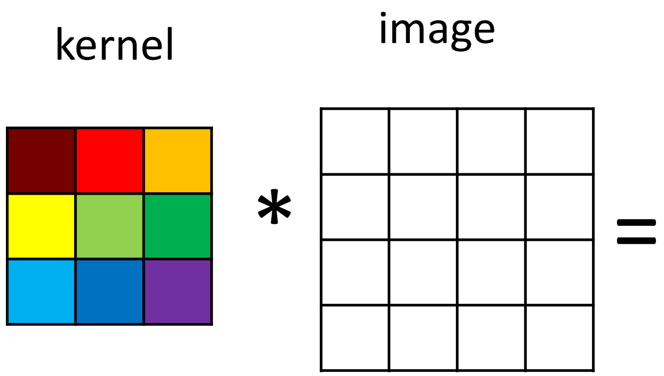
1x1 DNN convolution

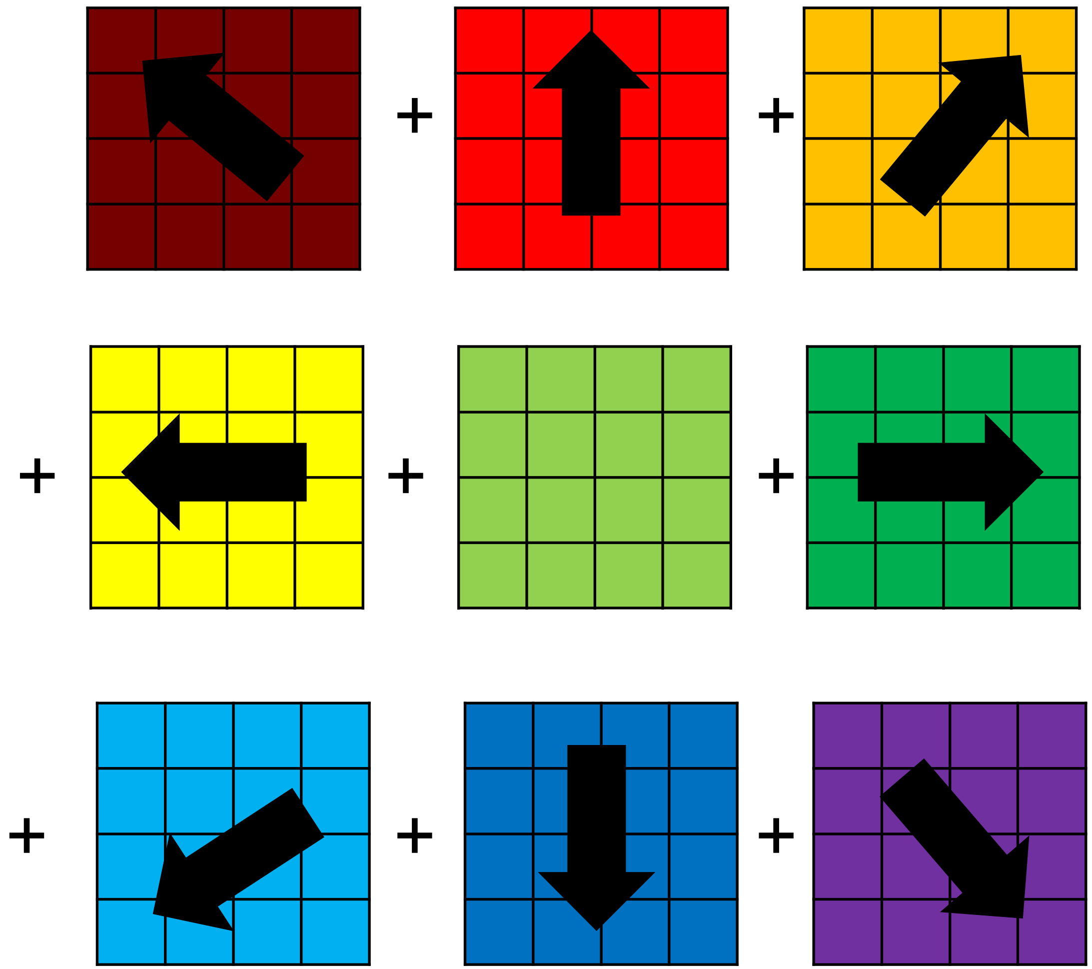


Matrix-Matrix Multiplication (GEMM)

Accumulating Algorithm

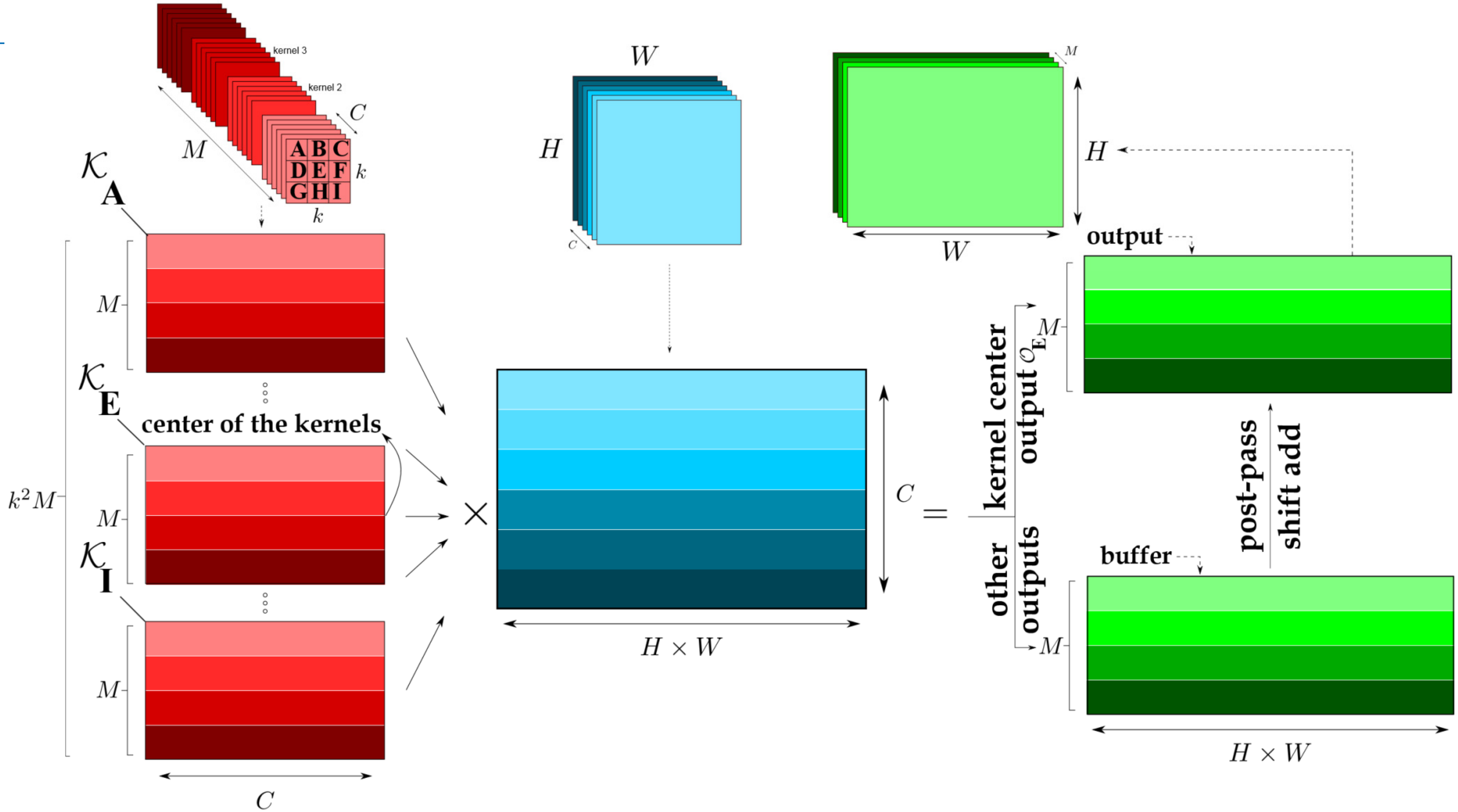
- **Compute one 1x1 convolution at a time and add to output**





