Avoiding Communication in Convolutional Neural Networks

PASC 28 June 2022

Anthony Chen[M], James Demmel[B], Grace Dinh[B], Mason Haberle[N], and Olga Holtz[B]

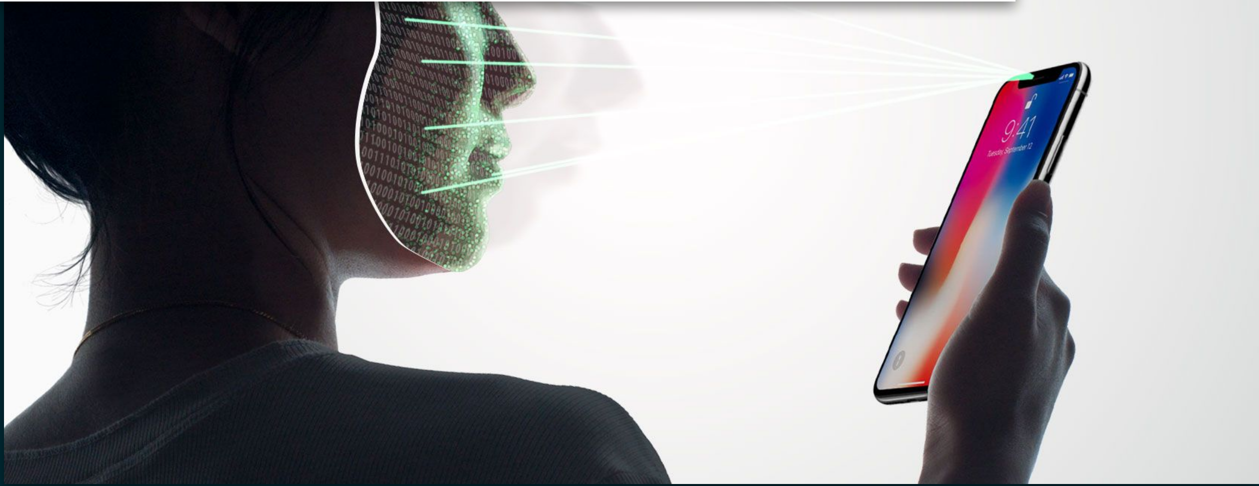
M: University of Michigan
B: University of California, Berkeley
N: New York University

ML Performance is Important

https://uxdesign.cc › apples-faceid-and-magnified-failur...

How a 2-second lag magnifies Apple's FaceID effects of failure

29 Sept 2018 — When **FaceID** fails it doesn't just show the passcode, it waits for what seems like forever (2 full seconds) before it shows you the passcode ...



ML Performance is Important

SIGN IN

The Register®

{* AI + ML *}

AI and ML could save the planet – or add more fuel to the climate fi

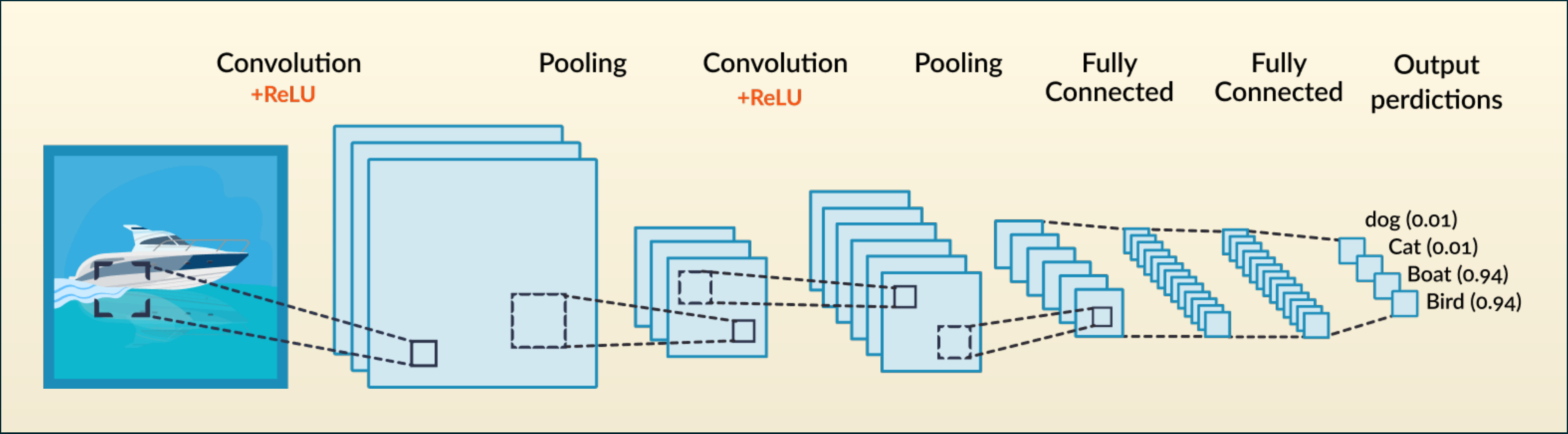
'Staggering amount of computa

A single V100 GPU can consume between 250 and 300 watts. If we assume 250 watts, then 512 V100 GPUS consumes 128,000 watts, or 128 kilowatts (kW). Running for nine days means the MegatronLM's training cost 27,648 kilowatt hours (kWh).

The average household uses 10,649 kWh annually,

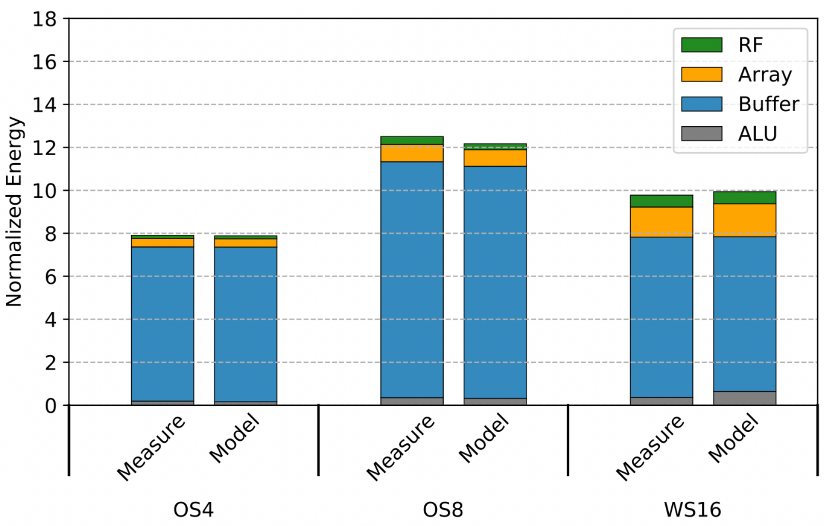
Convolutions: the core of ML

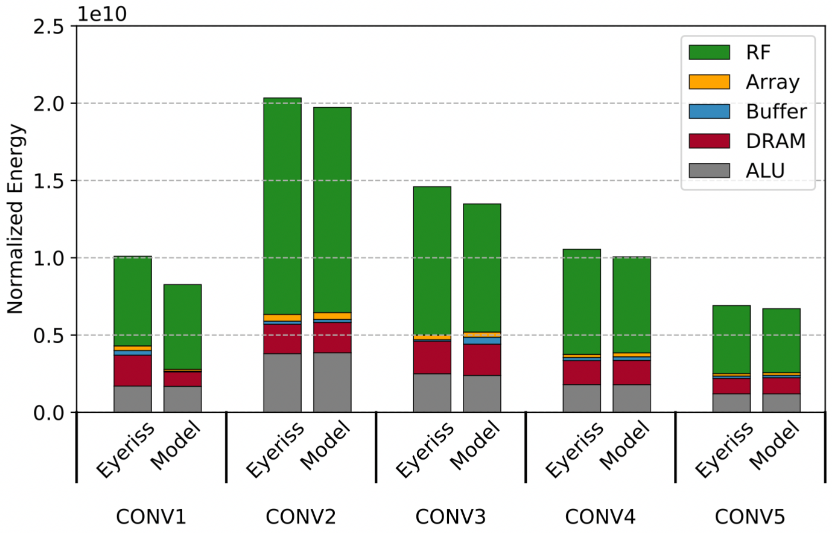
Image Source: Gurung, Paras Mani, *Introduction to Convolutional Neural N*



CNNs: Performance Analysis

Image Source: Yang et al. ASPLOS ’20, *Interstellar: Using Halide's Scheduling Language to Analyz*





Energy consumption of neural net on simulated systolic array, broken into components: RF = Register File data access, Buffer = scratchpad (on-chip memory), Array = systolic array communication, ALU = arithmetic cost. (Yang et al. ASPLOS ’20)

Reduce communication for better performance!

