Parallelization & Stationarity in DNN HW Accelerators

Intelligent Architectures: 5LIL0

March 25, 2025

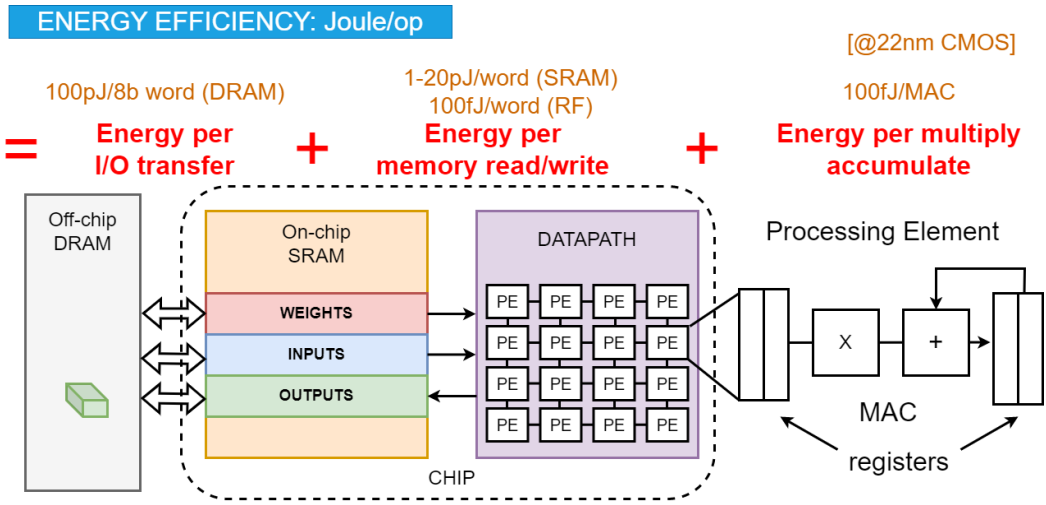
Dr. Federico Corradi

Department of Electrical Engineering, Electronic Systems Group

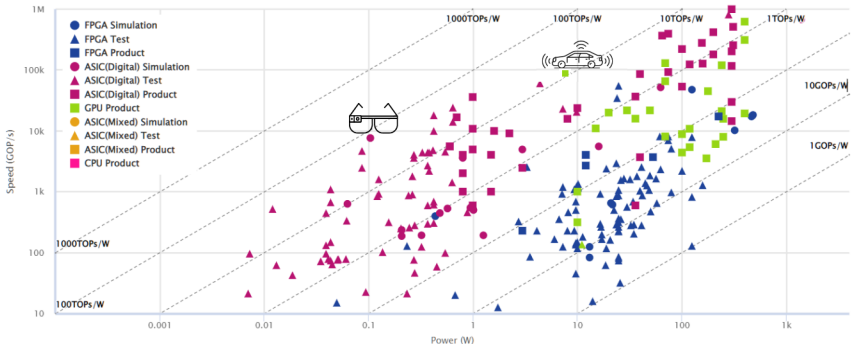
TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Outline

Energy Efficient DNN processors, where are we?



Energy Efficient DNN processors, where are we?



Are we there?

Unfortunately, not yet! These numbers report only peak efficiency and not workload efficiency... but we are getting close. source nicsefc.ee.tsinghua.edu.cn

Lesson Plan

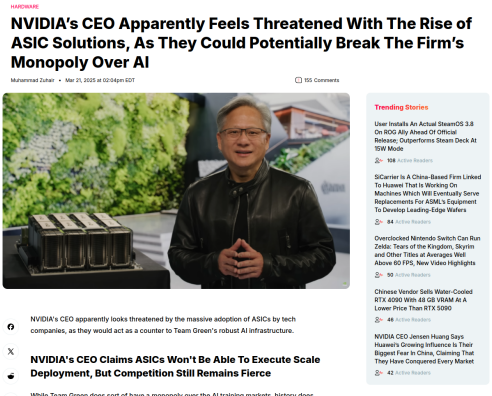
GeMM & CNN processors enhancements in ASIC, CPU, GPU

- Parallelization (spatial unrolling optimization)
- Stationarity (temporal unrolling optimization)
- Extending spatial and temporal unrolling to higher levels

Diving into state-of-the-art accelerators

- Tensor Core (GPUs)
- Tesla TPU
- Huawei Da Vinci
- Google Systolic Array TPU

Some News



HARDWARE

NVIDIA's CEO Apparently Feels Threatened With The Rise of ASIC Solutions, As They Could Potentially Break The Firm's Monopoly Over AI

Muhammad Zuhair • Mar 21, 2025 at 02:04pm EDT

155 Comments

NVIDIA's CEO apparently looks threatened by the massive adoption of ASICs by tech companies, as they would act as a counter to Team Green's robust AI infrastructure.

NVIDIA's CEO Claims ASICs Won't Be Able To Execute Scale Deployment, But Competition Still Remains Fierce

While Team Green does sort of have a monopoly over the AI training markets, history does

Trending Stories

User Installs An Actual SteamOS 3.8 On ROG Ally Ahead Of Official Release; Outperforms Steam Deck At 15W Mode

158 Active Readers

SiCarrier Is A China-Based Firm Linked To Huawei That Is Working On Machines Which Will Eventually Serve Replacements For ASML's Equipment To Develop Leading-Edge Wafers

84 Active Readers

Overclocked Nintendo Switch Can Run Zelda: Tears of the Kingdom, Skyrim and Other Titles at Averages Well Above 60 FPS, New Video Highlights

90 Active Readers

Chinese Vendor Sells Water-Cooled RTX 4090 With 48 GB VRAM At A Lower Price Than RTX 5090

46 Active Readers

NVIDIA CEO Jensen Huang Says Huawei's Growing Influence Is Their Biggest Fear In China, Claiming That They Have Conquered Every Market

42 Active Readers

[21/03/2025 WCCFTECH]

ASIC

Firms like Google, Microsoft, Broadcom, Apple, OpenAI and many others are involved in developing their custom chips, not only to challenge NVIDIA's dominance, but to find an alternative to the monopoly in the markets.

Outline

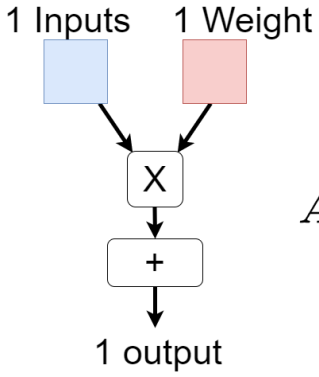
Arithmetic intensity as function of parallelism

```
for(b=0 to B-1); //for each image in the batch
        for(k=0 to K-1); //for each output channel
                parfor(c=0 to S-1); //for each input channel
                        o[b][k] += i[b][c]*w[c][k]
                // (#S multiply accumulate block in a chip)
```

Arithmetic intensity

How many operations can I do per data word that I fetch? (i.e., ops/data word)

Arithmetic intensity as function of parallelism

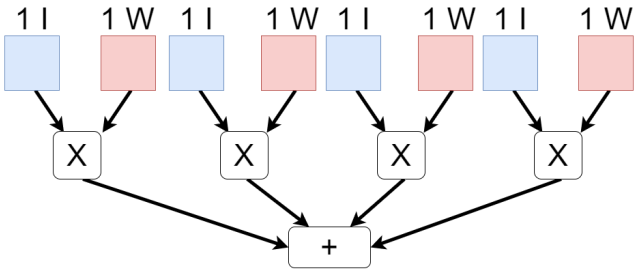


A_I = 1OP/3W = 1/3

- $BW_I = 1 words/cc$
- $BW_W = 1 words/cc$
- $BW_o = 1 words/cc$
- $TP = 1 ops/cc$
- $A_I(RF) = 1/3$ ops/words

BW = Bandwidth, TP = Throughput, A_i = Arithmetic Intensity, RF = Register File

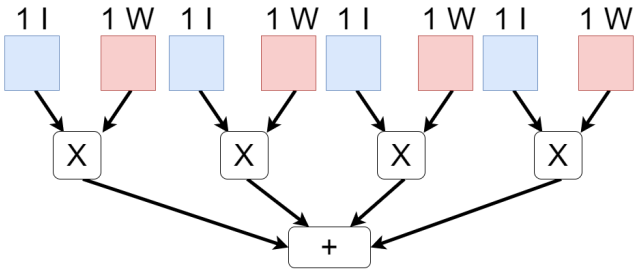
Arithmetic intensity as function of parallelism



A_I = 4OP/9W = 4/9

- $BW_I = 4 words/cc$
- $BW_W = 4 words/cc$
- $BW_o = 1 words/cc$
- $TP = 4 ops/cc$
- $A_I(RF) = 4/9$ ops/words

Arithmetic intensity as function of parallelism



- $BW_I = 4 words/cc$
- $BW_W = 4 words/cc$
- $BW_o = 1 words/cc$
- $TP = 4 ops/cc$
- $A_I(RF) = 4/9$ ops/words

The higher A_I, the better!

Fetching in parallel is not only positive for speed but also for energy efficiency, i.e. less data movement per operation! How to achieve the highest possible A_I?

Optimal reuse of I and W data!

```
parfor(b=0 to B-1); //for each image in the batch
        parfor(k=0 to K-1); //for each output channel
                parfor(c=0 to S-1); //for each input channel
                        o[b][k] += i[b][c]*w[c][k]
                // (#S multiply accumulate block in a chip)
```

How to achieve the highest possible A_I?

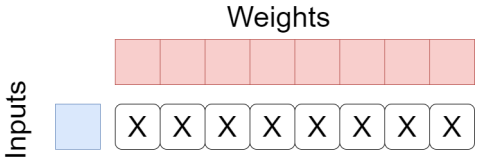
Carry everything in parallel with as much as possible data reuse!

Optimal reuse of I and W data!

Example for B=9; C=4; K=8

Maximum data reuse for CNN?

- Every input element is multiplied with K weights

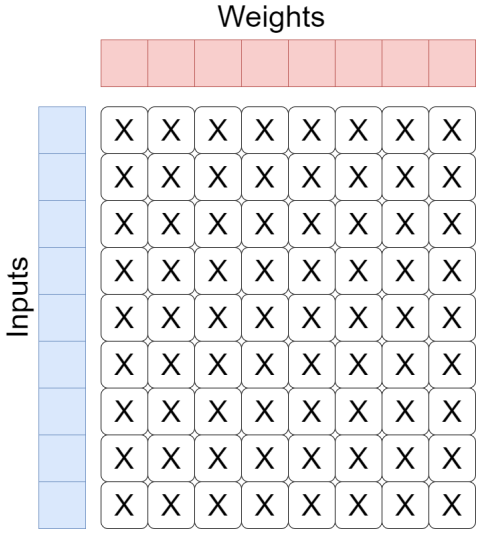


Optimal reuse of I and W data!