- **Temporary space is just one result matrix at a time**
  - CHW space for one result matrix
  - Versus K^2CHW for im2col

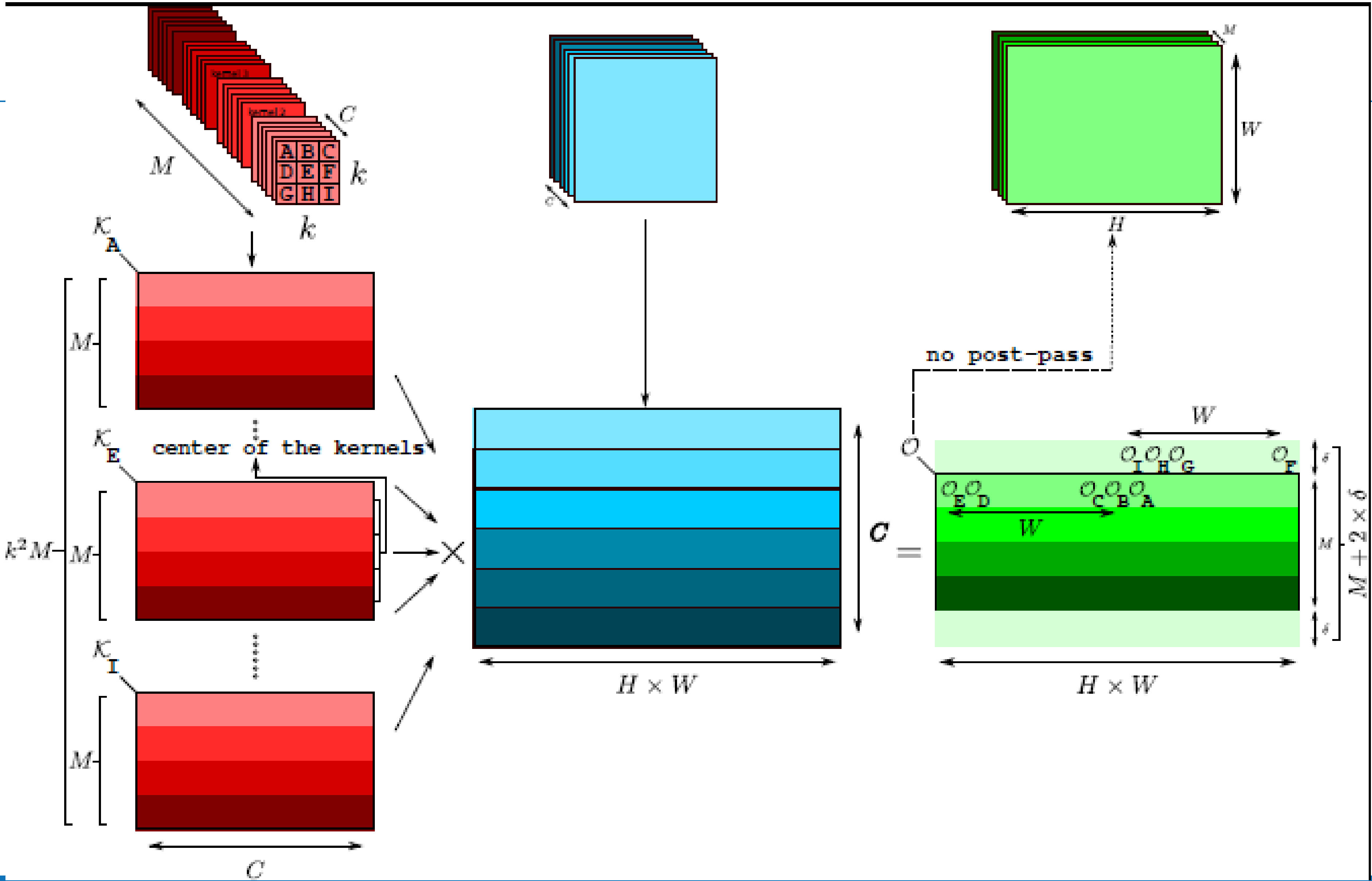
Accumulating Algorithm



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?