Sample speedups: communication avoidance

Our Approach

Doing the same operation, different order:

- Up to 12x faster for 2.5D dense matmul on 64K core IBM BG/P
- Up to 100x faster for 1.5D sparse-dense matmul on 1536 core Cray XC30
- Up to 6.2x faster for 2.5D All-Pairs-Shortest-Path on 24K core Cray XE6
- Up to 11.8x faster for direct N-body on 32K core IBM BG/P

Mathematically identical answer, but different algorithm

- Up to 13x faster for Tall Skinny QR on Tesla C2050 Fermi NVIDIA GPU
- Up to 6.7x faster for symeig(band A) on 10 core Intel Westmere
- Up to 4.2x faster for BiCGStab (MiniGMG bottom solver) on 24K core Cray XE6
- Up to 5.1x faster for coordinate descent LASSO on 3K core Cray XC30

Different algorithm, different approximate answer

- Up to 16x faster for SVM on a 1536 core Cray XC30
- Up to 135x faster for ImageNet training on 2K Intel KNL nodes

The Communication Model

Processor
(CPU, accelerator, etc.)

memory
(arbitrarily large)

The Communication Model

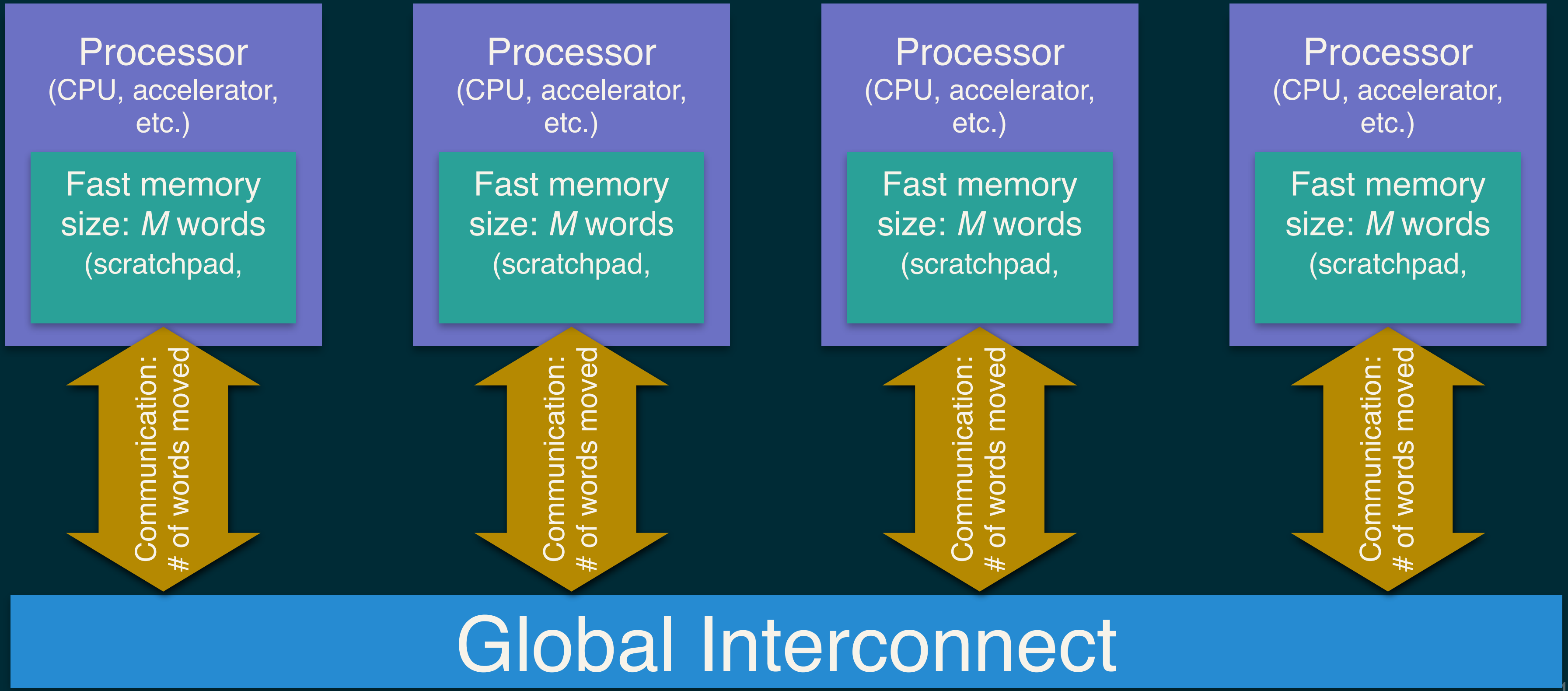
Processor
(CPU, accelerator, etc.)

Slow memory
(arbitrarily large)

The Communication Model

The Communication Model

The Communication Model: Parallel Edition



How do we reduce communication?

Tiling

Split tensors into blocks to reduce communication costs.

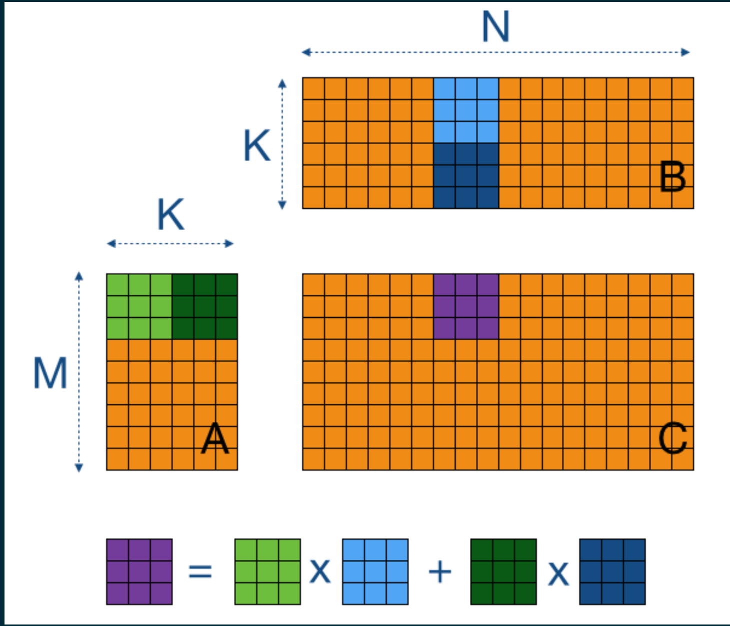
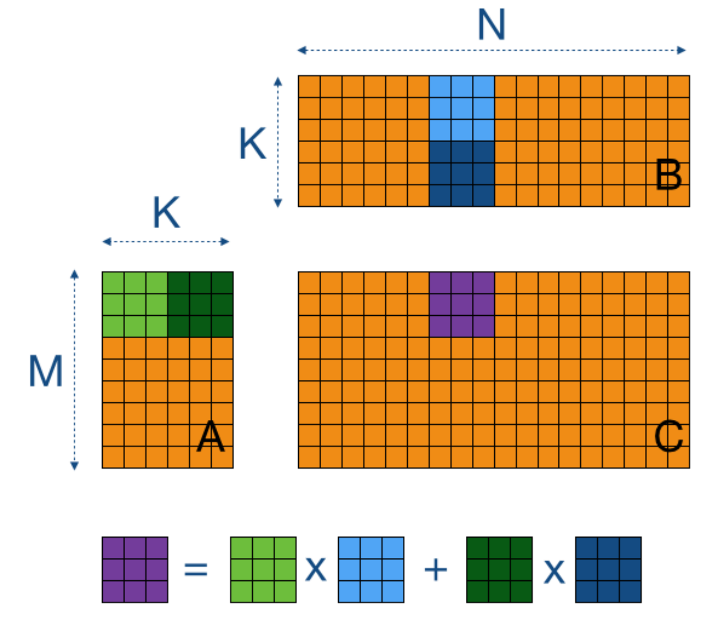


