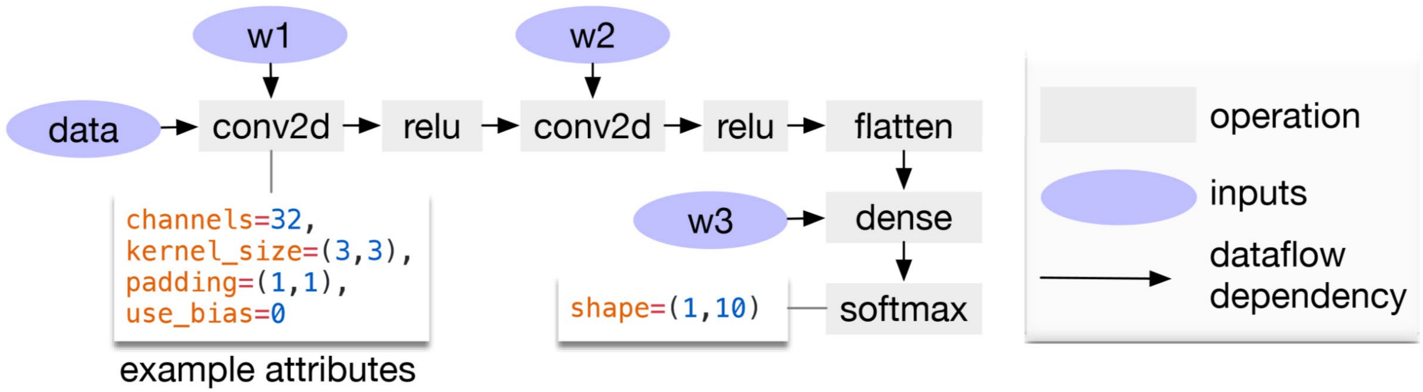
Lecture 5: Neural Network Compilation for Performance

Puneet Gupta

Logistics

- Part 1 of Project 1 should be on Piazza today. Part 2 likely tomorrow.
- Due April 29
- Code submission details to be announced by TA
- Report should be **6 slides** uploaded on Gradescope.
  - Briefly outline your approach and results
  - For group projects, create a group submission in Gradescope and submit once

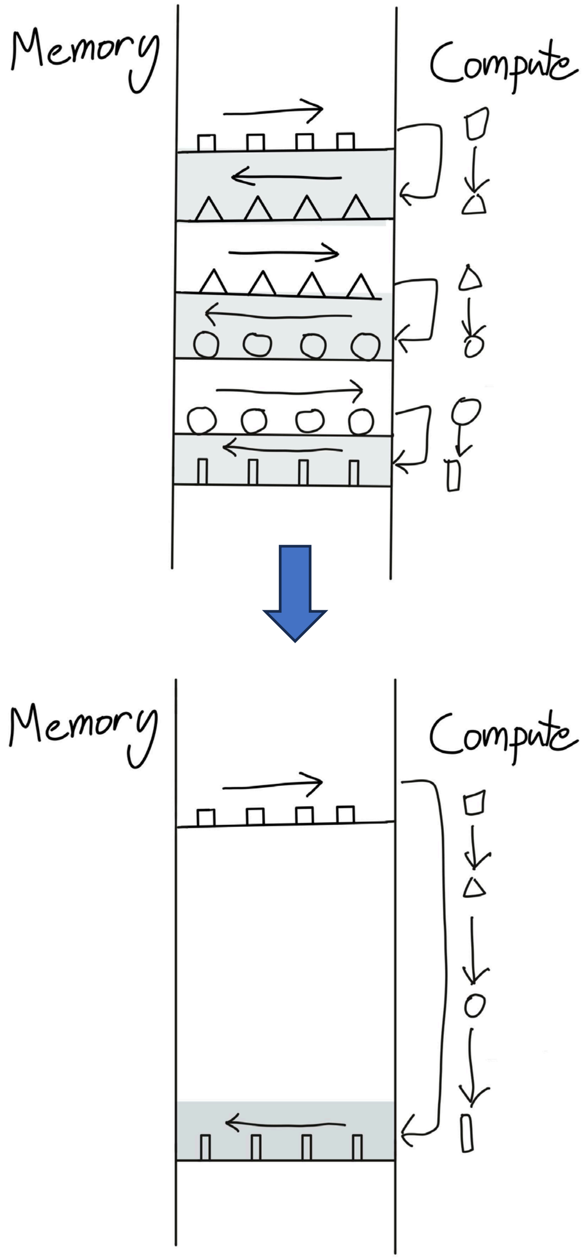
What would a NN “compiler” do ?