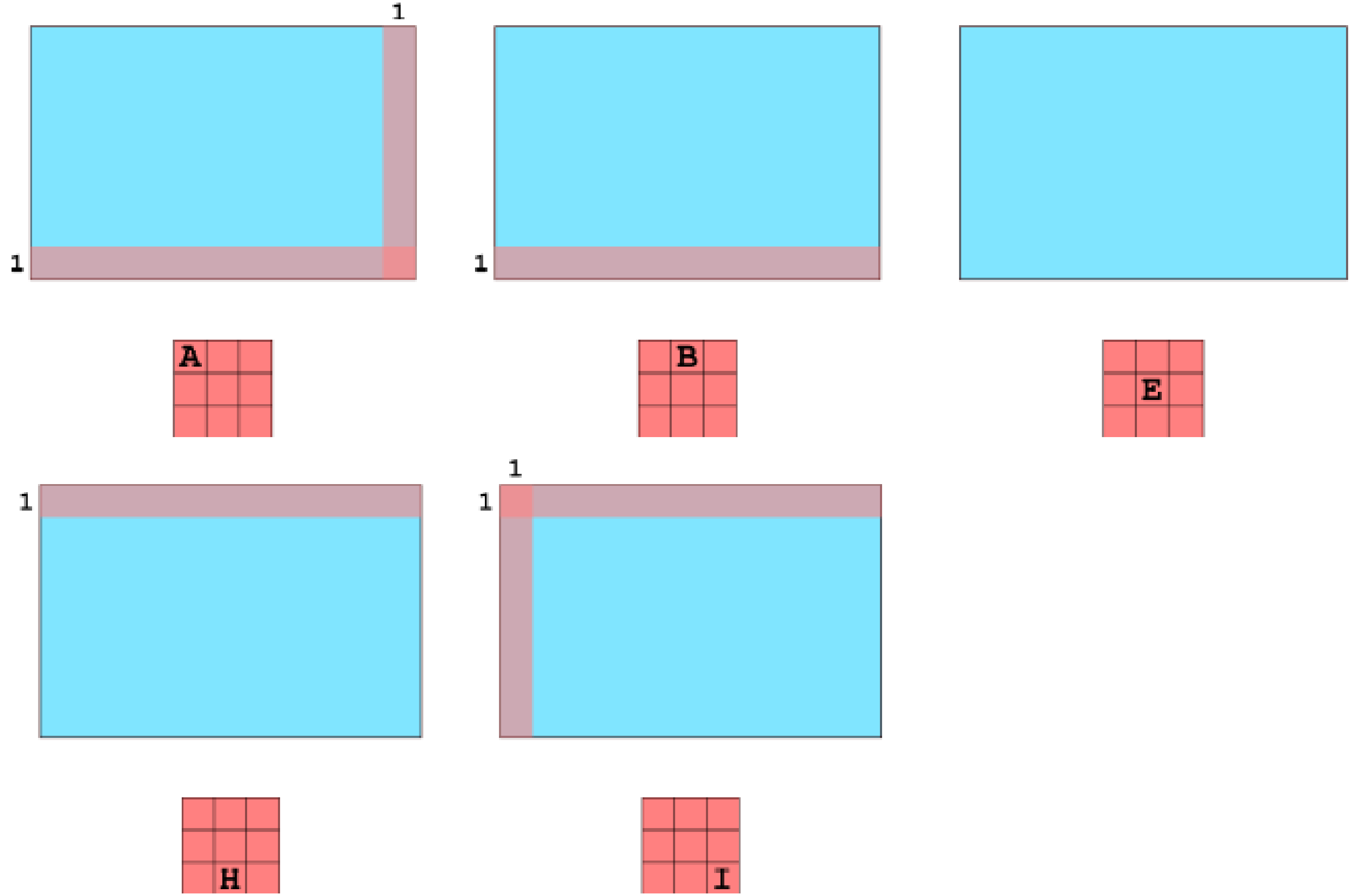
GEMM-accumulating algorithm



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

Algorithm	#GEMM calls	Ops/GEMM call	Extra space
Im2col	1	K^2CHWM	$O(K^2CHW)$
Kernel accumulating	K^2	CHWM	O(CHW)
GEMM accumulating	K^2	CHWM	O(KW+HC+WC)