Image Source: Nugteren, Cedric, *OpenCL SGEMM tuning for Kepler,* 2014

Tiling

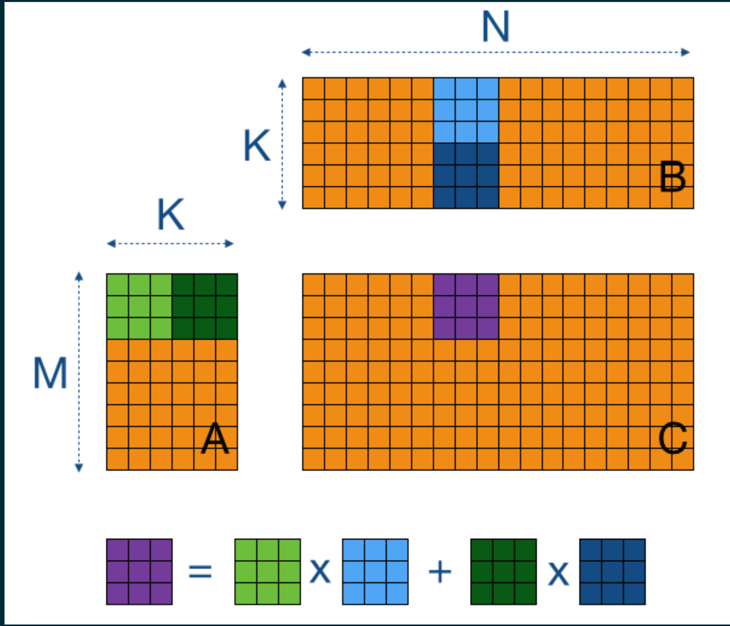
Split tensors into blocks to reduce communication costs.




```
for m in [0..M):
  for k in [0..K):
    for n in [0..N):
      C[m,n] += A[m,k]B[k,n]
```

Tiling

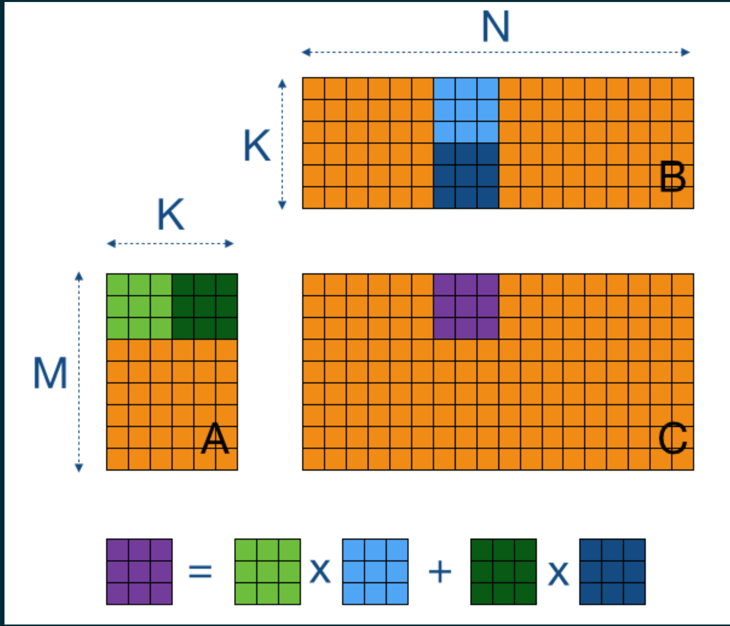
Split tensors into blocks to reduce communication costs.




```
for m_out in [0..M/B_m):
  for m_in in [0..B_m):
    for k in [0..K):
      for n in [0..N):
        C[m,n] += A[m,k]B[k,n]
```

Tiling

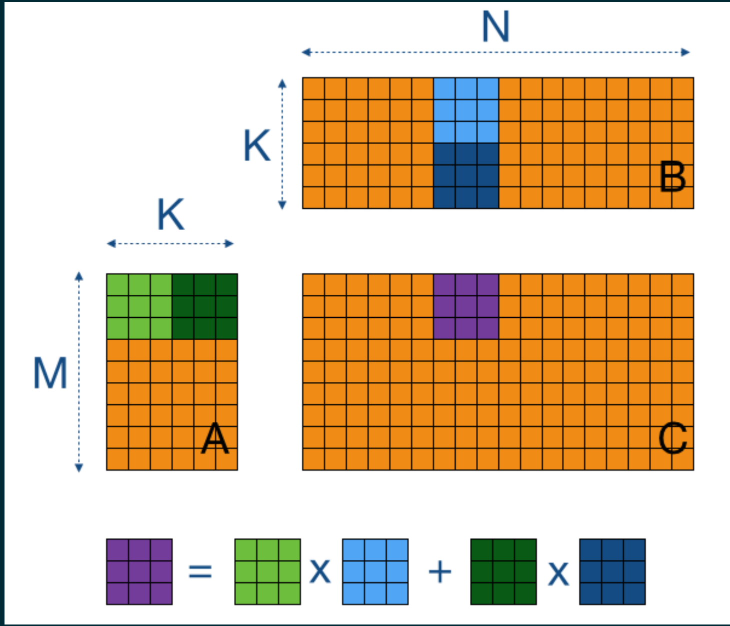
Split tensors into blocks to reduce communication costs.



for m_{out} in [0..M/B_m):
  for m_{in} in [0..B_m):
    for k in [0..K):
      for n in [0..N):
        $m = m_{out} B_m + m_{in}$
        C[m,n] += A[m,k]B[k,n]