Example for B=9; C=4; K=8

Maximum data reuse for CNN?

- Every input element is multiplied with K weights
- Every weight element is multiplied with 9 inputs

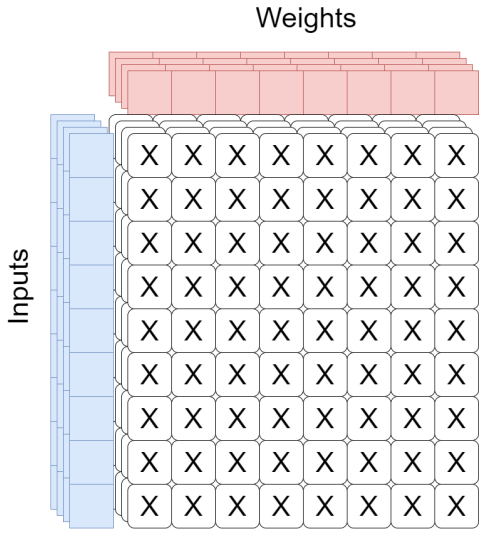


Optimal reuse of I and W data!

Example for B=9; C=4; K=8

Maximum data reuse for CNN?

- Every input element is multiplied with K weights
- Every weight element is multiplied with 9 inputs
- Do this for all input channels C in parallel

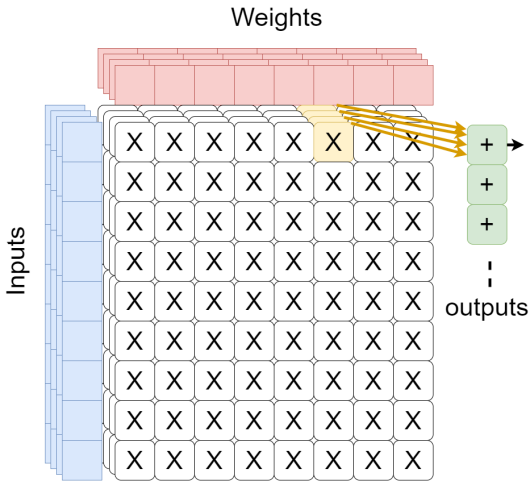


Optimal reuse of I and W data!

Example for B=9; C=4; K=8

Maximum data reuse for CNN?

- Every input element is multiplied with K weights
- Every weight element is multiplied with 9 inputs
- Do this for all input channels C in parallel
- Locally accumulate partial sums for every K.B output element

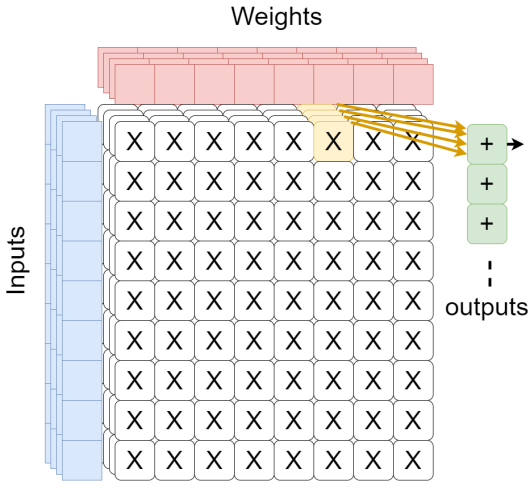


Optimal reuse of I and W data!

Example for $B = 9$; $C = 4$; $K = 8$

Maximum data reuse for CNN

- We only fetch each input once, each weight once, and the outputs are generated in one go!

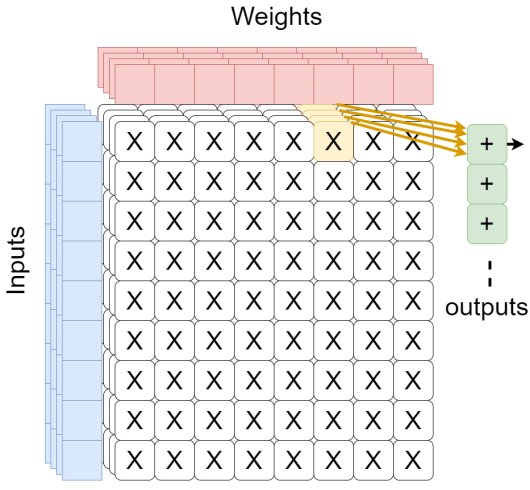


Optimal reuse of I and W data!

Example for $B = 9$; $C = 4$; $K = 8$

Maximum data reuse for CNN

- We only fetch each input once, each weight once, and the outputs are generated in one go!
- num. MAC
  $9 * 4 * 8 = 288, 140 (9 * 4 + 8 * 4 + 9 * 8)$
  memory operations

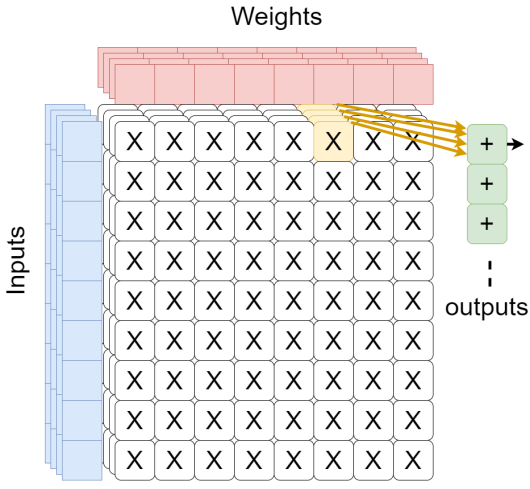


Optimal reuse of I and W data!

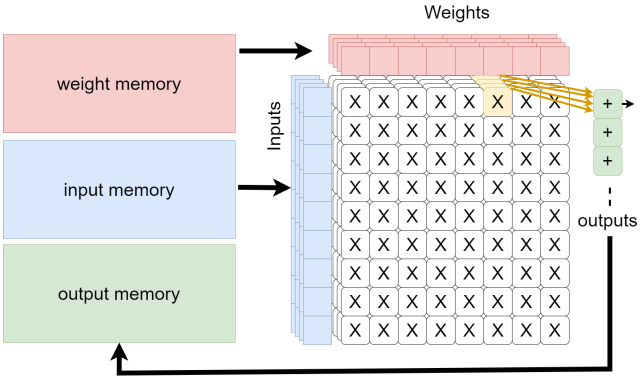
Example for $B = 9$; $C = 4$; $K = 8$

Maximum data reuse for CNN

- We only fetch each input once, each weight once, and the outputs are generated in one go!
- num. MAC
  $9 * 4 * 8 = 288, 140 (9 * 4 + 8 * 4 + 9 * 8)$
  memory operations
- $A_I(RF) = 288/140 \sim 2.0$

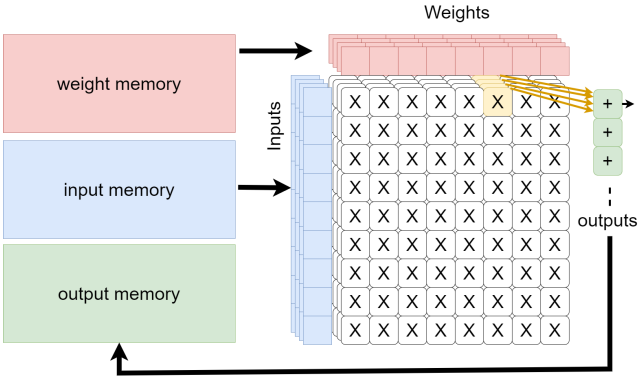


Optimal reuse of I and W data!



What is the problem?

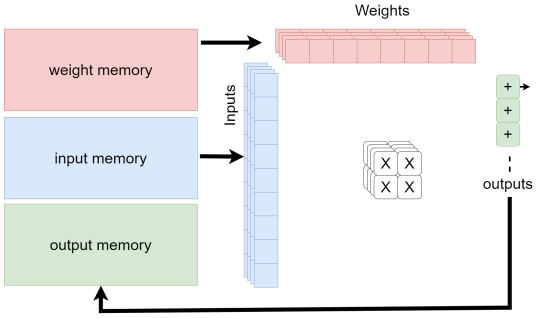
Optimal reuse of I and W data!



What is the problem?

In a realistic deep neural networks, we cannot afford to fully unroll the network (k,c,w,i dimensions are too large). Impractical for silicon area (i.e., cost).

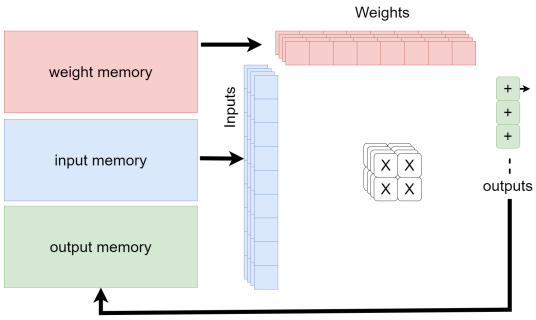
The main challenge: optimal reuse of data with limited number of MACs



Fewer multipliers?

- Typical accelerator has few 100's till few 1000's MACs

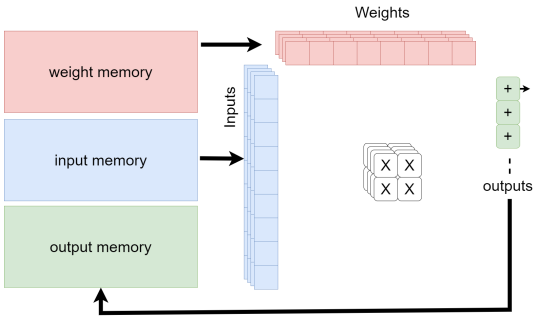
The main challenge: optimal reuse of data with limited number of MACs



Fewer multipliers?

- Typical accelerator has few 100's till few 1000's MACs
- Smartly 'fold' convolutions on multiplier array (optimizing spatial unrolling)

The main challenge: optimal reuse of data with limited number of MACs



Fewer multipliers?