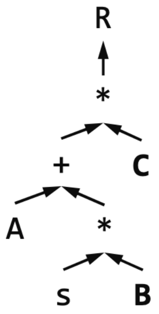
- Rewrite the computational graphs to functionally equivalent, but more efficient ones.
- Subject to what hardware backend we are running on.
- High level computation graph optimizations as well as operator level optimizations
  - Fuses whichever operators it can to reduce memory operations
  - Transforms the shapes of intermediate tensor data to allow for more efficient execution on the used hardware

Example: Operator Fusion

- MAC vs. FMA
  - Fused Multiply-Add done in one step with a single rounding
  - Fused operator can also simplify control logic (no need to shuttle data from MULT to ADD)
- Key Idea behind operator fusion
  - Do not go to memory repeatedly but store intermediate tensors locally
  - Can fuse one-to-one operations (e.g., Bias+ReLU) or reduction operators with one-to-one (e.g,, CONV with ReLU)

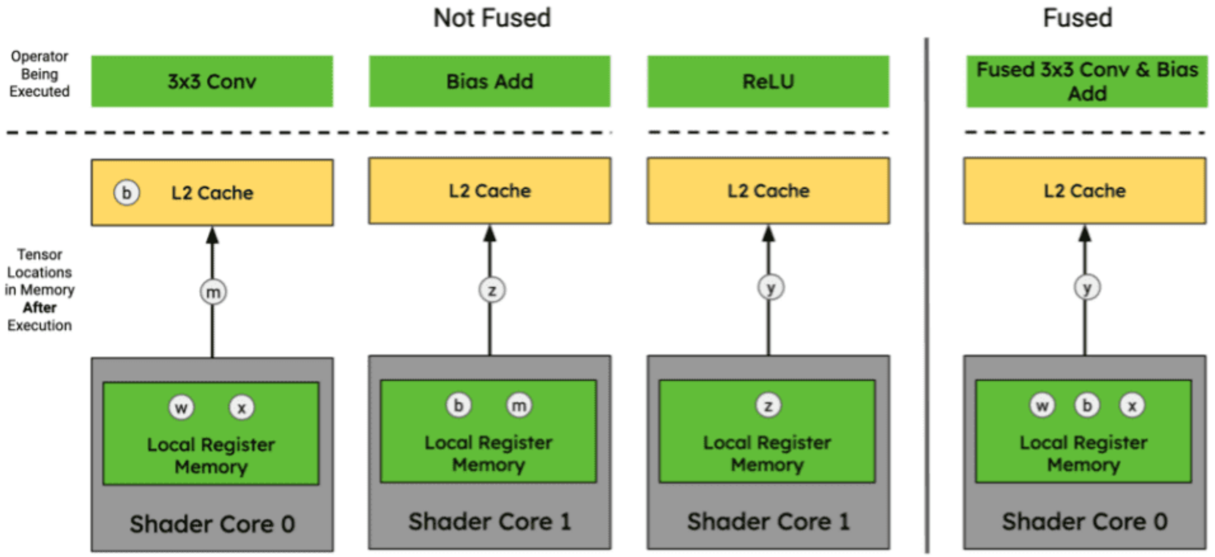


Operator Fusion




```
for( i in 1:n )
  tmp1[i,1] = s * B[i,1];
for( i in 1:n )
  tmp2[i,1] = A[i,1] + tmp1[i,1];
for( i in 1:n )
  R[i,1] = tmp2[i,1] * C[i,1];
```

```
for( i in 1:n )
  R[i,1] = (A[i,i] + s*B[i,1]) * C[i,1];
```