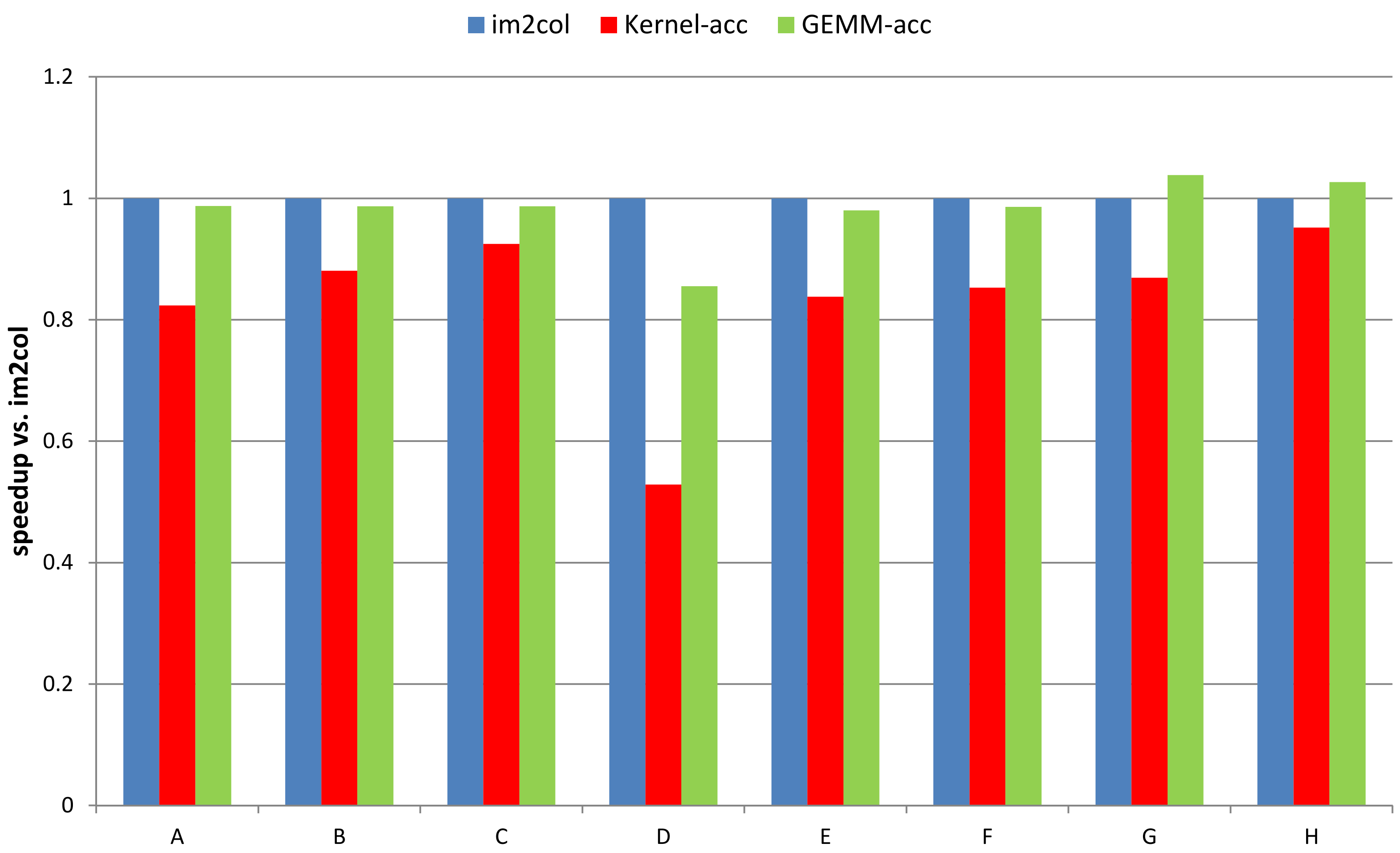
Experiments

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

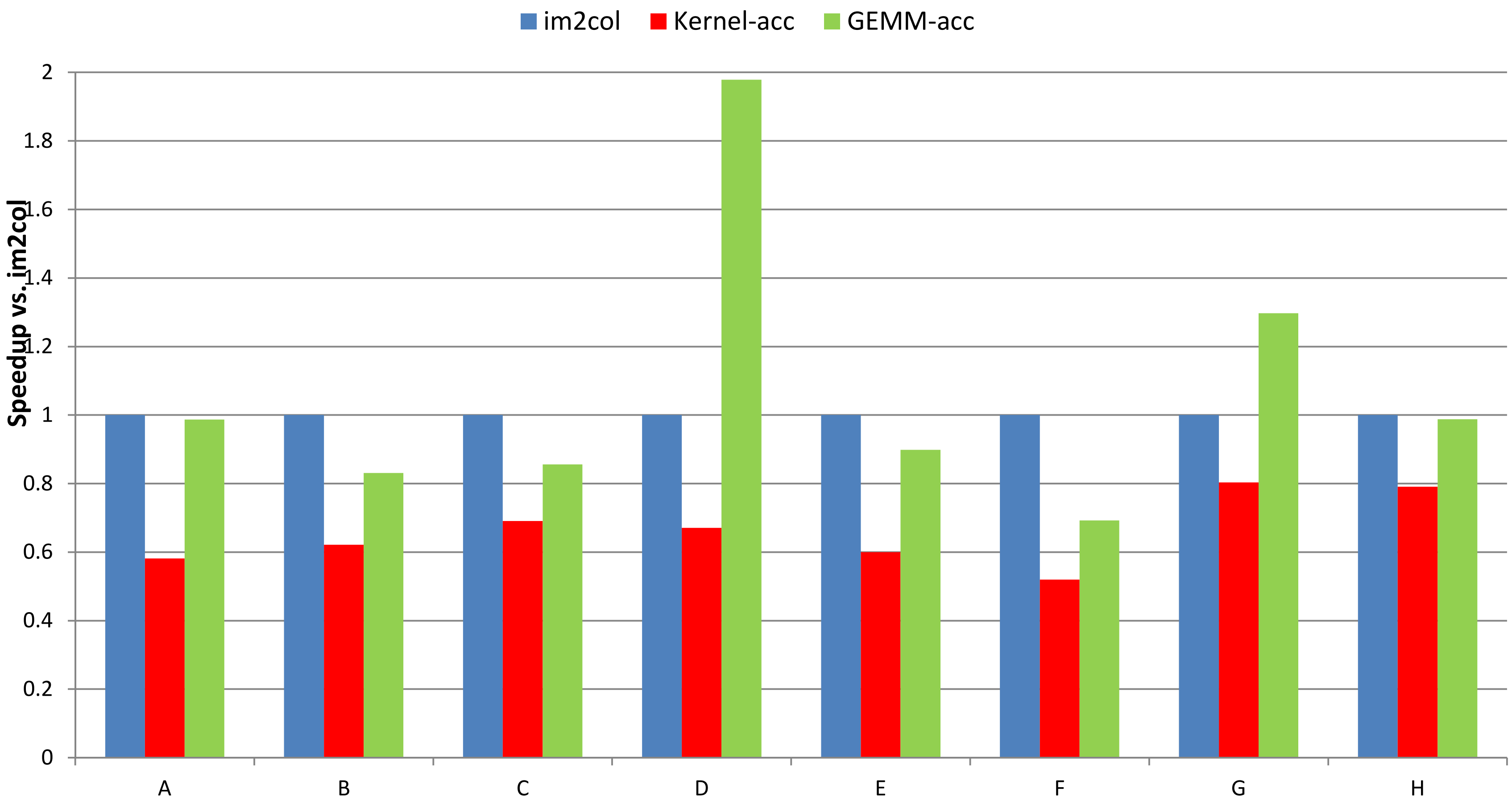
Algorithm	#variants implemented	Variant used
Im2col	23	Row-major, copy short from patch
Kernel accumulating	2	Row-major
GEMM accumulating	4	Row-major AB^T