- Typical accelerator has few 100's till few 1000's MACs
- Smartly 'fold' convolutions on multiplier array (optimizing spatial unrolling)
- Optimize towards minimal memory access! (reload weights, inputs, and data reuse)

Possible schemes for data reuse

Popular data reuse schemes

- Reuse of weight data (weight reuse)

Possible schemes for data reuse

Popular data reuse schemes

- Reuse of weight data (weight reuse)
- Reuse of input data (input reuse)

Possible schemes for data reuse

Popular data reuse schemes

- Reuse of weight data (weight reuse)
- Reuse of input data (input reuse)
- Reuse of output data (output reuse)

Possible schemes for data reuse

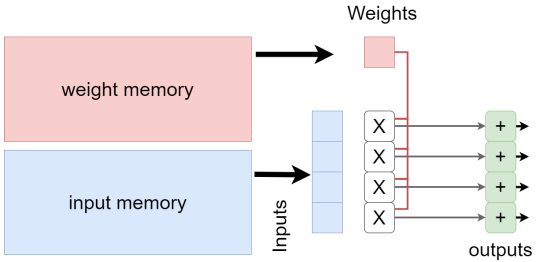
Popular data reuse schemes

- Reuse of weight data (weight reuse)
- Reuse of input data (input reuse)
- Reuse of output data (output reuse)
- Combination of the above (e.g., row reuse)

Possible schemes for data reuse: weight reuse

Data reuse schemes

- Reuse of weight data

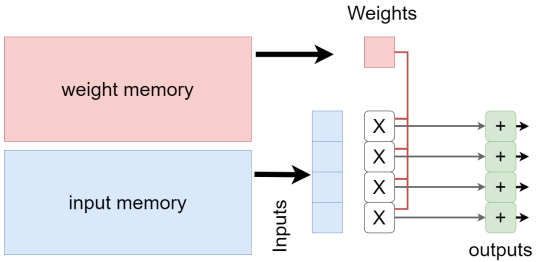



```
for(b2=0 to B/S-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(c=0 to C-1); //for each input channel
   parfor(b1=0 to S-1); //for each image in the batch
        o[b2*S+b1][k] += i[b2*S+b][c]*w[c][k]
   // (#S multiply accumulate block in a chip)
```

Possible schemes for data reuse: weight reuse

Data reuse schemes

- Reuse of weight data
- E.g. 4 multipliers

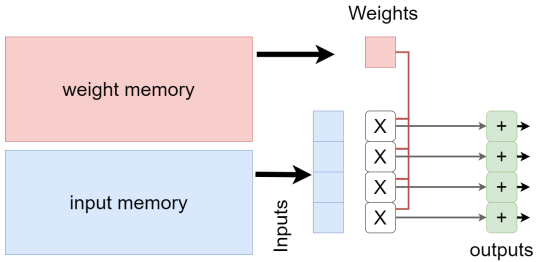



```
for(b2=0 to B/S-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(c=0 to C-1); //for each input channel
   parfor(b1=0 to S-1); //for each image in the batch
        o[b2*S+b1][k] += i[b2*S+b][c]*w[c][k]
   // (#S multiply accumulate block in a chip)
```

Possible schemes for data reuse: weight reuse

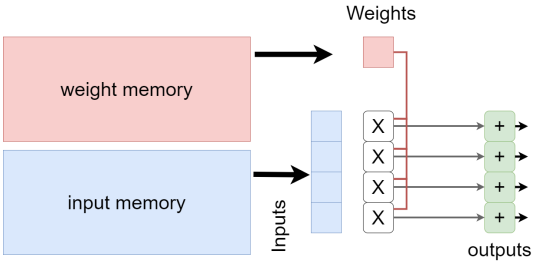
Data reuse schemes

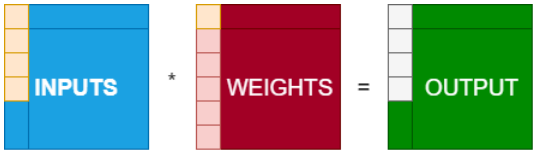
- Reuse of weight data
- E.g. 4 multipliers
- parfor over B as there is no B dependency in the weight




```
for(b2=0 to B/S-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(c=0 to C-1); //for each input channel
   parfor(b1=0 to S-1); //for each image in the batch
        o[b2*S+b1][k] += i[b2*S+b][c]*w[c][k]
   // (#S multiply accumulate block in a chip)
```

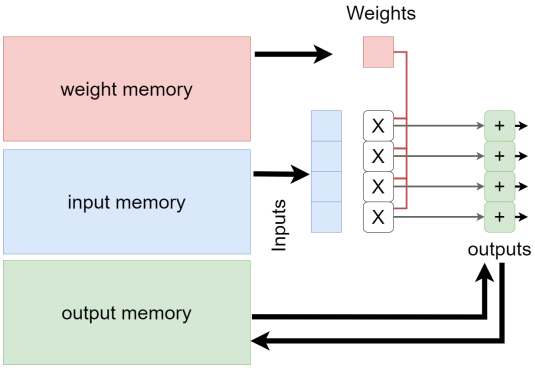
Possible schemes for data reuse: weight reuse

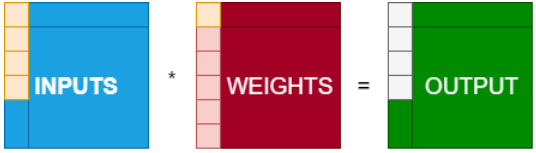




What is the problem with this?

Possible schemes for data reuse: weight reuse





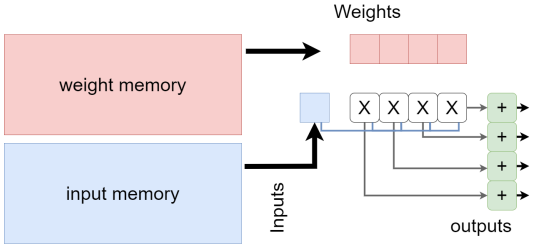
What is the problem with this?

- We only produce a partial sum on the outputs, and we will need to fetch back and forth the partial sums to complete the output.

Possible schemes for data reuse: input reuse

Data reuse schemes

- Reuse of input data

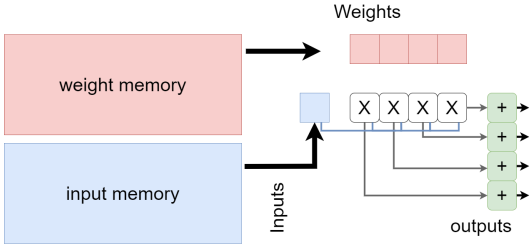



```
for(b=0 to B-1); //for each image in the batch
 for(k2=0 to K/S-1); //for each output channel
  for(c=0 to C-1); //for each input channel
        parfor(k1=0 to S-1); //for each input channel
         o[b][k2*S+k1] += i[b][c]*w[c][k2*S+k1]
        // (#S multiply accumulate block in a chip)
```

Possible schemes for data reuse: input reuse

Data reuse schemes

- Reuse of input data
- E.g. 4 multipliers

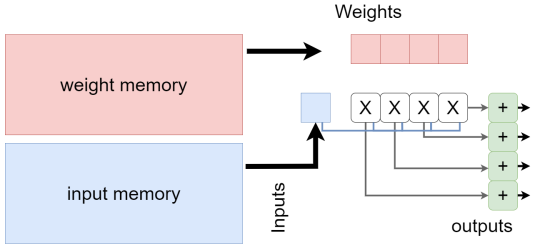



```
for(b=0 to B-1); //for each image in the batch
 for(k2=0 to K/S-1); //for each output channel
  for(c=0 to C-1); //for each input channel
        parfor(k1=0 to S-1); //for each input channel
         o[b][k2*S+k1] += i[b][c]*w[c][k2*S+k1]
        // (#S multiply accumulate block in a chip)
```

Possible schemes for data reuse: input reuse

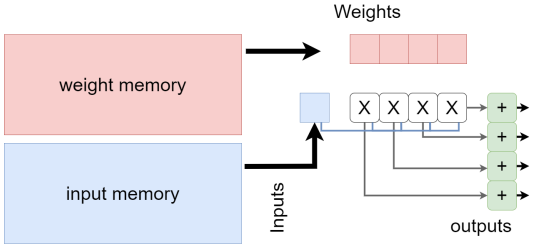
Data reuse schemes

- Reuse of input data
- E.g. 4 multipliers
- compute many partial results with 1 loaded input (parallelize across k)



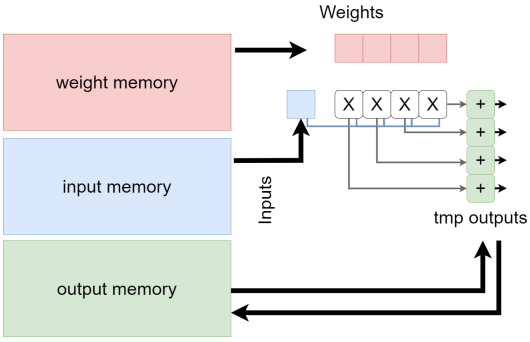

```
for(b=0 to B-1); //for each image in the batch
 for(k2=0 to K/S-1); //for each output channel
  for(c=0 to C-1); //for each input channel
        parfor(k1=0 to S-1); //for each input channel
         o[b][k2*S+k1] += i[b][c]*w[c][k2*S+k1]
        // (#S multiply accumulate block in a chip)
```

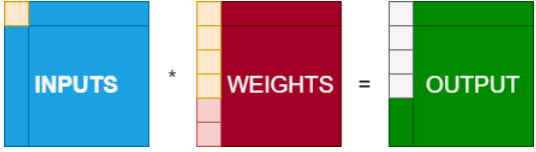
Possible schemes for data reuse: input reuse



What is the problem with this?

Possible schemes for data reuse: input reuse





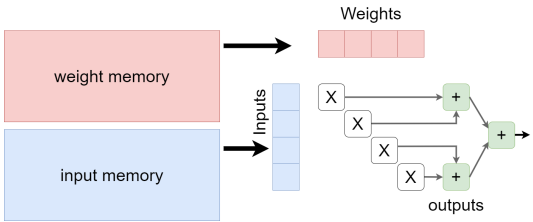
What is the problem with this?

- We only produce a partial sum on the outputs, and we will need to fetch back and forth the partial sums to complete the output.