Tiling

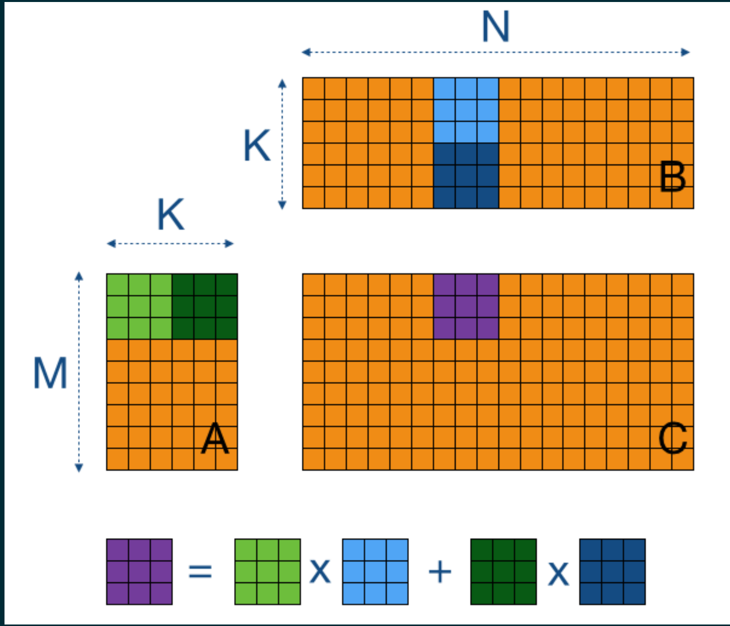
Split tensors into blocks to reduce communication costs.




```
for m_out in [0..M/B_m):
  for k in [0..K):
    for n in [0..N):
      for m_in in [0..B_m):
        m = m_out B_m + m_in
        C[m,n] += A[m,k]B[k,n]
```

Tiling

Split tensors into blocks to reduce communication costs.



$\text{for } m_{out} \text{ in } [0..M/B_m):$

$\quad \text{for } k_{out} \text{ in } [0..M/B_k):$

$\quad\quad \text{for } n \text{ in } [0..N):$

$\quad\quad\quad \text{for } m_{in} \text{ in } [0..B_m):$

$\quad\quad\quad\quad \text{for } k_{in} \text{ in } [0..B_k):$

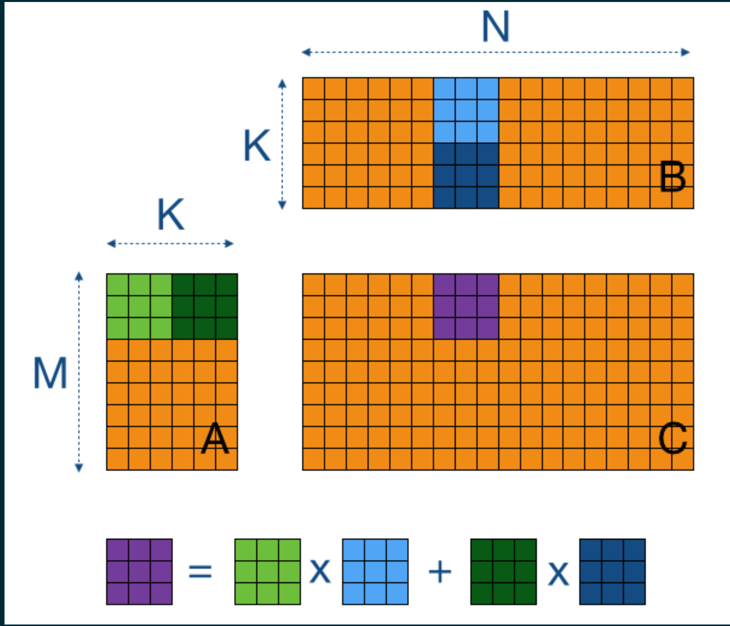
$\quad\quad\quad\quad\quad m = m_{out}B_m + m_{in}$

$\quad\quad\quad\quad\quad k = k_{out}B_k + k_{in}$

$\quad\quad\quad\quad\quad C[m,n] \mathrel{+}= A[m,k]B[k,n]$

Tiling

Split tensors into blocks to reduce communication costs.

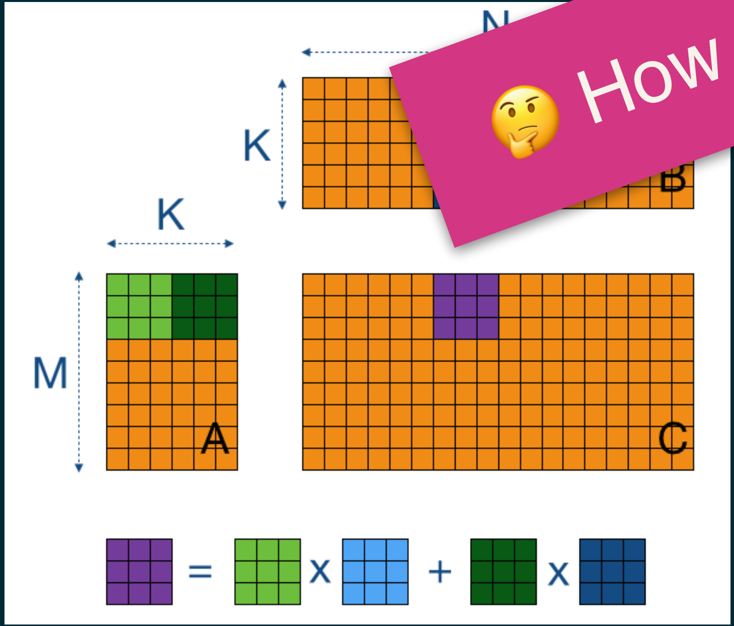



```
for m_out in [0..M/B_m):
 for k_out in [0..K/B_k):
  for n_out in [0..N/B_n):
   for m_in in [0..B_m):
    for k_in in [0..B_k):
     for n_in in [0..B_n):
      m = m_out B_m + m_in
      k = k_out B_k + k_in
      n = n_out B_n + n_in
      C[m,n] += A[m,k]B[k,n]
```

Tiling

Split tensors into blocks to reduce …

🤔 How do we find good tiles




```
… [0..M/B_m):
… k_out in [0..K/B_k):
  for n_out in [0..N/B_n):
    for m_in in [0..B_m):
      for k_in in [0..B_k):
        for n_in in [0..B_n):
          m = m_out B_m + m_in
          k = k_out B_k + k_in
          n = n_out B_n + n_in
          C[m,n] += A[m,k]B[k,n]
```

Finding Tilings…

…is difficult.

Results often for either specific hardware models (e.g. Huang et al. ISCA ’21, needs cost function + HW params), or reliant on expensive per-architecture autotuning (e.g. ATLAS).

Previous theoretical methods prove and attain *asymptotic* lower bounds but are difficult to apply in practice

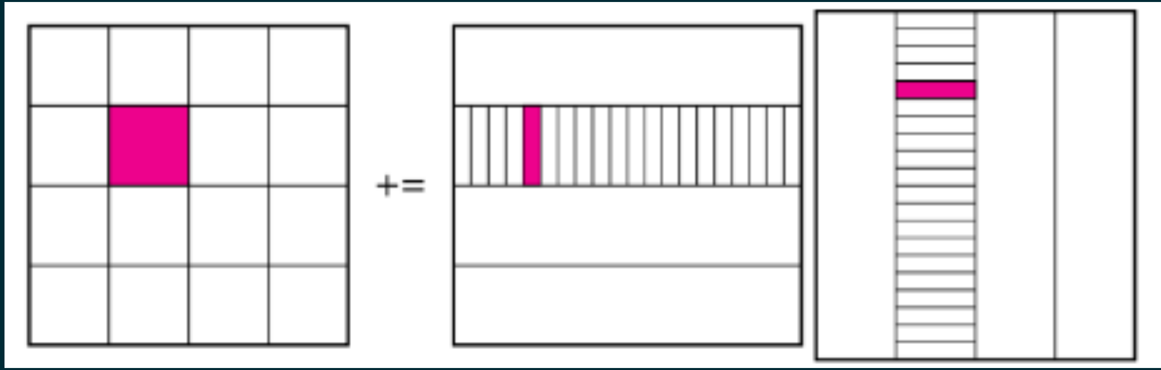


Image source: T. Smith et al. (Feb 2019). *A Tight I/O Lower Bound for Matrix Multiplication*

Example: cubic tiles are asymptotically optimal for matmul, but rectangular tiles give a better constant.