See paper http://arxiv.org/abs/1709.03395

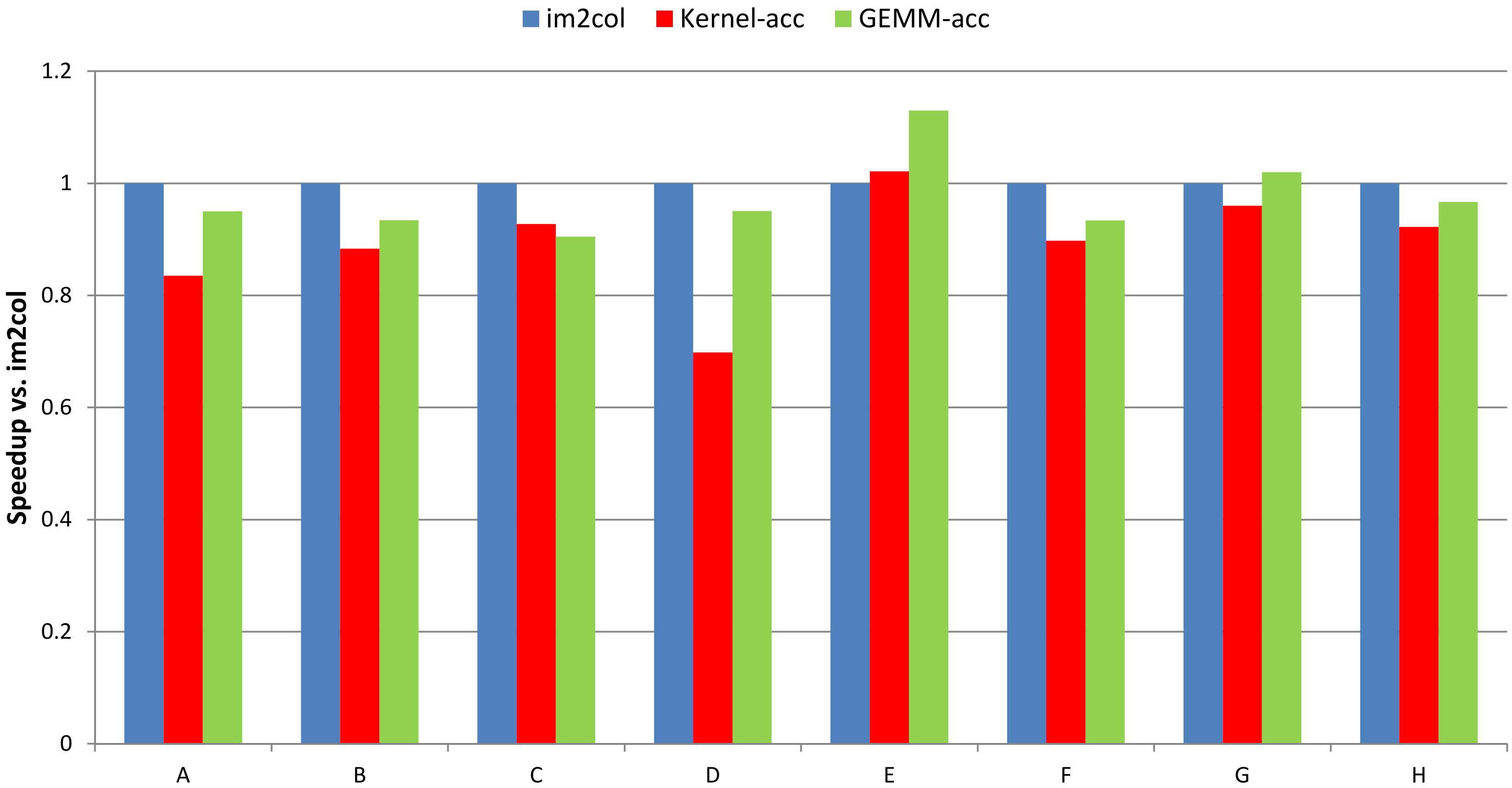
Intel Core i5-4570 single core



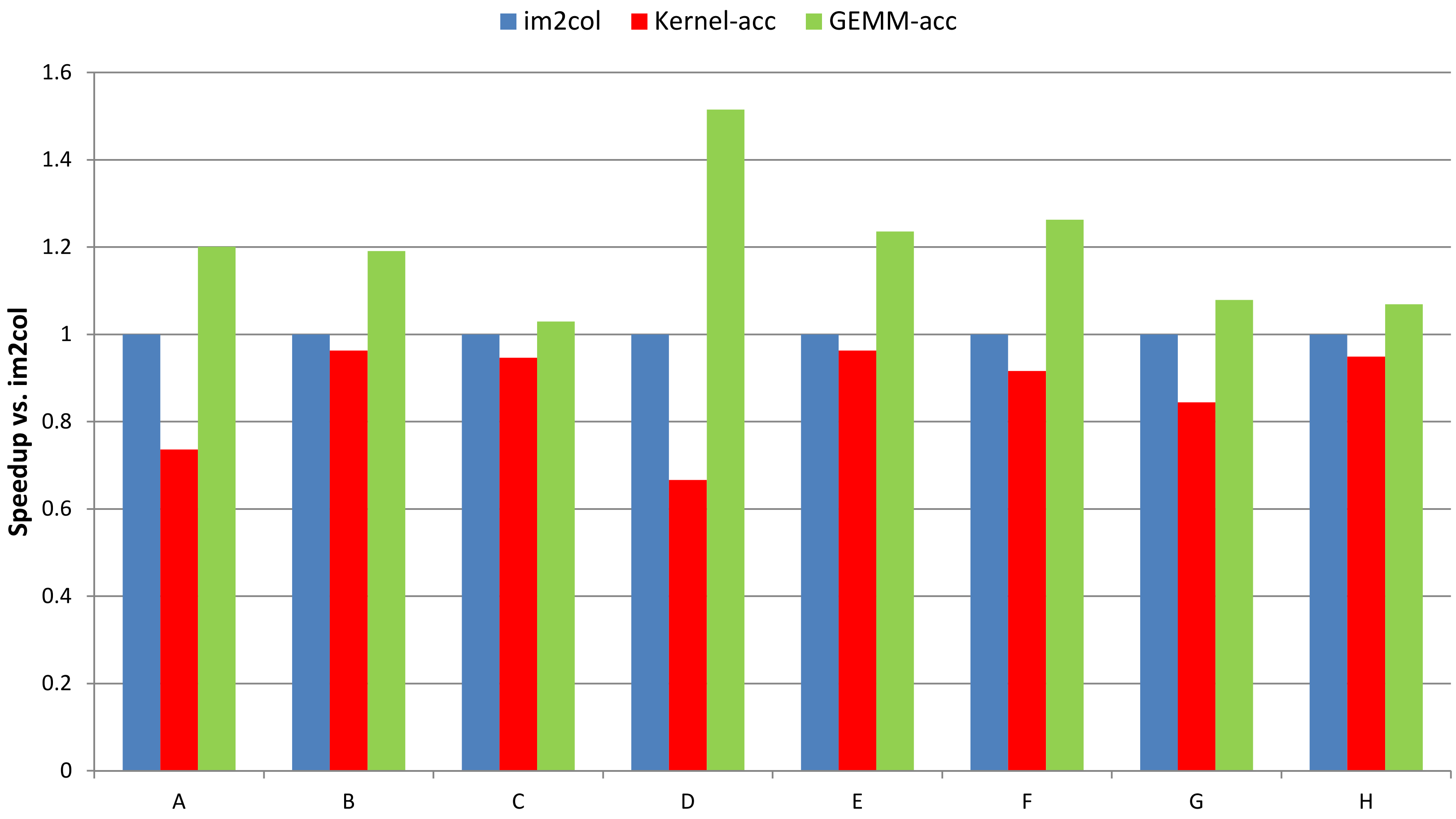
Intel Core i5-4570 multi core



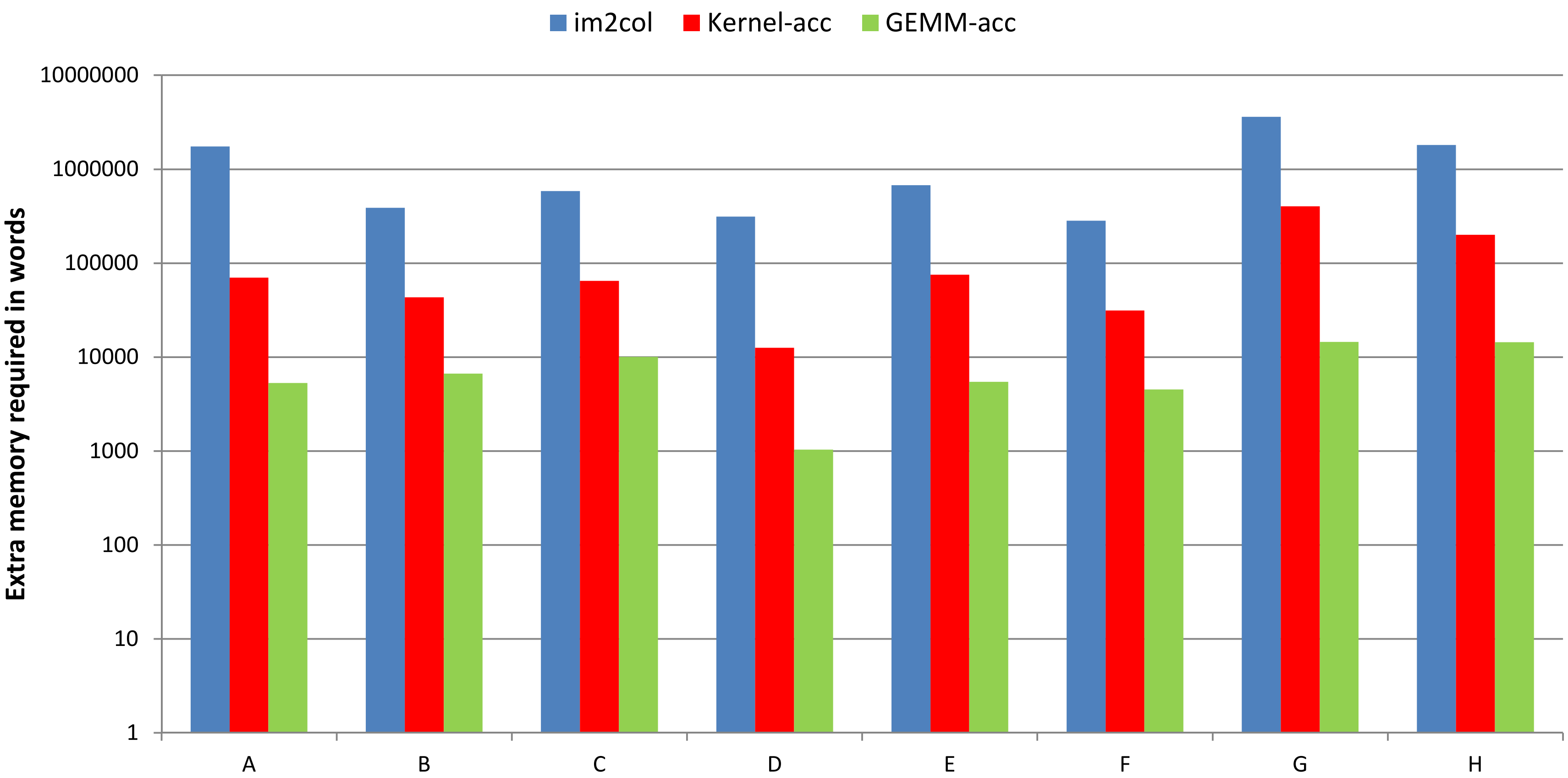
ARM Cortex A57 single core



ARM Cortex A57 multi core



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

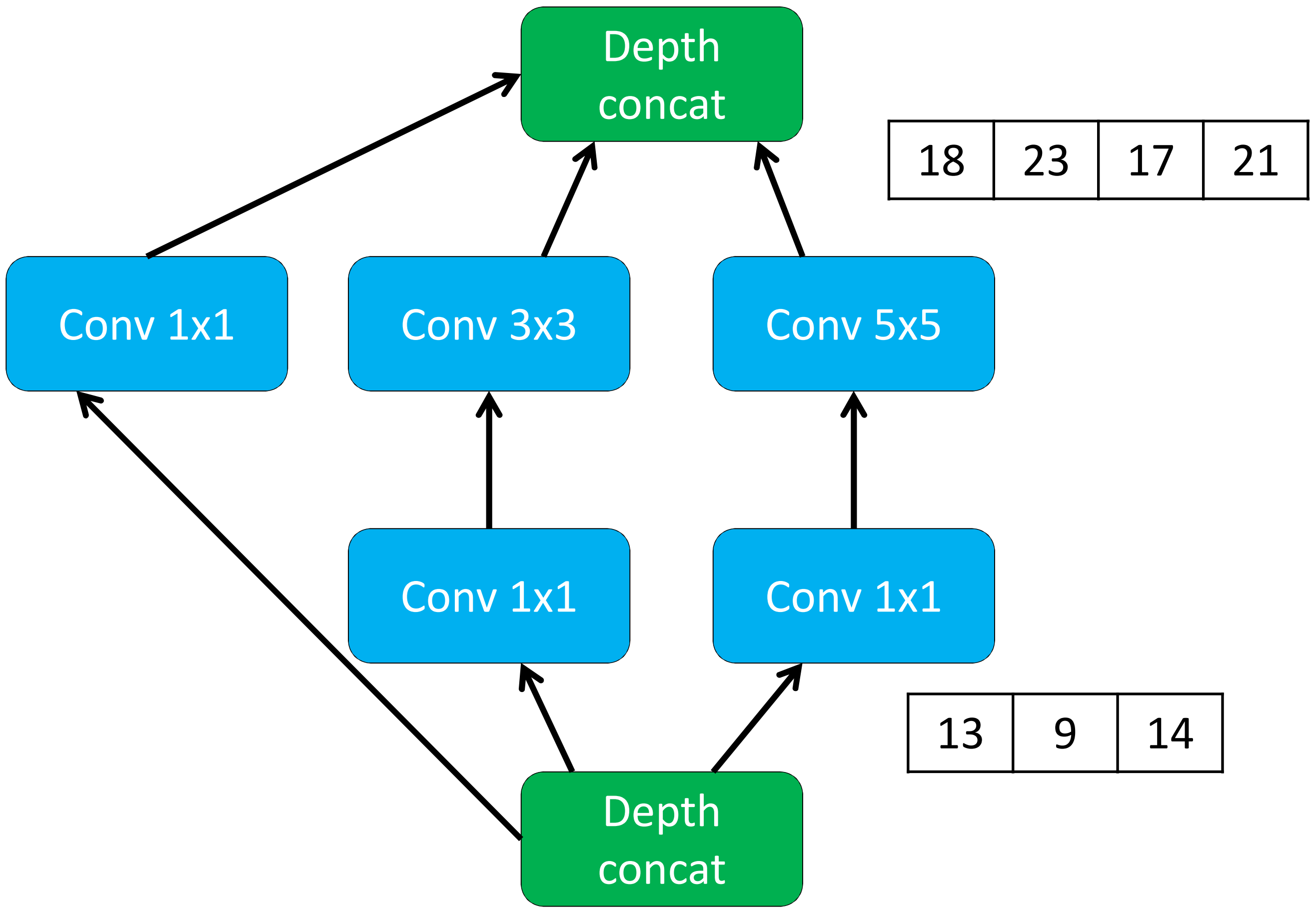
Approach	Good for	Bad for
Simple loop nest	Memory size	Execution time