Possible schemes for data reuse: output reuse

Data reuse schemes

- Reuse of output data

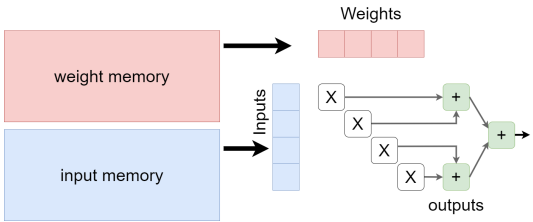



```
for(b=0 to B-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(c2=0 to C/S-1); //for each input channel
   parfor(c1=0 to S-1);
    o[b][k] += i[b][c2*S+c1]*w[c2*S+c1][k]
   // (#S multiply accumulate block in a chip)
```

Possible schemes for data reuse: output reuse

Data reuse schemes

- Reuse of output data
- E.g. 4 multipliers

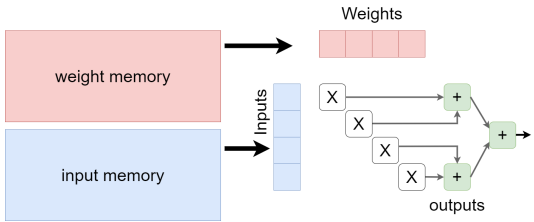



```
for(b=0 to B-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(c2=0 to C/S-1); //for each input channel
   parfor(c1=0 to S-1);
    o[b][k] += i[b][c2*S+c1]*w[c2*S+c1][k]
   // (#S multiply accumulate block in a chip)
```

Possible schemes for data reuse: output reuse

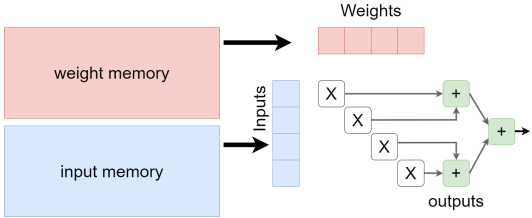
Data reuse schemes

- Reuse of output data
- E.g. 4 multipliers
- compute many partial results of 1 output in 1 cycle



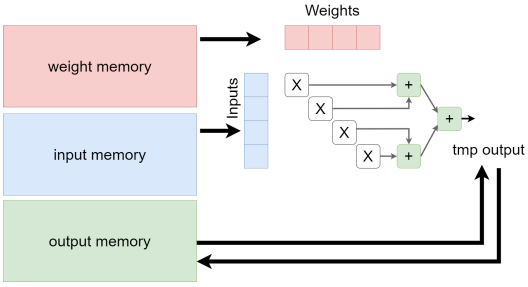

```
for(b=0 to B-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(c2=0 to C/S-1); //for each input channel
   parfor(c1=0 to S-1);
    o[b][k] += i[b][c2*S+c1]*w[c2*S+c1][k]
   // (#S multiply accumulate block in a chip)
```

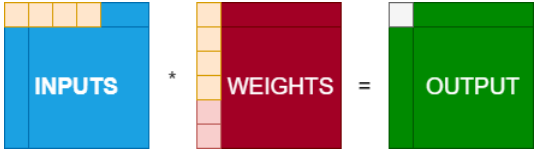
Possible schemes for data reuse: output reuse



What is the benefit of this?

Possible schemes for data reuse: output reuse

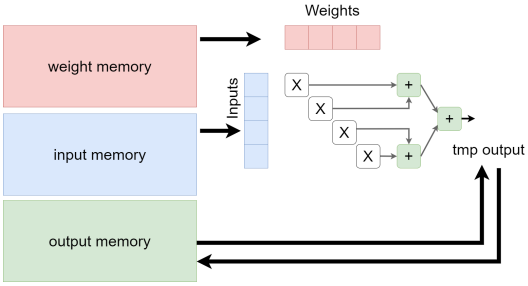


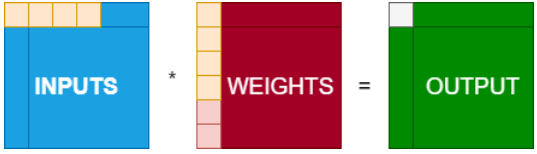


What is the benefit of this?

- We need to fetch less partial results

Possible schemes for data reuse: output reuse



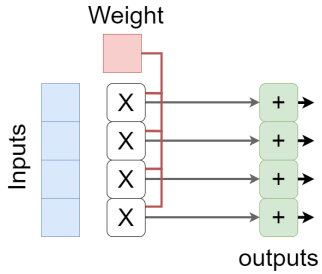


What is the benefit of this?

- We need to fetch less partial results
- This is called Fetch Multiply Accumulate (FMA) and CPUs support it via advanced vector extension (AVX) (e.g., Intel CPU processors)

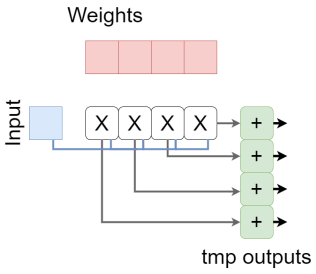
Spatial data reuse (spatial unrolling): summary

weight reuse



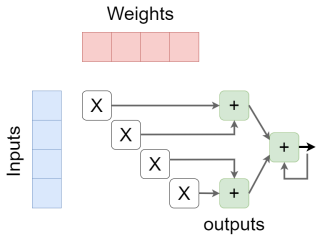
- weight BW = 1
- input BW = S
- output BW = S
- $A_I(RF)$ = S / (2S+1)

input reuse



- weight BW = S
- input BW = 1
- output BW = S
- $A_I(RF)$ = S / (2S+1)

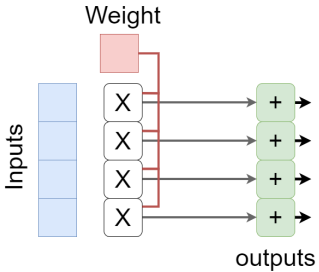
output reuse



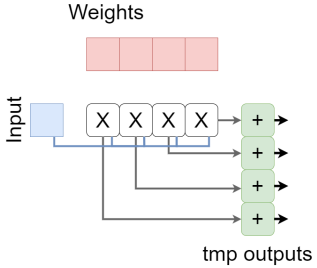
- weight BW = S
- input BW = S
- output BW = 1
- $A_I(RF)$ = S / (2S+1)

Spatial data reuse (spatial unrolling): summary

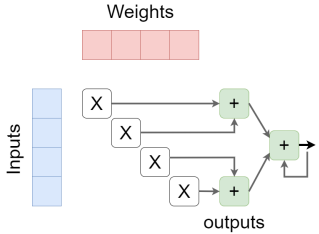
weight reuse



input reuse



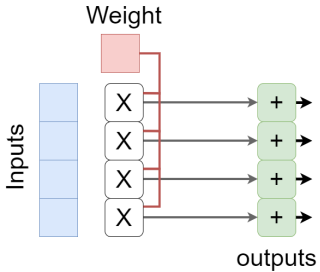
output reuse



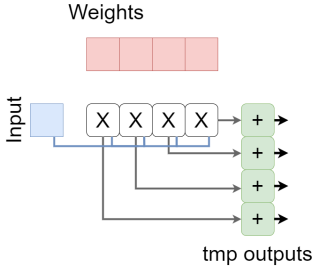
Which one is the best?

Spatial data reuse (spatial unrolling): summary

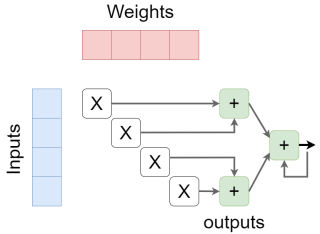
weight reuse



input reuse



output reuse

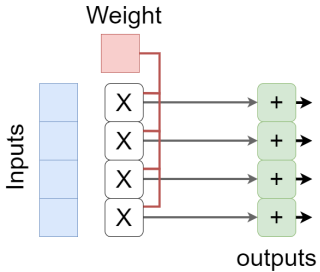


Which one is the best?

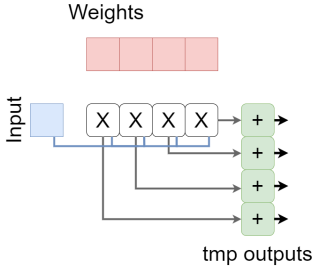
- It depends on your workloads

Spatial data reuse (spatial unrolling): summary

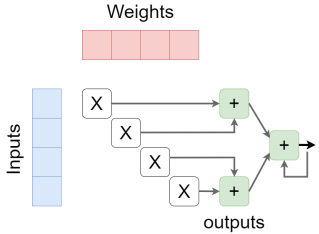
weight reuse



input reuse



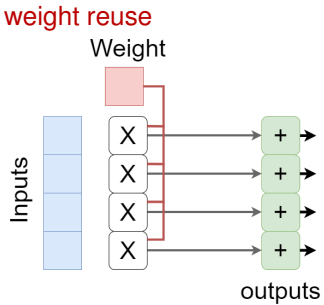
output reuse

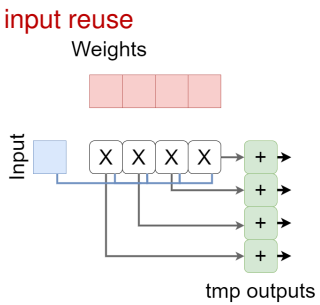


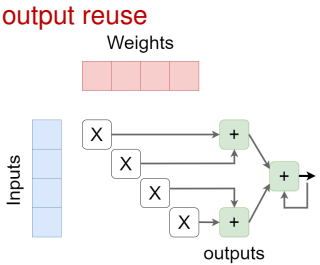
Which one is the best?

- It depends on your workloads
- Accelerators use a combination of those, we will see some practical examples

Spatial data reuse (spatial unrolling): summary





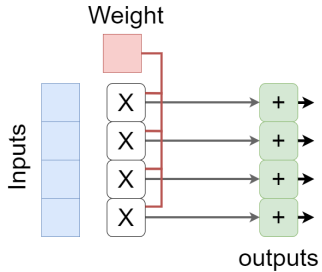


Which one is the best?

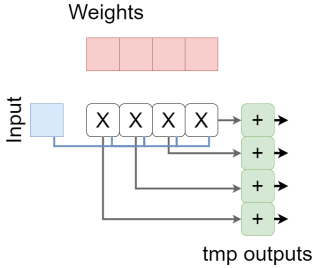
- It depends on your workloads
- Accelerators use a combination of those, we will see some practical examples
- But if you need to pick one? which one would you pick? and why?

Spatial data reuse (spatial unrolling): summary

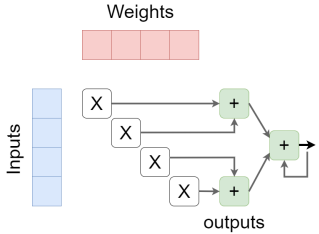
weight reuse



input reuse



output reuse



Which one is the best?

- It depends on your workloads
- Accelerators use a combination of those, we will see some practical examples
- But if you need to pick one? which one would you pick? and why?
- OS, because the output typically has more bits than the inputs!