- Can save costly large memory accesses but may require more storage in the local buffers/registers/memory

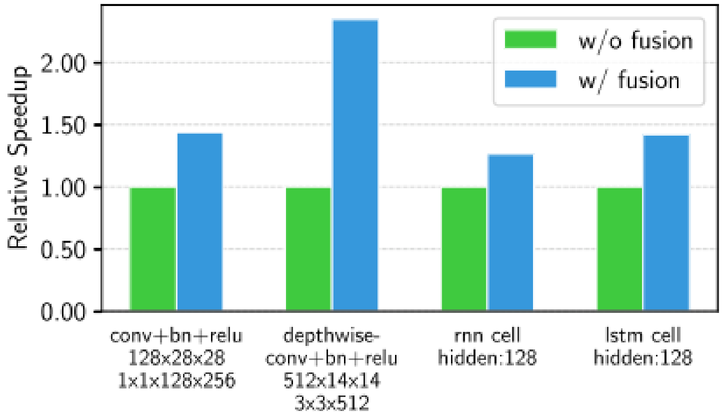


Figure 4: Performance comparison between fused and non-fused operations. TVM generates both operations. Tested on NVIDIA Titan X.

Example: Loop Tiling/Data Layout

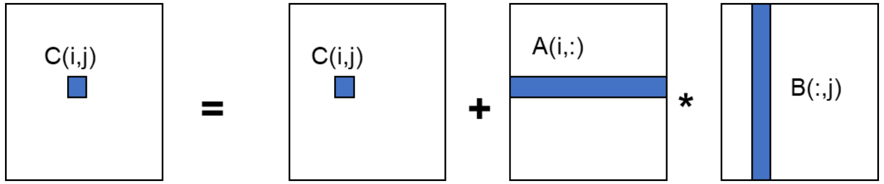
- Naieve GEMM:

for i = 1 to n
{read row i of A into fast memory}
for j = 1 to n
{read C(i,j) into fast memory}
{read column j of B into fast memory}
for k = 1 to n
*C(i,j) = C(i,j) + A(i,k) * B(k,j)*
{write C(i,j) back to slow memory}

- Tiled GEMM
  - Block/tile size *b* = n/N

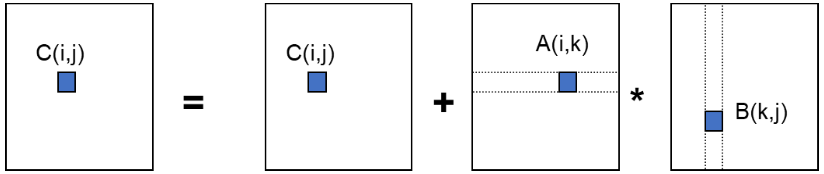
for i = 1 to N
for j = 1 to N
{read block C(i,j) into fast memory}
for k = 1 to N
{read block A(i,k) into fast memory}
{read block B(k,j) into fast memory}
*C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}*
{write block C(i,j) back to slow memory}