GEMM – im2col	Good all-rounder; strided	Memory size
GEMM – accumulating	Memory size	Strided ; few channels
MEC algorithm	Strided convolution	Execution time*
FFT convolution	Large kernels	Memory size
Winograd convolution	3x3, 5x5 execution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

* Based on our implementation; the MEC authors have not released their implementation

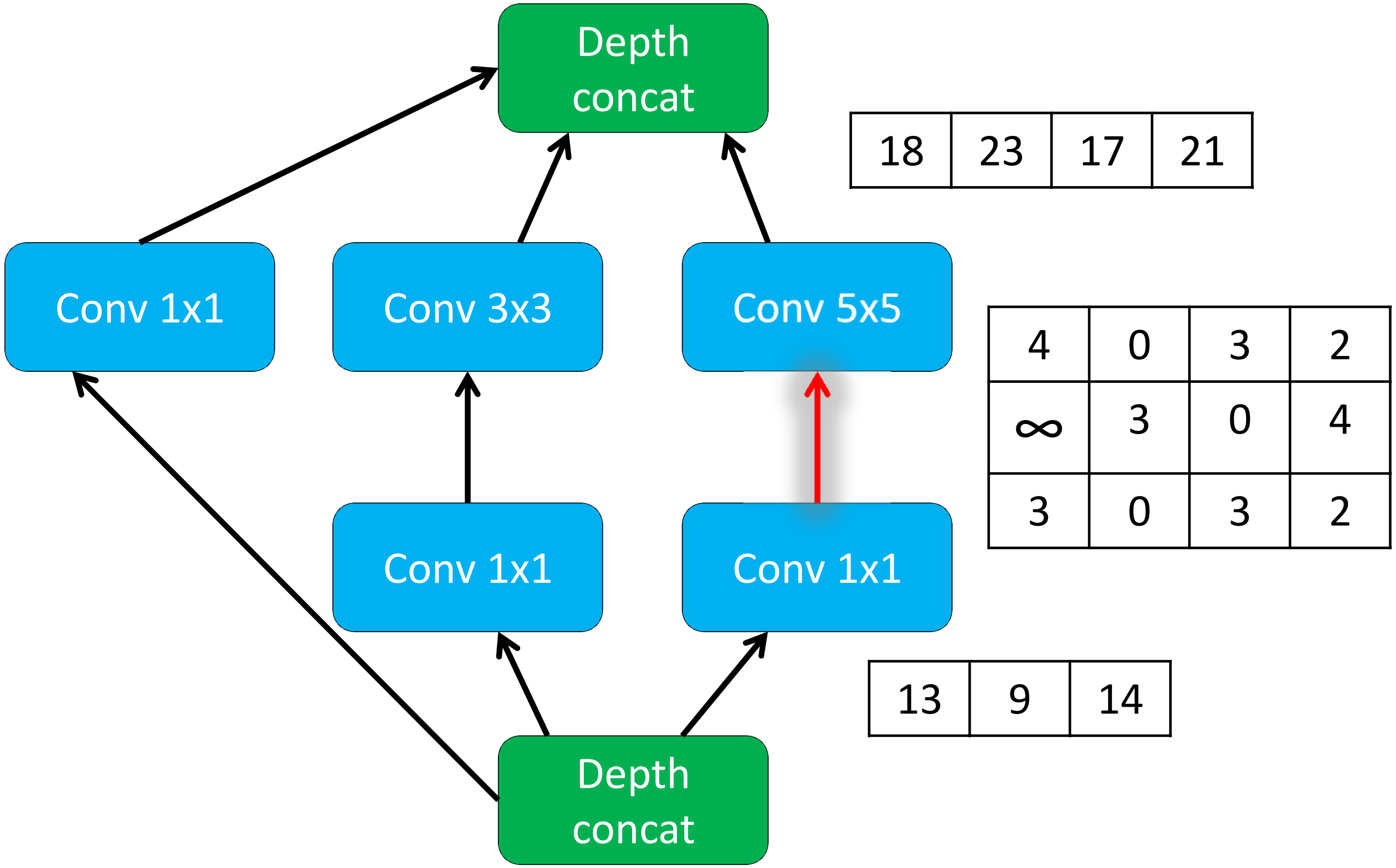
Selecting primitive functions to implement layers