Diving into state-of-the-art accelerators I

SOTA acceleratos

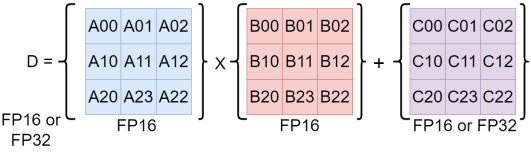
- Tensor cores in recent GPUs (NVIDIA)
- Intel Advanced Matrix eXtension (AMX)
- Tesla NPU
- Huawei DaVinci

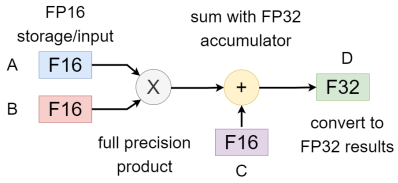
Tensor cores in recent GPUs (NVIDIA)

what is a Tensor Core?

A tensor core is a small piece of hardware inside the streaming processor of the GPU that:

- performs efficient GeMM tensor/matrix operation
- operates on 4x4 matrices and performs D=AxB+C
- tiles matrix operation in blocks of 4x4



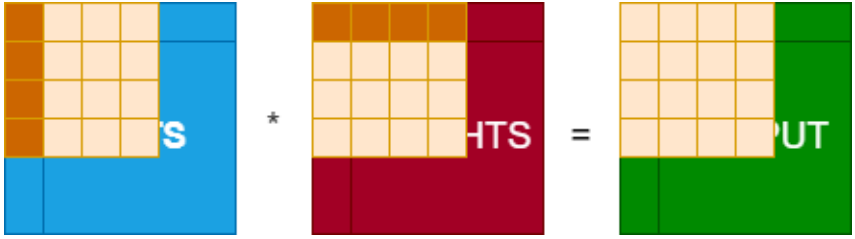


Why is it efficient?

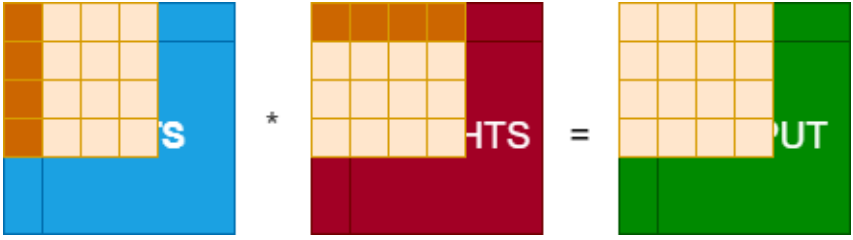
It exploits data reuse in a 4x4 matrix, input and weight reuse!

Tensor cores in recent GPUs (NVIDIA)

```
for(b2=0 to B/PB-1); //for each image in the batch
 for(k2=0 to K/PK-1); //for each output channel
  for(c2=0 to C/PC-1); //for each input channel
   for(c1=0 to PC-1); //for each input channel
    parfor(k1=0 to PK-1); // for each output channel
     parfor(b1=0 to PB-1); // for each image in the batch
      o[b2*PB+b1][k2*PK+k1] += i[b2*P+b1][c]*w[c][k2*PK+k1]
```



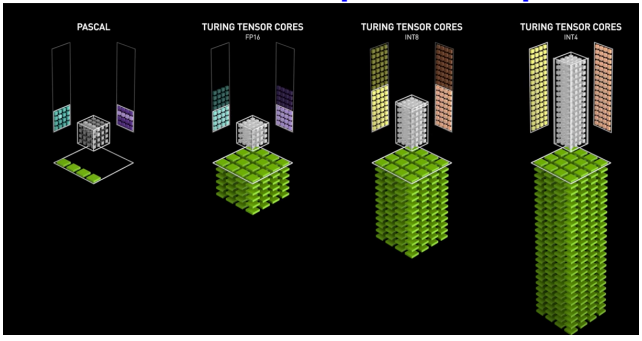
Tensor cores in recent GPUs (NVIDIA)



- 16 multipliers, 16 accumulators (16 partial results in 1 clock cycles, iterated over 4 clock cycles for full results)
- every i & w are reused 4 times
- every o is accumulated 4 times
- num. MAC$= 64; BW_I = BW_W = BW_O = 16$
- $A_I = 64/48 = 1.333$
- 4x data reuse on the input and weights!

Tensor cores in recent GPUs (NVIDIA)

Video from NVIDIA [Watch at home]

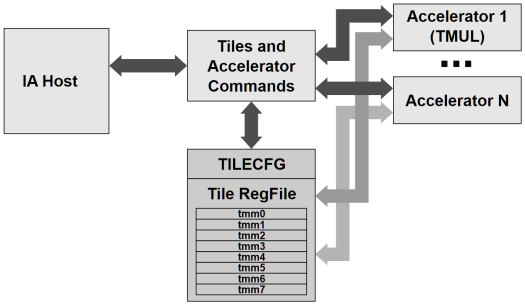


Tensor Cores in GPUs

They boost the performance of DNN in GPUs by about a factor 10! (i.e., from 100GOPS/W $\rightarrow$ 1 TOPS/W)

Intel Advanced Matrix eXtension (AMX)

- in 2022 Intel released the Sapphire Rapids with AMX instructions
- x86 Advanced Matrix eXtension (AMX)
- 2D-FMA (Fuse Multiply Add), cfr. Tensor Cores
- Executed in Tile Matrix Multiply Unit (TMUL) (i.e., a separate HW co-processor)



[source WikiChip]

Tile multiplication with register file

- Any M, K dymensions as soon as the sum of K+M fits in 1KiB and fits in the TILECFG reg file (16x64 bytes)
- $Tile_C[M][N] = Tile_A[M][k] * Tile_B[k][N]$

Downsides?

Cons

- Only limited reuse / parameter (e.g., 4x in Tensor Core) $\rightarrow$ limits achievable A_I

Downsides?

Cons

- Only limited reuse / parameter (e.g., 4x in Tensor Core) $\rightarrow$ limits achievable A_I
- Works well for 2D matrices (FC), but low efficiency otherwise (e.g. CNN), and if no batching (B=1)

Downsides?

Cons

- Only limited reuse / parameter (e.g., 4x in Tensor Core) $\rightarrow$ limits achievable A_I
- Works well for 2D matrices (FC), but low efficiency otherwise (e.g. CNN), and if no batching (B=1)
- What about CNN workloads? still it not much efficient, because of memory usage...

Downsides?

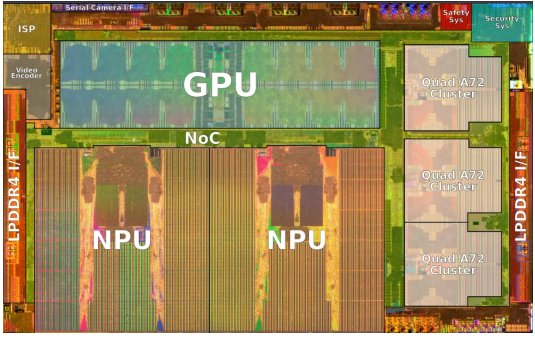
Cons

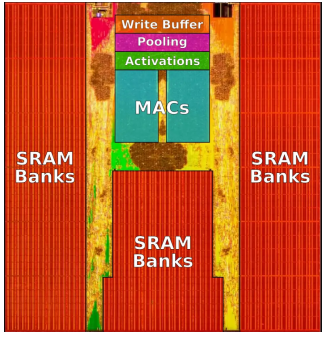
- Only limited reuse / parameter (e.g., 4x in Tensor Core) $\rightarrow$ limits achievable A_I
- Works well for 2D matrices (FC), but low efficiency otherwise (e.g. CNN), and if no batching (B=1)
- What about CNN workloads? still it not much efficient, because of memory usage...

Then how can we have more than 4X data reuse of input and weights?

We need specialized chips!

Tesla FSD chip





Tesla made its own processors

- CPUs, GPUs and Neural Processing Units
- NPU is a 96x96 MAC array (almost 10000 MACs/Cc/NPU)
- [source WikiChip]

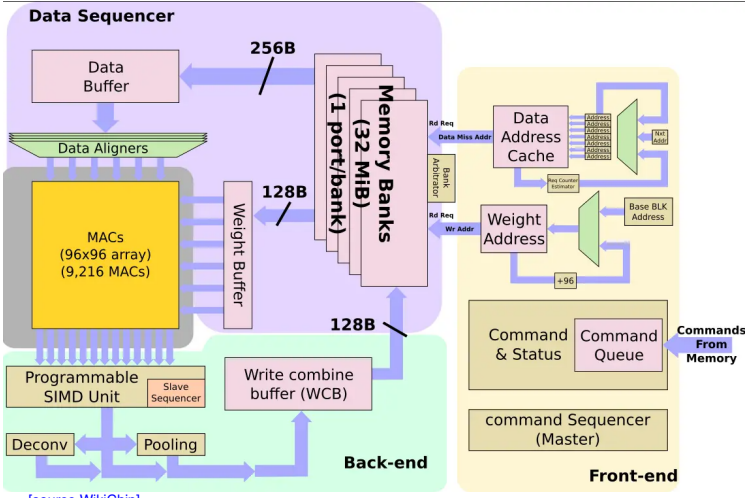
Tesla NPU

```
for(c=0 to C-1); //for each input channel
 for(fx=0 to FX-1); //for each kernel row
  for(fy=0 to FY-1); //for each kernel column
   parfor(k=0 to K-1); //for each output channel
    parfor(x=0 to X-1); // for each in/out column
     parfor(y=0 to Y-1); // for each in/out row
      o[b][k][x][y] += i[b][c][x+fx][y+fy]*w[k][c][fx][fy]
```

Tesla NPU

- Every cycles 2*96 new inputs, doing 9216 MACs!
- using GeMM to execute CNN
- No batching, but for real time applications!
- input reuse, weight reuse, but no output reuse

Tesla NPU micro-architecture



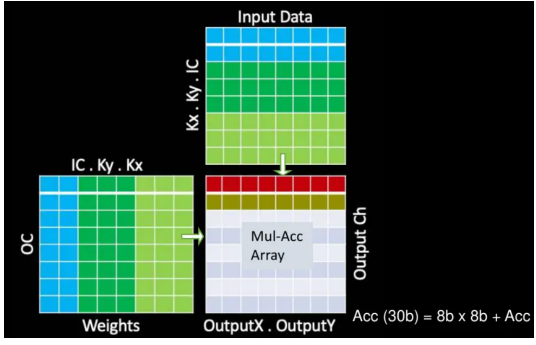
[source WikiChip]

- Huge memory
- Wide busses
- Flexible post-processing (pooling, normalization, ReLu, ...)