- Need to be able to fit enough data (3b) in local memory/caches



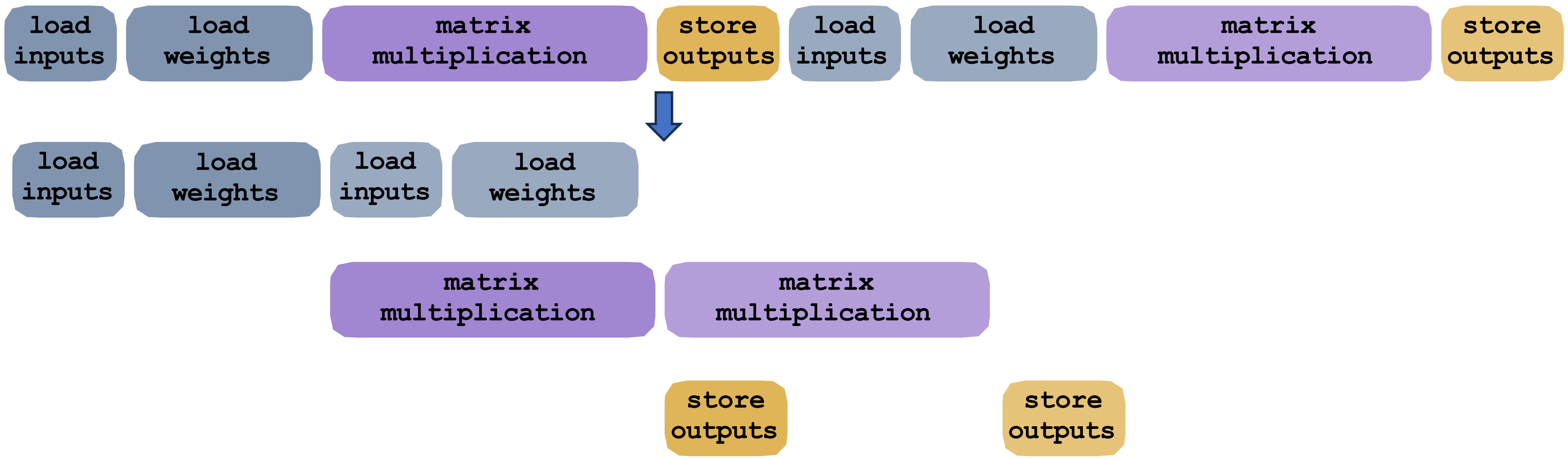
Arithmetic intensity:

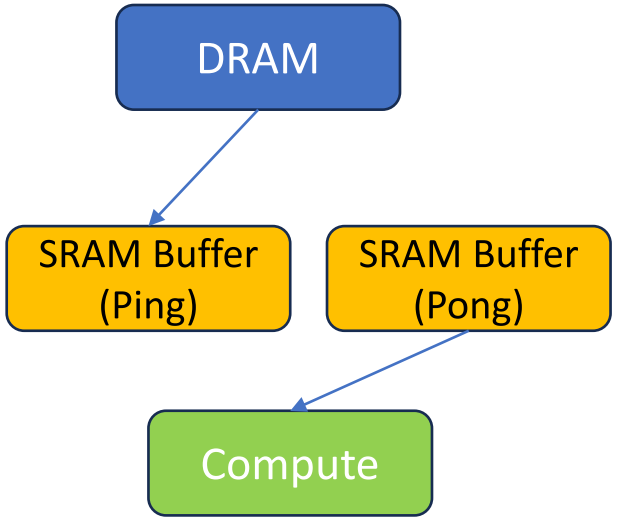
n^3 to read each column of B n times
$+ n^2$ to read each row of A once
$+ 2n^2$ to read and write each element of C once
$= n^3 + 3n^2$
ops = $2n^3$ → **AI ≈ 2**



$N*n^2$ read each block of B N^3 times ($N^3 * b^2 = N^3 * (n/N)^2 = N*n^2$)
$+ N*n^2$ read each block of A N^3 times
$+ 2n^2$ read and write each block of C once
$= (2N + 2) * n^2$
AI ≈ n / N = b for large n

Example: Hiding Memory Latency





- Control needs to account for dependencies
- You may need on-chip ping-pong buffers to overlap DRAM reads with compute
  - Ping buffer filled at the same time as Pong buffer being read

Papers for Week 4

1. Chen, Tianqi, et al. "TVM: end-to-end optimization stack for deep learning." *arXiv preprint arXiv:1802.04799* 11.2018 (2018): 20.
2. Jacob, Benoit, et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.
3. Wang, Kuan, et al. "Haq: Hardware-aware automated quantization with mixed precision." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.
4. Rastegari, Mohammad, et al. "Xnor-net: Imagenet classification using binary convolutional neural networks." *European conference on computer vision*. Cham: Springer International Publishing, 2016.
5. Han, Song, et al. "Learning both weights and connections for efficient neural network." *Advances in neural information processing systems* 28 (2015).