For Convolutions…

Even more complicated loop nest to tile: 7-D, now with scaling factors!

batch size | input channels | output channels | output dimensions | filter dimensions

```
for (b, c, k, w, h, r, s) in range(0,{B, C, K, W, H, R, S}):
  Out[k, h, w, b] += Image[r+wσ_r, s+hσ_h, c, b]* Filter[k, r, s, c]
```

strides

Intuition: maximize tile size with optimization

As an optimization problem…

maximize the tile size: $b_b b_c b_k b_w b_h b'_r b''_r b'_s b''_s$ subj. to:

$$b_c b_k b'_r b''_r b'_s b''_s \;+\; b_b b_k b_w b_h \;+\; b_b b_c (b_w + b'_r)(b_h + b'_s) b''_r b''_s \leq M$$

(plus constraints on tile sizes being smaller than program sizes and some extra technical constraints)

Relax the problem: assume equal scratchpad sizes for each tensor tile:

maximize the tile size: $b_b b_c b_k b_w b_h b'_r b''_r b'_s b''_s$ subj. to:

$$b_c b_k b'_r b''_r b'_s b''_s \; + \; b_b b_k b_w b_h \; + \; b_b b_c (b_w + b'_r)(b_h + b'_s) b''_r b''_s \; \leq M$$

Relax the problem: assume equal scratchpad sizes for each tensor tile:

maximize the tile size: $b_b b_c b_k b_w b_h b'_r b''_r b'_s b''_s$ subj. to:

$$b_c b_k b'_r b''_r b'_s b''_s \leq \frac{M}{3}$$

$$b_b b_k b_w b_h \leq \frac{M}{3}$$

$$b_b b_c (b_w + b'_r)(b_h + b'_s) b''_r b''_s \leq \frac{M}{3}$$

Constrain block sizes ≤

nonlinear 😭

maximize the tile size: $b_b b_c b_k b_w b_h b'_r b''_r b'_s b''_s$ subj. to:

$$b_c b_k b'_r b''_r b'_s b''_s \leq \frac{M}{3}$$

$$b_b b_k b_w b_h \leq \frac{M}{3}$$

$$b_b b_c (b_w + b'_r)(b_h + b'_s) b''_r b''_s \leq \frac{M}{3}$$

maximize the tile size: $\log_M(b_b b_c b_k b_w b_h b'_r b''_r b'_s b''_s)$ subj. to:

$$\log_M(b_c b_k b'_r b''_r b'_s b''_s) \le \log_M \frac{M}{3}$$

$$\log_M(b_b b_k b_w b_h) \le \log_M \frac{M}{3}$$

$$\log_M(b_b b_c (b_w + b'_r)(b_h + b'_s) b''_r b''_s) \le \log_M \frac{M}{3}$$

"almost"

Multiply out and relax again by replacing constraint with *M/12* ≥ each of 4 product terms individually maximize $c^T x$ subj. to $Ax \leq L$, where

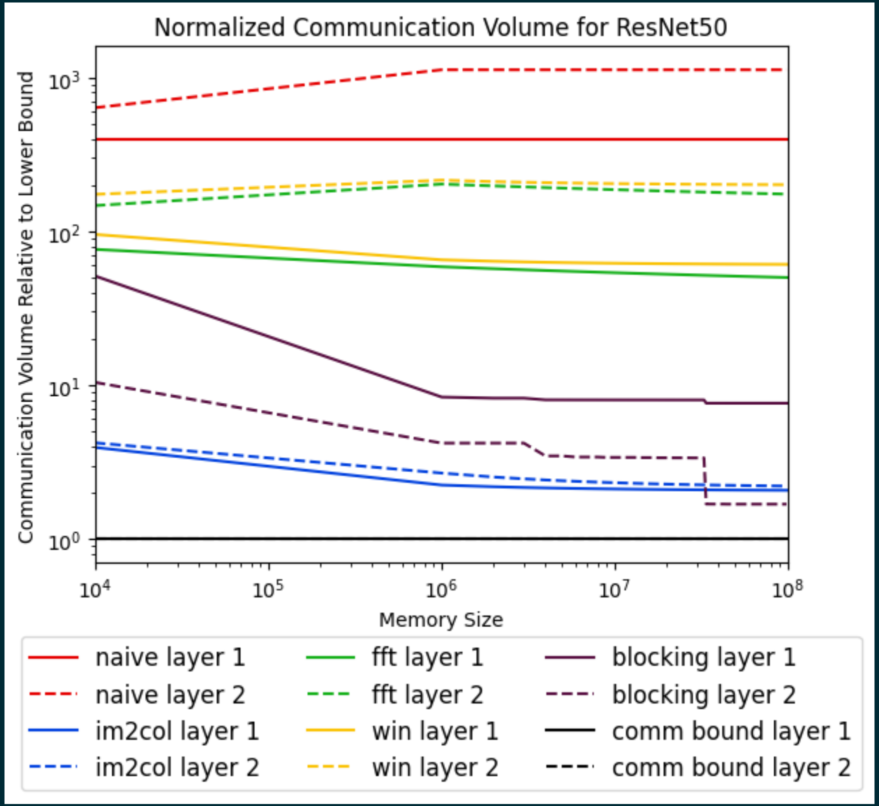
$$c^T = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$b^T = 1 - \begin{bmatrix} \log_M 3 & \log_M 3 & \log_M 12 & \log_M 12 & \log_M 12 & \log_M 12 \end{bmatrix}$$

How good are these tilings?

Theoretical comparison:



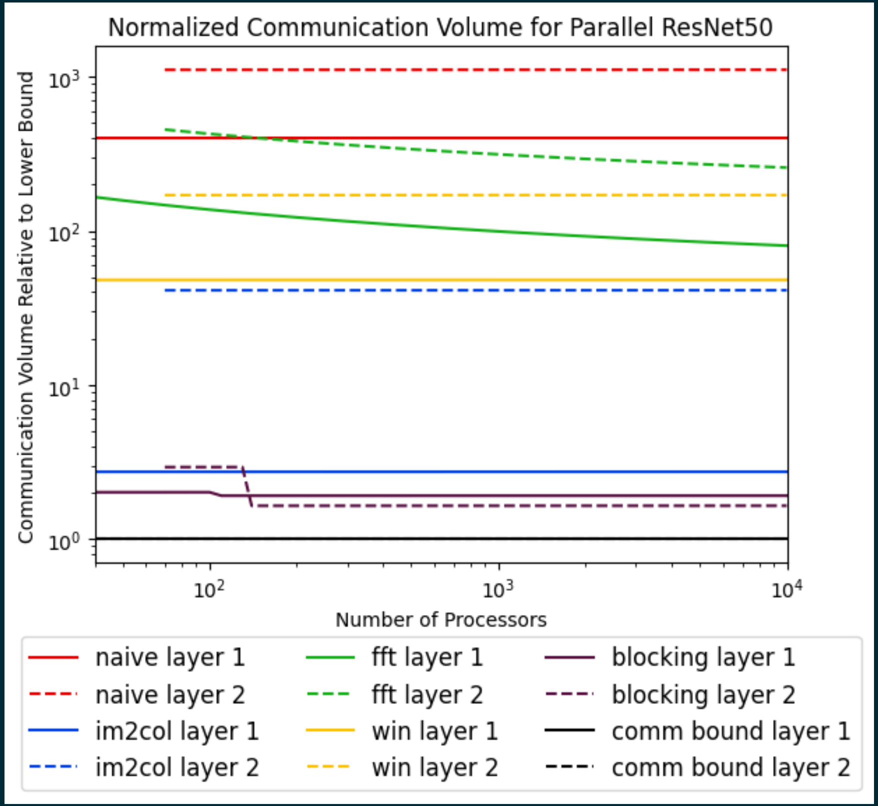
Theoretical analysis: tiled direct convolution requires less computation than:

- alternative kernels (Winograd, FFT)
- for sufficiently small problems (or sufficiently large memory sizes), all matmul based models (including im2col)
- naive, by over an order of magnitude

(Example for Resnet50 layers 1 and 2, mixed

Spoiler alert: this is a communication *lower bound*. More on this in 10 minutes…

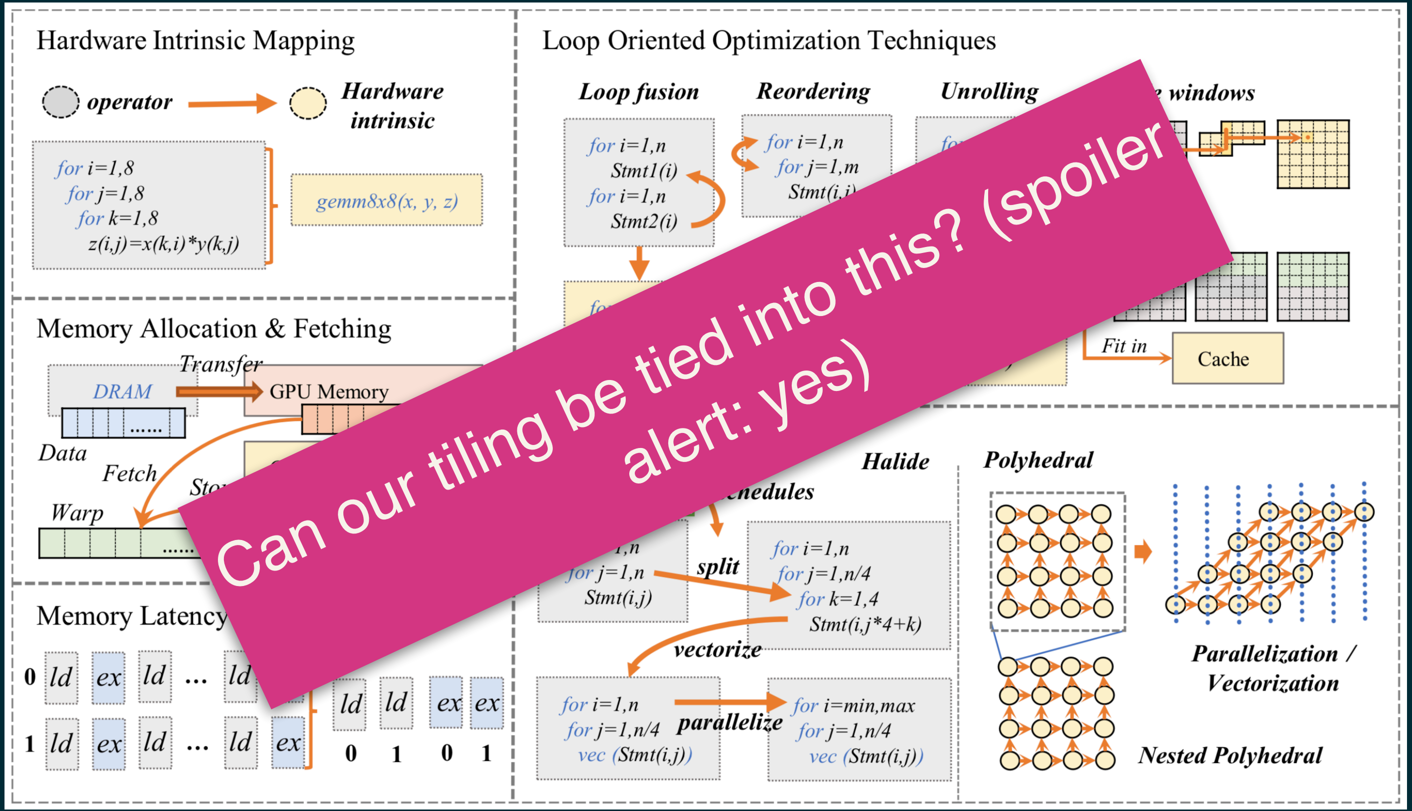
Theoretical comparison:



Even greater impact for parallel case ('smaller' problem size)!

How well does it work in practice?

Real HW is complicated



10^{13} possibilities for mapping space for one layer alone!

Image source: *The Deep Learning Compiler: A Comprehensive Survey* (Li et. al '20),

HW-specific quirks

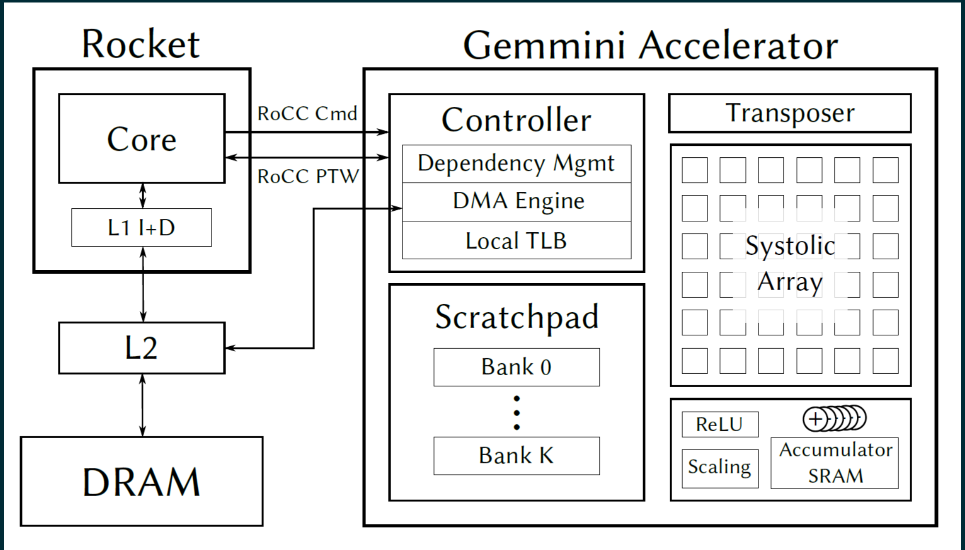
Benchmark platform

- Cycle-accurate simulation of a GEMMINI (Genc et al DAC '21 best paper) systolic accelerator connected to a a RISCV Rocket core
- Accurate DRAM model simulates on/off chip memory access.