Tesla NPU micro-architecture

Tesla NPU

Each cycle, one row of the input data and one column of the weights are broadcasted across the entire MAC array. On the next cycle, the input data is pushed down by a row while the weights grid is pushed to the right by a row. The process is repeated with the bottom-most row of the input data and the right-most column of the weights getting broadcasted across the array. At the conclusion of a full dot product convolution, the MAC array is shifted down a row of 96 elements at a time which is also the throughput of the SIMD unit.



[source WikiChip]

Tesla NPU micro-architecture

Downsides?

Tesla NPU micro-architecture

Downsides?

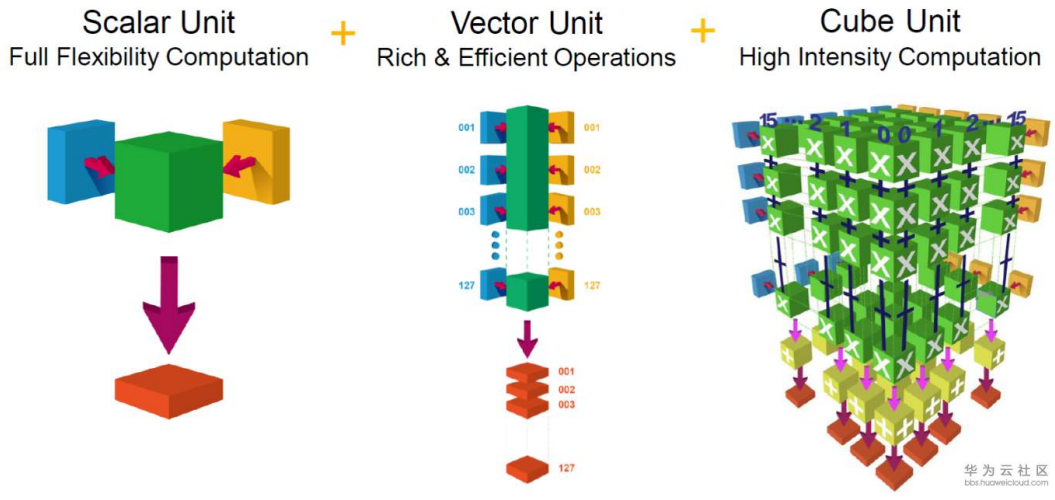
- Still (only) 2D multiplication grid

Tesla NPU micro-architecture

Downsides?

- Still (only) 2D multiplication grid
- Low utilization if we have less/more than 96 inputs or channels.(i.e., if we do not perfectly fit)

Huawei DaVinci: 3D spatial unrolling



[Da Vinci Huawei]

Huawei DaVinci: 3D spatial unrolling

```
for(b1=0 to B/16-1); //for each image/pixel in the batch
 for(k1=0 to K/16-1); //for each output channel
  for(c1=0 to C/16-1); //for each input channel
   parfor(b2=0 to 15); //for each image in the batch
    parfor(k2=0 to 15); // for each output channel
     parfor(c2=0 to 15); // for each input channel
      o[b][k] += i[b][c]*w[k][c]
```

Functional, it is a 3D operating units

- It results in many operations 16*16*16 = 4096 MAC, yet good utilization for typical C,K,..
- 3*(16*16)+256 outputs = 1024
- $A_I = 4096/1024 = 4$!
- It combats underutilization (if you have multiple of 16 in all dimensions)

GeMM & CNN processors enhancements in ASIC, CPU, GPU

GeMM & CNN processors enhancements in ASIC, CPU, GPU

- Parallelization (spatial unrolling optimization)

Conclusions

- We looked at basic concepts (weight, input, output reuse)
- People use hybrid version of these
- There isn't a single one that is better than another...
- Questions ?

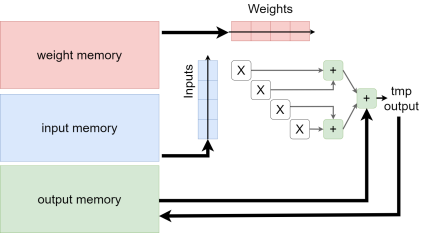
Outline

GeMM & CNN processors enhancements in ASIC, CPU, GPU

GeMM

- Parallelization (spatial unrolling optimization)
- Stationarity (temporal unrolling optimization)
- Extending spatial and temporal unrolling to higher levels

Spatial baseline to explain temporal reuse

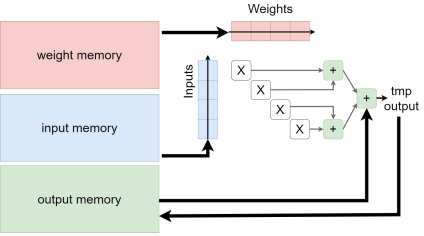



```
for(b=0 to B-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(c2=0 to C/S-1); //for each input channel
   parfor(c1=0 to S-1);
     o[b][k] += i[b][c2*S+c1]*w[c2*S+c1][k]
   // (#S multiply accumulate block in a chip)
```

Assume we are in output reuse case..

- What for loop should I put before my parfor?

Spatial baseline to explain temporal reuse

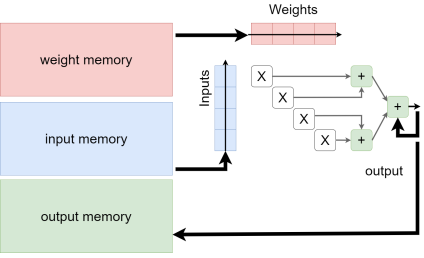



```
for(b=0 to B-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(c2=0 to C/S-1); //for each input channel
   parfor(c1=0 to S-1);
     o[b][k] += i[b][c2*S+c1]*w[c2*S+c1][k]
   // (#S multiply accumulate block in a chip)
```

Assume we are in output reuse case..

- What for loop should I put before my parfor?
- Why is this important? This will decide which data needs to be replaced and which data stays in.

Spatial baseline to explain temporal reuse

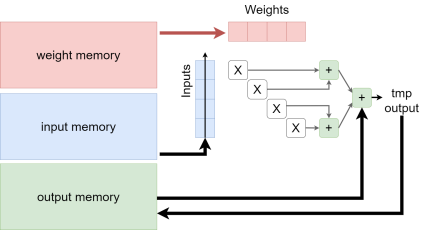



```
for(b=0 to B-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(c2=0 to C/S-1); //for each input channel
   parfor(c1=0 to S-1);
     o[b][k] += i[b][c2*S+c1]*w[c2*S+c1][k]
   // (#S multiply accumulate block in a chip)
```

Assume we are in output reuse case..

- What for loop should I put before my parfor?
- Why is this important? This will decide which data needs to be replaced and which data stays in.
- Because my output can stay stationary, i.e. (output stationary).

Reuse IA and W data: weight stationary

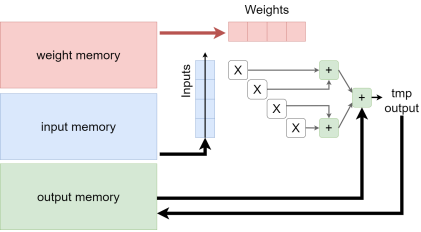



```
for(k=0 to K-1); //for each output channel
 for(c2=0 to C/P-1); //for each input channel
  for(b=0 to B-1); //for each image in the batch
   parfor(c1=0 to S-1);
    o[b][k] += i[b][c2*P+c1]*w[c2*P+c1][k]
    // (#S multiply accumulate block in a chip)
```

Assume we are in output reuse case..

- I have now put the B loop before the parfor (i.e., w does not depend on B)

Reuse IA and W data: weight stationary

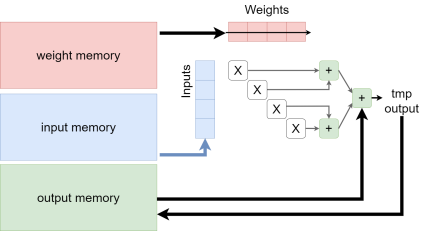



```
for(k=0 to K-1); //for each output channel
 for(c2=0 to C/P-1); //for each input channel
  for(b=0 to B-1); //for each image in the batch
   parfor(c1=0 to S-1);
    o[b][k] += i[b][c2*P+c1]*w[c2*P+c1][k]
    // (#S multiply accumulate block in a chip)
```

Assume we are in output reuse case..

- I have now put the B loop before the parfor (i.e., w does not depend on B)
- The weights can stay where they are for B-1 cycles (weight stationary)

Reuse IA and W data: input stationary

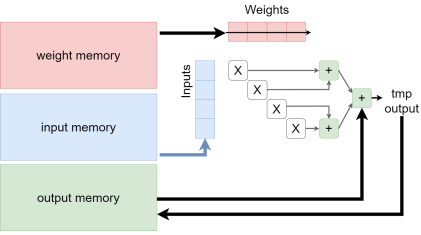



```
for(c2=0 to C/P-1); //for each input channel
 for(b=0 to B-1); //for each image in the batch
  for(k=0 to K-1); //for each output channel
   parfor(c1=0 to P-1);
    o[b][k] += i[b][c2*P+c1]*w[c2*P+c1][k]
```

Assume we are in output reuse case..

- I have now put the K loop before the parfor (i.e., i does not depend on K)

Reuse IA and W data: input stationary



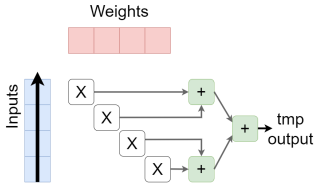

```
for(c2=0 to C/P-1); //for each input channel
 for(b=0 to B-1); //for each image in the batch
  for(k=0 to K-1); //for each output channel
   parfor(c1=0 to P-1);
    o[b][k] += i[b][c2*P+c1]*w[c2*P+c1][k]
```

Assume we are in output reuse case..