- **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

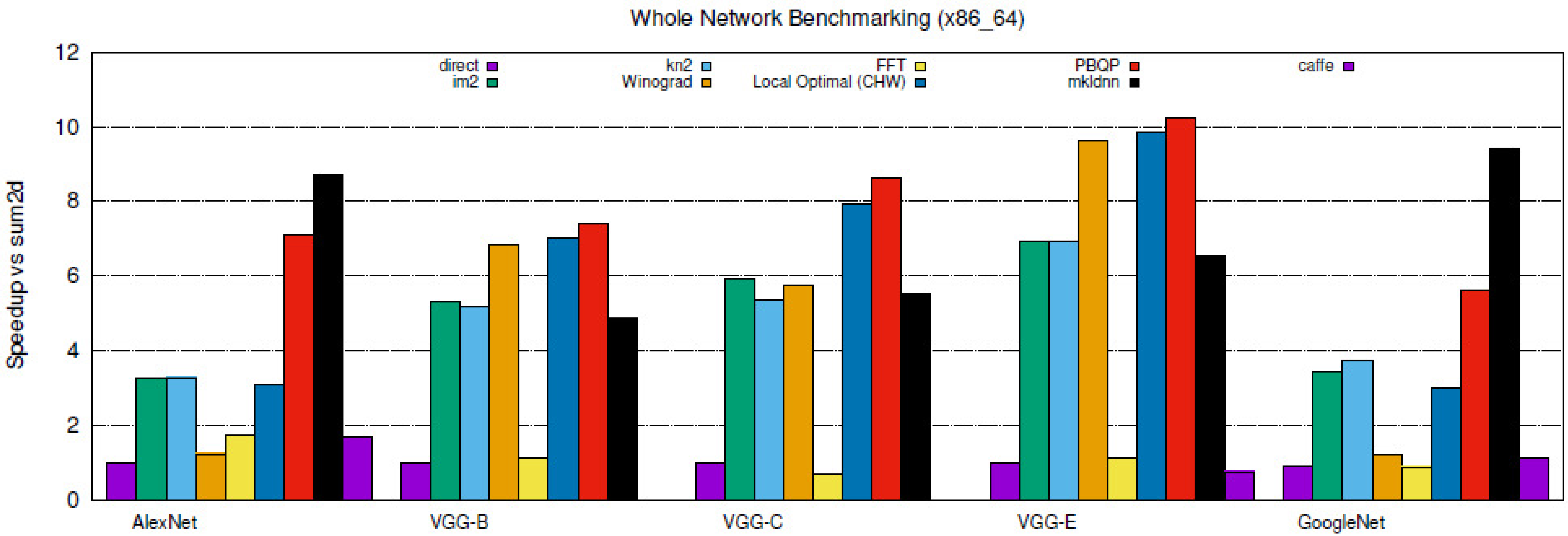
Partitioned Boolean quadratic programming (PBQP)



Partitioned Boolean quadratic programming (PBQP)