HW-specific quirks

🧐 Separate buffers: accumulators for output and scratchpad for inputs and weights, cannot be shared

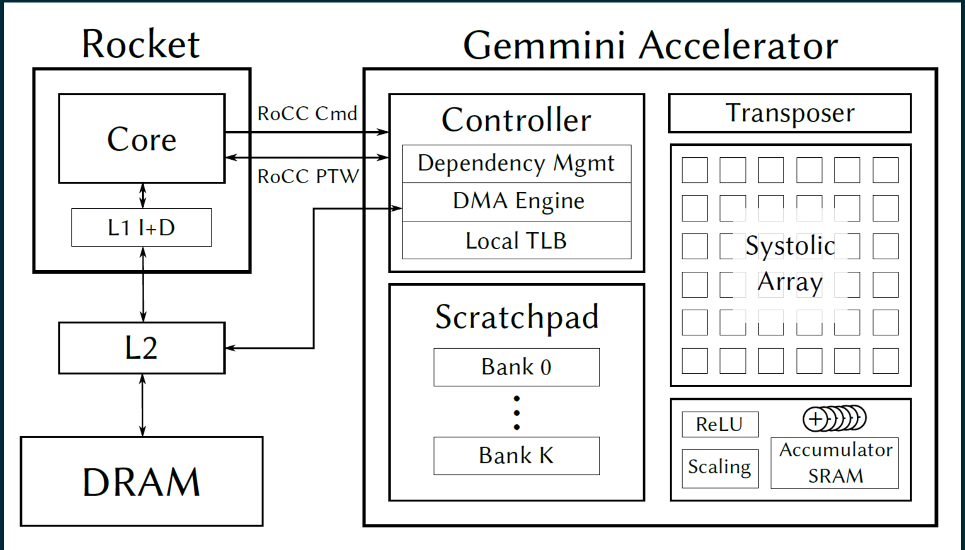
🧙 Replace *M/3* in constraints with sizes of separate buffers, and use sigmoidal optimization to handle "sum of tile < M" constraints.



HW-specific quirks

🧐 Mixed precision: scratchpad is 16-bit, accumulator is 32-bit but writes to DRAM at 16-bit - cannot tile along reduction axes.

🧙 Scale memory sizes in constraints by 1/word size (our theoretical results apply to mixed precision - more on that in a bit)



HW-specific quirks

🧐 Double-buffering: use half of buffer to work while other half is receiving data from memory to interleave computation and communication.

🧙 Halve memory size when computing tile sizes.

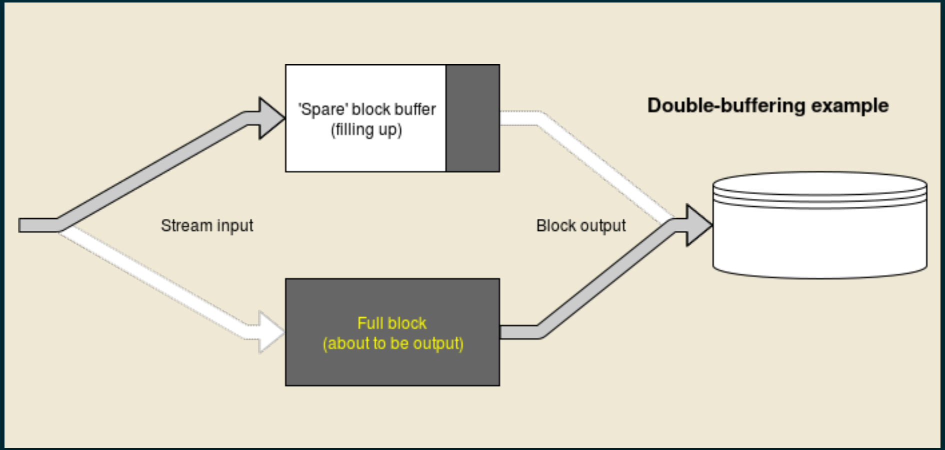


Image source: U. Manchester COMP25111, https://xerxes.cs.manchester.ac.uk/comp251/kb/

HW-specific quirks

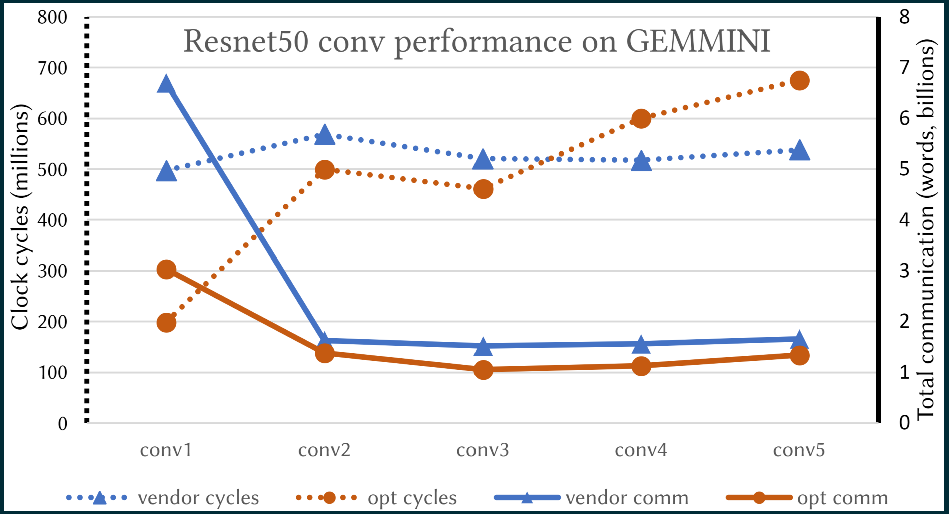
🧐 hot-start: begin computing output pixels when just enough inputs have been received (instead of waiting for whole tile to be loaded)

🧙 Lower-order term.

Benchmark results

Significant perf boost and reduction in communication over vendor library for resnet conv1 (large filter, where our algorithm gives most benefit), smaller one for conv2.

Frame conv3-5 not memory-bound in GEMMINI library; as a result our algorithm (which does not account for things like alignment) performs slightly worse in cycles despite doing better in communication.



Ongoing work: integration into compiler framework to allow HW-specific optimizations.

Can we do better?
(No*)

Communication Lower Bounds

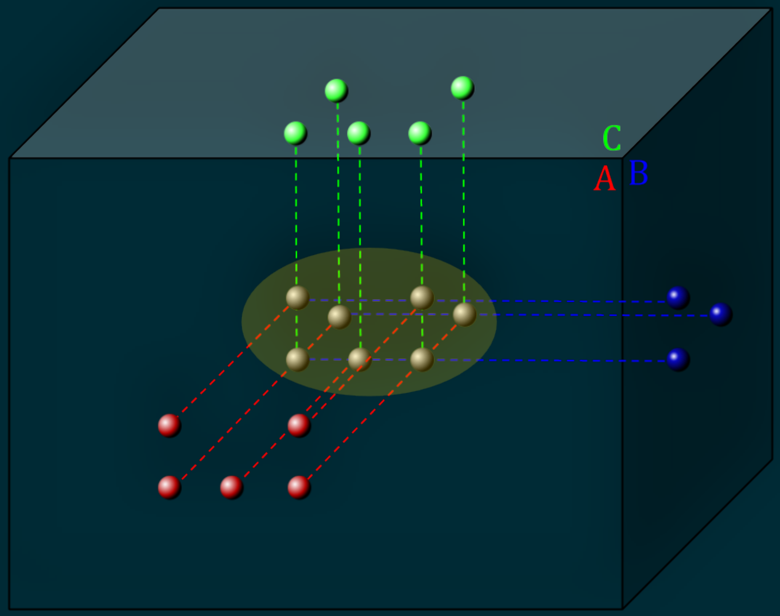
Strategy: Derive new **lower bounds** on communication volume over all possible reorderings of the algorithm.

Our tiling **attains the lower bound** (proof in paper).

Proof Technique: the Loomis-Whitney Inequality

Lemma: Given a finite set $V \subset \mathbb{Z}^3$ and projections V_A, V_B, V_C onto each coordinate plane, the volume of V is at most $|V| \leq \sqrt{|V_A||V_B||V_C|}$.