- I have now put the K loop before the parfor (i.e., i does not depend on K)
- The inputs can stay where they are for K-1 cycles (input stationary)

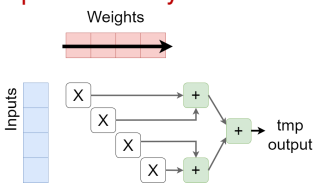
Impact on temporal data reuse T (output reuse baseline)

weight stationary



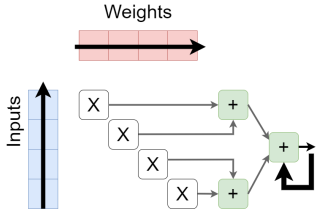
- weight BW = S/T
- input BW = S
- output BW = 1
- $A_I(SRAM)$ = S / (S/T+S+1)

input stationary



- weight BW = S
- input BW = S/T
- output BW = 1
- $A_I(SRAM)$ = S / (S+S/T+1)

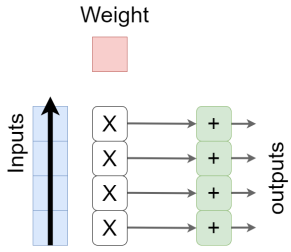
output stationary



- weight BW = S
- input BW = S
- output BW = 1/T
- $A_I(SRAM)$ = S / (2S+1/T)

Impact on temporal data reuse T (weight reuse baseline)

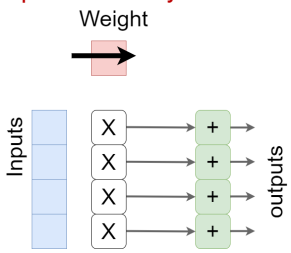
weight stationary



- weight BW = 1/T
- input BW = S
- output BW = S
- $A_I(SRAM)$ = S / (1/T+2S)

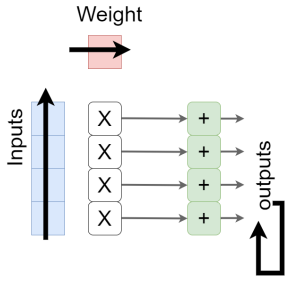
many hybrid exists!

input stationary



- weight BW = 1
- input BW = S/T
- output BW = S
- $A_I(SRAM)$ = S / (1+S/T+S)

output stationary



- weight BW = 1
- input BW = S
- output BW = S/T
- $A_I(SRAM)$ = S / (1+S+S/T)

Base reuse combinations

Question

- Can I combine weight-stationary, input-stationary, and output stationary all together?

Base reuse combinations

Question

- Can I combine weight-stationary, input-stationary, and output stationary all together?
- No, because some input data has to change, otherwise what are we computing new?

Base reuse combinations

Question

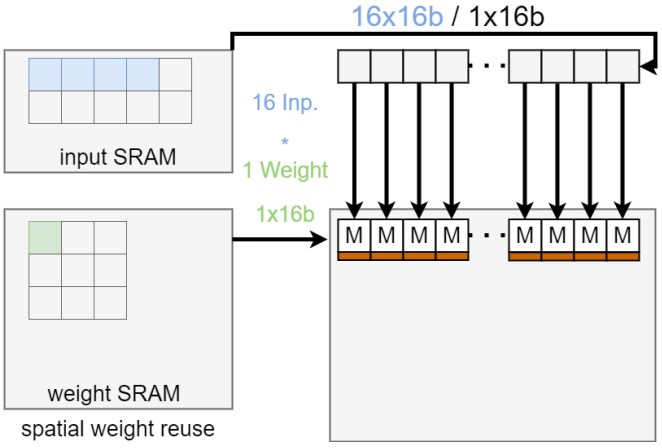
- Can I combine weight-stationary, input-stationary, and output stationary all together?
- No, because some input data has to change, otherwise what are we computing new?
- It is all about the inner level of the for loop before the parfor... we cannot just combine all of them.

Diving into state-of-the-art accelerators II

SOTA accelerators

- Envision (KU) Leuven
- Tesla NPU
- Systolic Arrays Google TPUs

Envision processor (KU) Leuven

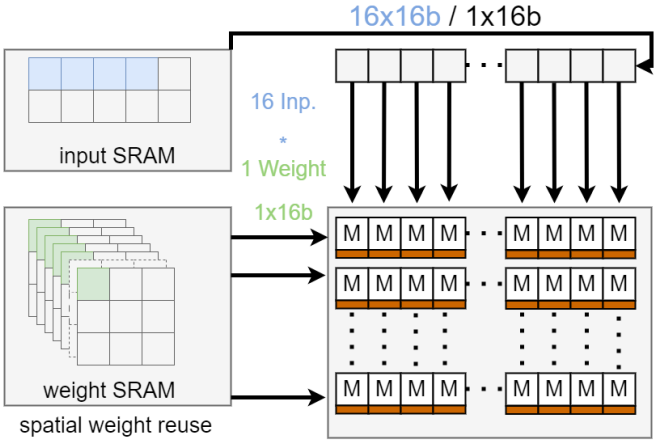


[Moons B. et al, ISSCC, 2017]

What type of reuse?

- spatial weight reuse

Envision processor (KU) Leuven

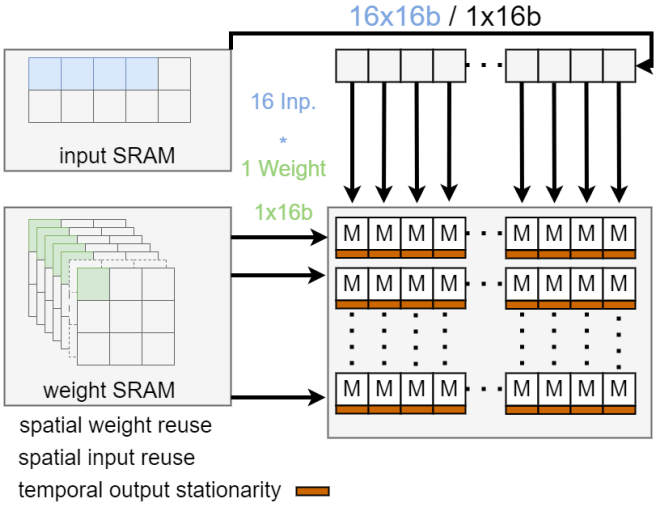


[Moons B. et al, ISSCC, 2017]

What type of reuse?

- spatial weight reuse
- spatial input reuse

Envision processor (KU) Leuven

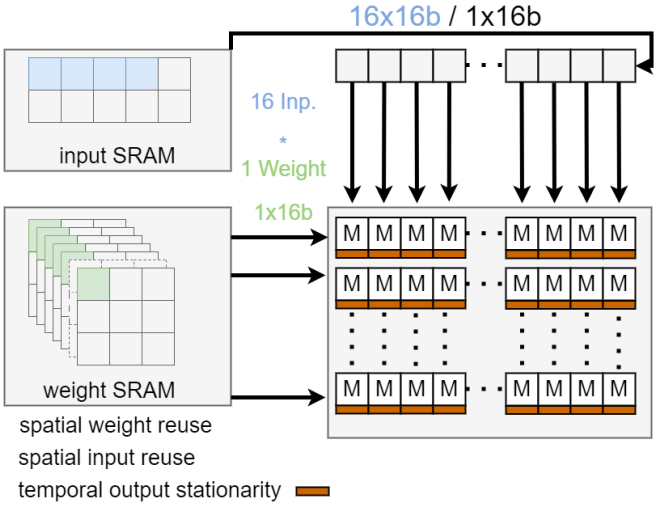


[Moons B. et al, ISSCC, 2017]

What type of reuse?

- spatial weight reuse
- spatial input reuse
- temporal output stationary (RF)

Envision processor (KU) Leuven



[Moons B. et al, ISSCC, 2017]

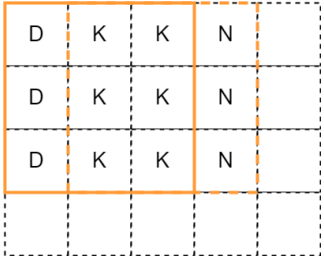
What type of reuse?

- spatial weight reuse
- spatial input reuse
- temporal output stationary (RF)
- 256 (16*16) multiplications/cycle for only 32 memory fetches
  $A_I = 256/32 = 8$

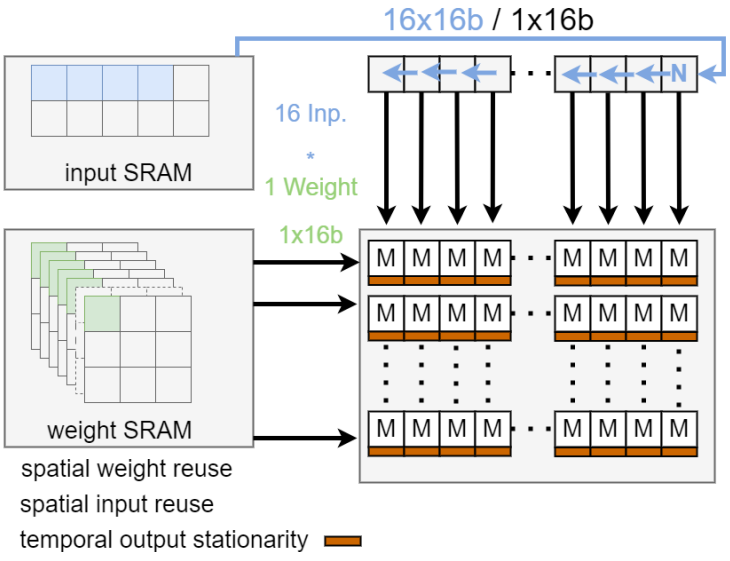
Envision processor (KU) Leuven: input data reuse in convolutions

```
for(y=0 to Y-1); //for each input column
 for(c=0 to C-1); //for each input channel
  for(fy=0 to FY-1); //for each kernel column
   for(fx=0 to FX-1); //for each kernel row
    parfor(k=0 to K-1); // for each output channel
     parfor(x=0 to X-1); // for each input row
      o[b][k][x][y] += i[b][c][x+fx][y+fy]*w[k][c][fx][fy]
```

- D = Discard
- K = Keep stationary
- N = new fetch
- GeMM is typically output stationary



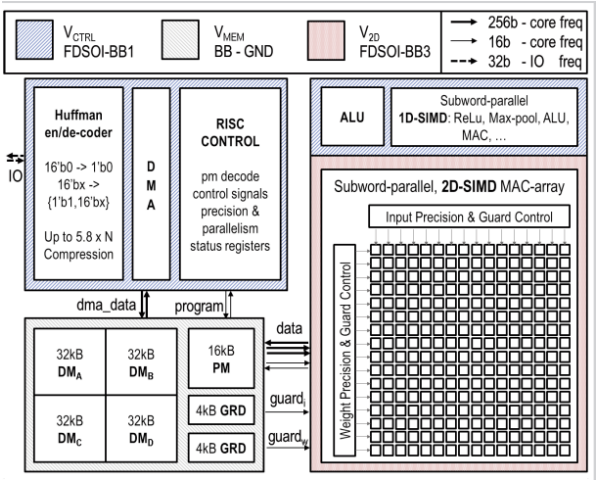
Envision processor (KU) Leuven



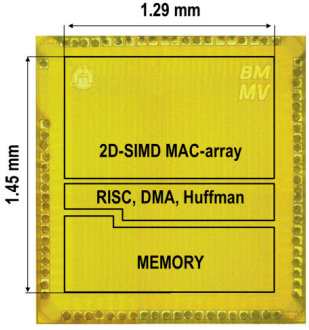
What type of reuse?