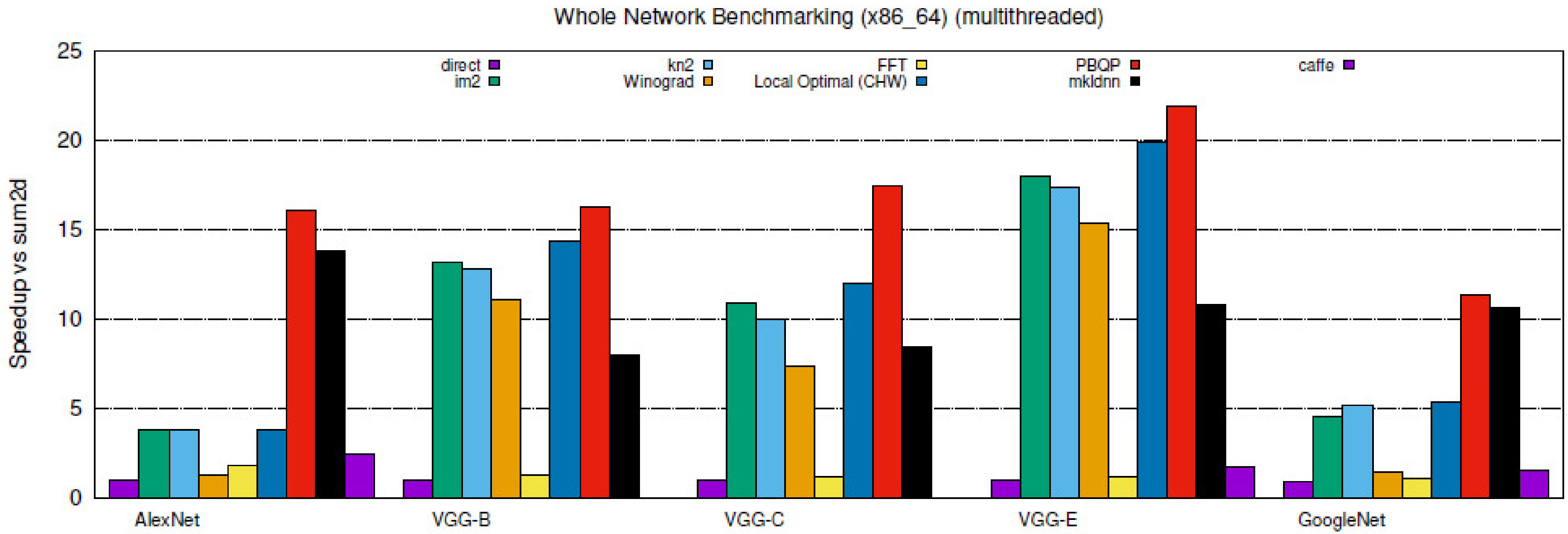


Intel Haswell one core



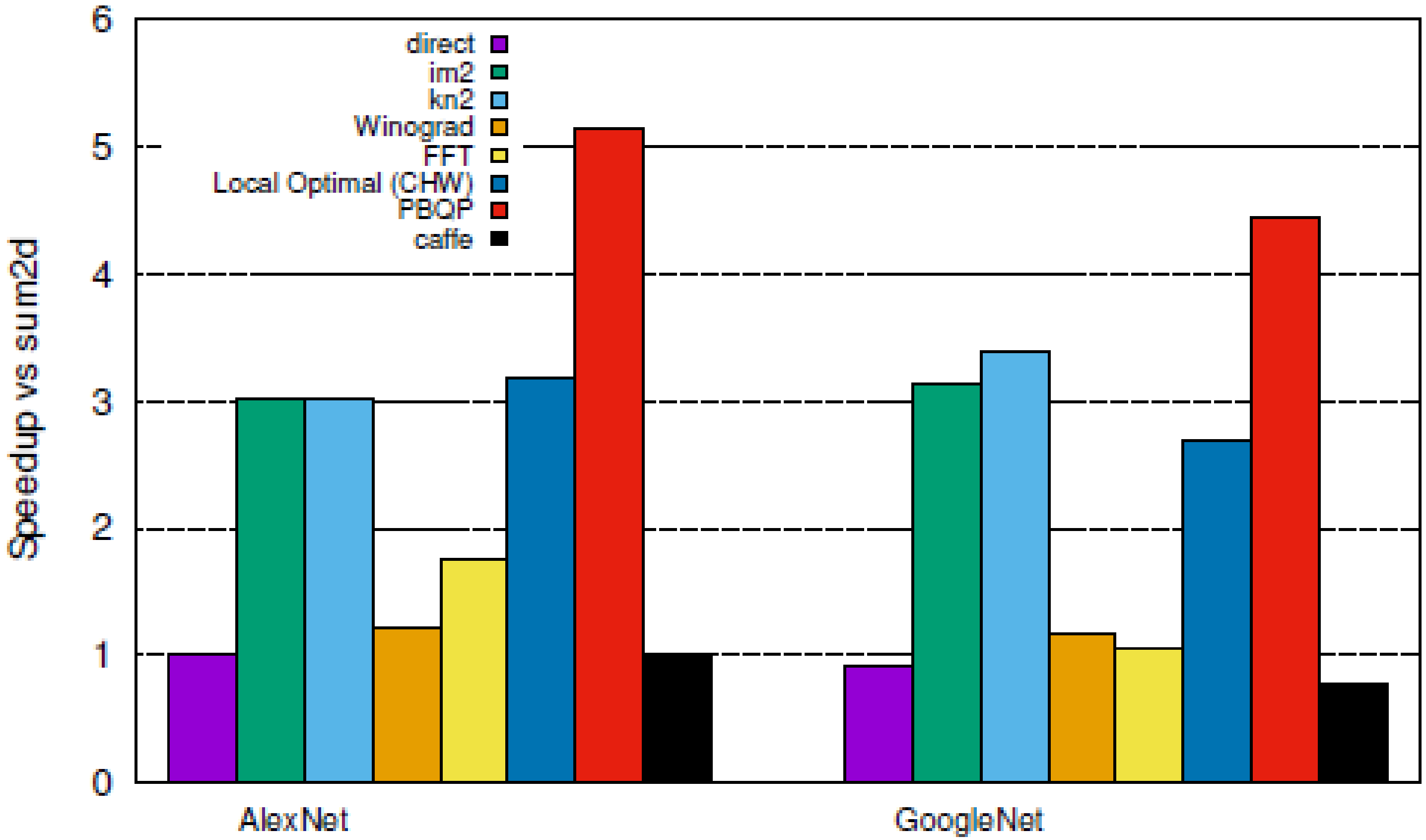
Baseline is simple sum2d algorithm on one core

Intel Haswell multiple cores



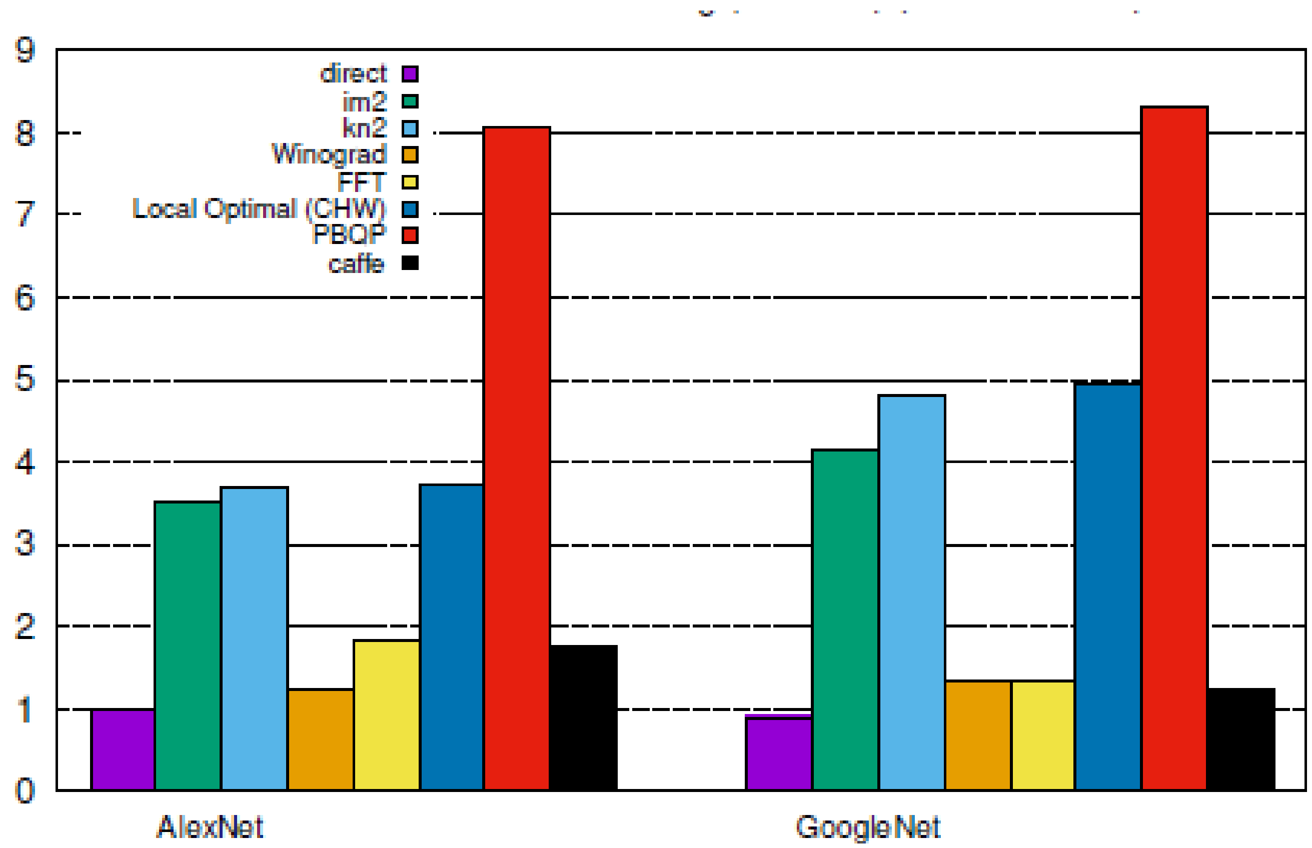
Baseline simple sum2d algorithm on <u>one</u> core

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/**

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Thank You

- A. Anderson, A. Vasudevan, C. Keane and D. Gregg. Low-memory GEMM-based convolution algorithms for deep neural networks. arXiv:1709.03395v1
- A. Anderson and D. Gregg. Optimal DNN primitive selection with partitioned Boolean quadratic programming, CGO 2018.
- 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).