Represent each *arithmetic operation* as a point in *iteration domain* V, with V_A, V_B, V_C representing array accesses



Proof Sketch: Matmul Communication Bound

Communication lower bound for multiplying two $n \times n$ matrices $C = AB$:

1. Split code into L segments, each doing M communications
2. Each segment has access to $2M$ elements of A, B, and C
3. Accesses are projections, so L-W Inequality: Number of operations per segment $\leq \sqrt{8M^3}$
4. Need to do n^3 operations, so need $n^3/\sqrt{8M^3}$ segments
5. M loads per segment, so need $n^3/\sqrt{8M}$ communications

Smith et al. optimized the constant using Lagrange multipliers.

Generalizations…

- Matrix inversion, QR decomposition, LU decomposition, and other dense linear algebra (Christ et al. ’13)
- More complicated loop nests with affine array accesses (Demmel and Rusciano ’16)
- Different architectures (multilayer caches, parallel architectures)
- Sparse operations
- Mixed-precision data

Our contribution: a new generalization to CNNs including a constant factor and capable of handling mixed precision and parallel architectures.

Loomis-Whitney to Hölder-Brascamp-Lieb

Let $\phi_j : \mathbb{Z}^d \to \mathbb{Z}^{d_j}$ be group homomorphisms and $s \in [0,1]^m$. Then if for all subgroups $H \leq \mathbb{Z}^d$

$$\operatorname{rank}(H) \leq \sum_{j=1}^{m} s_j \operatorname{rank}(\phi_j(H))$$

Then for all nonempty finite $V \subseteq \mathbb{Z}^d$:

$$|V| \leq \prod_{j=1}^{m} |\phi_j(V)|^{s_j}$$

Proving HBL

- Standard proof: inductive argument involving careful decomposition along the kernels of the homomorphisms.

- Tao et al, "The Brascamp-Lieb Inequalities: Finiteness, Structure and Extremals" proves HBL using heat flow:
  - Gaussians maximize the inequality
  - Show monotonicity in the inequality along a heat flow

Using HBL

$$\phi_I(i_1, i_2, i_3, i_4, i_5, i_6, i_7) = (i_1, i_2, i_6 + \sigma_w i_4, i_7 + \sigma_h i_5)$$
$$\phi_F(i_1, i_2, i_3, i_4, i_5, i_6, i_7) = (i_2, i_3, i_6, i_7)$$
$$\phi_O(i_1, i_2, i_3, i_4, i_5, i_6, i_7) = (i_1, i_3, i_4, i_5)$$

Using HBL

H	rk	$\text{rk} \circ \phi_I$	$\text{rk} \circ \phi_F$	$\text{rk} \circ \phi_O$	Constraint
$C_{1,1}$	1	1	0	1	$1 \le s_I + s_O$
$C_{2,1}$	1	1	1	0	$1 \le s_I + s_F$
$C_{3,1}$	1	0	1	1	$1 \le s_F + s_O$
$C_{4,1}$	1	1	0	1	$1 \le s_I + s_O$
$C_{4,2}$	1	1	1	0	$1 \le s_I + s_F$
$C_{4,3}$	1	0	1	1	$1 \le s_F + s_O$
$C_{4,4}$	2	1	1	1	$2 \le s_I + s_F + s_O$
$C_{5,1}$	1	1	0	1	$1 \le s_I + s_O$
$C_{5,2}$	1	1	1	0	$1 \le s_I + s_F$
$C_{5,3}$	1	0	1	1	$1 \le s_F + s_O$
$C_{5,4}$	2	1	1	1	$2 \le s_I + s_F + s_O$

$$s_j = 2p_j/(p_I + p_F + p_O)$$

Our result 😎

Main Theorem: The number of words X communicated by a convolution which does G operations, uses filters of size $w_F \times h_F$, has strides σ_w, σ_h and runs with a fast memory of size M is:

$$X \geq \max\left\{\frac{9G}{4M} - M, \frac{2G\sqrt{\sigma_w \sigma_h}}{\sqrt{w_F h_F M}} - 2M, |\,\mathrm{Image}\,| + |\,\mathrm{Filter}\,| + |\,\mathrm{Out}\,|\right\}$$

Attained (to within small factor) by our tiling. Proof by LP duality!

Parallel results

Theorem: The number of words X communicated by a convolution which does G operations, uses filters of size $w_F \times h_F$, has strides σ_w, σ_h and runs on a distributed architecture with P processors with memory size M is:

$$X \geq \max\left\{ \frac{9G}{4PM} - M, \frac{2G\sqrt{\sigma_w \sigma_h}}{P\sqrt{w_F h_F M}} - 2M \right\}$$

Further work

- Designing hardware accelerators to take advantage of optimal CNN tilings
- Optimizations for GPUs and other parallel architectures
- Closing the gap between current CNN algorithms and lower bounds
- Integrate into compiler framework (EXO, Bernstein et al. PLDI ’22) to allow easy HW-specific optimizations.

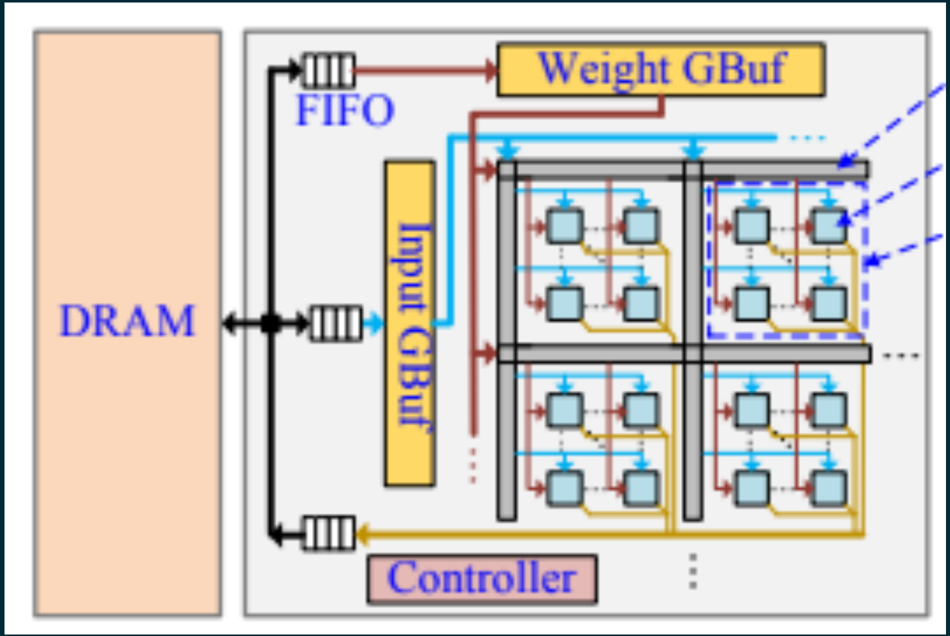


Image Source: Chen et al., 2020, *Communication Lower Bounds in Convolution Accelerators*,, ar

Selected references

- Online collection of papers: bebop.cs.berkeley.edu
- Survey: J. Demmel et al. 2014, *Communication lower bounds and optimal algorithms for numerical linear algebra.*
- Lower bounds using HBL: M. Christ, J. Demmel et al. 2013, *Communication lower bounds and optimal algorithms for programs that reference arrays.*
- CNN lower bounds: J. Demmel, G. Dinh 2018, *Communication-optimal convolutional neural nets.*
- Communication-optimal matmul: T. Hoefler, G. Kwasniewski et al. 2019, *Red-blue pebbling revisited: Near optimal parallel matrix-matrix multiplication.*
- Communication-optimal matmul: T. Smith et al. (Feb 2019). *A Tight I/O Lower Bound for Matrix Multiplication*

Thank you!

Scan above to read our paper…

…or come talk to us!

Anthony Chen
cygnari@umich.edu

Grace Dinh
gnd@berkeley.edu

Mason Haberle
mason.haberle@nyu.edu