- spatial weight reuse
- spatial input reuse
- temporal output stationary (RF)
- pseudo input stationary, inputs stay in local RF and shift! (multi-level input stationarity)
- $A_I = 256/17 \sim 15$

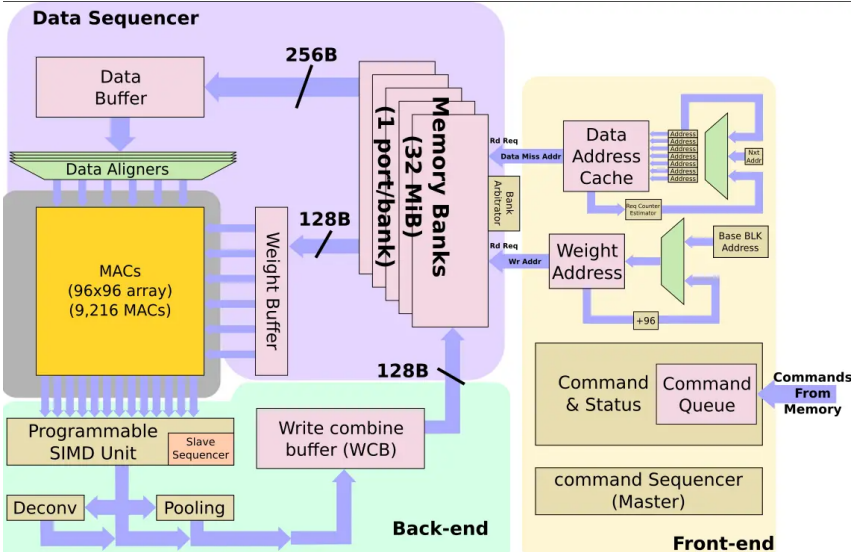
Envision processor (KU) Leuven



[Moons B. et al, ISSCC, 2017]



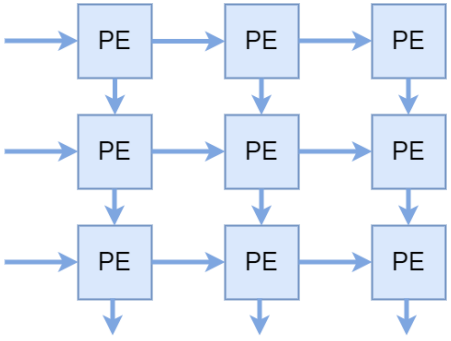
Tesla FSD, also exploit temporal stationarity



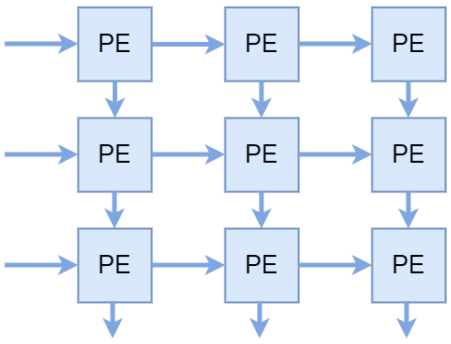
Data aligners

- They retain and shift data from one clock cycle to the next

What are systolic arrays?



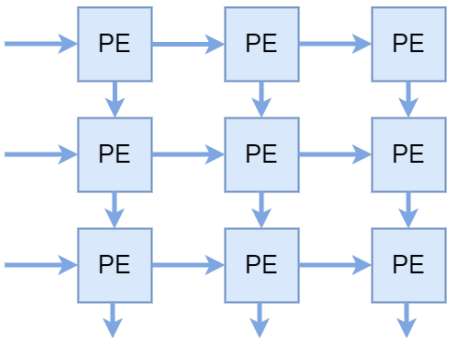
What are systolic arrays?



Definition: Systolic Array

Systolic arrays are hardware structures built for fast and efficient operation of regular algorithms that perform the same task with different data at different time instants.

What are systolic arrays?



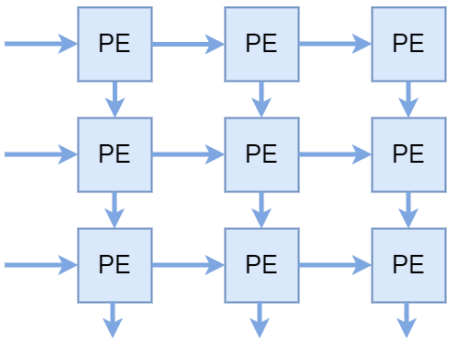
Definition: Systolic Array

Systolic arrays are hardware structures built for fast and efficient operation of regular algorithms that perform the same task with different data at different time instants.

Why?

- Simple, regular design (keep num. of parts small and regular)

What are systolic arrays?



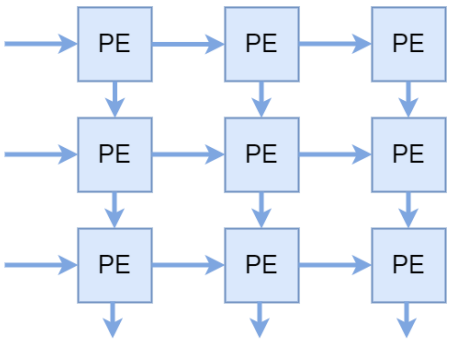
Definition: Systolic Array

Systolic arrays are hardware structures built for fast and efficient operation of regular algorithms that perform the same task with different data at different time instants.

Why?

- Simple, regular design (keep num. of parts small and regular)
- High-concurrency $\rightarrow$ (high-performance)

What are systolic arrays?



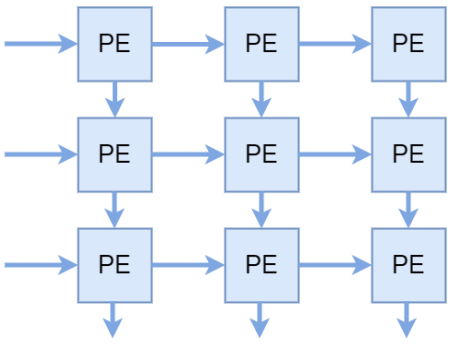
Definition: Systolic Array

Systolic arrays are hardware structures built for fast and efficient operation of regular algorithms that perform the same task with different data at different time instants.

Why?

- Simple, regular design (keep num. of parts small and regular)
- High-concurrency $\rightarrow$ (high-performance)
- Balance computation and I/O (memory) bandwidth

What are systolic arrays?



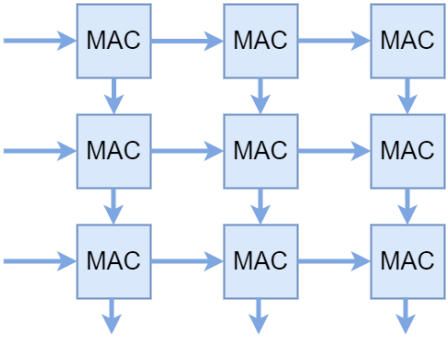
Definition: Systolic Array

Systolic arrays are hardware structures built for fast and efficient operation of regular algorithms that perform the same task with different data at different time instants.

Why?

- Simple, regular design (keep num. of parts small and regular)
- High-concurrency $\rightarrow$ (high-performance)
- Balance computation and I/O (memory) bandwidth

Systolic MAC arrays operations (Google TPU)



Exploit local buffers to:

- pass in/out to neighboring MACs in next clock cycle
- allows doing efficient matrix multiplication
- typically weight (or row) stationary

Pipeline everything

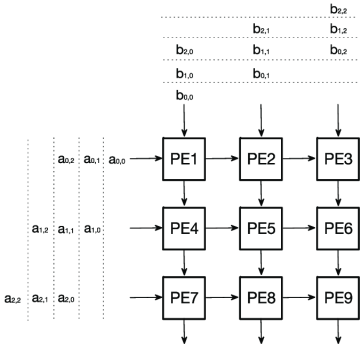
- they can scale to very large numbers of PEs
- no need to drive data in long lines (capacitive load is a problem for buffers, e.g., Envision)

Systolic Arrays: GeMM

Example, how to organize the computation? how to multiply a 3x3 matrix?

Each PE does three operations:

- multiplies $a \cdot b$
- adds the result to the previous partial result (i.e., c) and stores it locally
- sends the incoming data to the PEs at its right and bottom

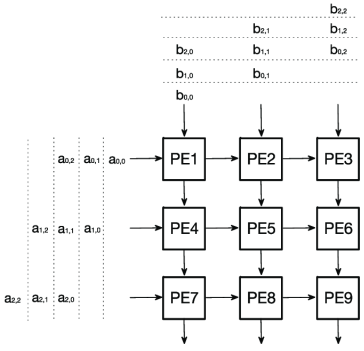


Systolic Arrays: GeMM

Example, how to organize the computation? how to multiply a 3x3 matrix?

Each PE does three operations:

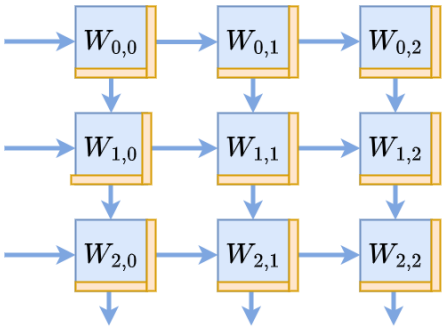
- multiplies $a \cdot b$
- adds the result to the previous partial result (i.e., c) and stores it locally
- sends the incoming data to the PEs at its right and bottom



Let's see it in action!

Systolic array slide data in an array of programmable elements (PEs) that perform the same operation simultaneously. [Animation Video]

Systolic Arrays: Google TPU



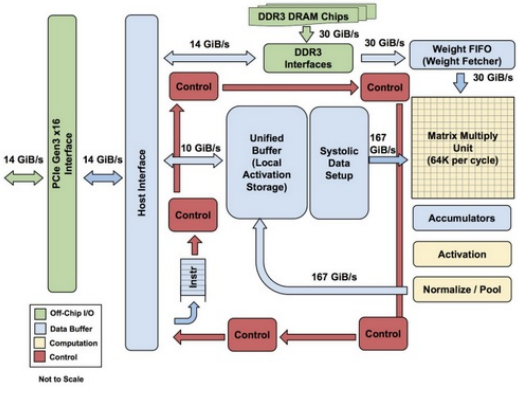
Summary

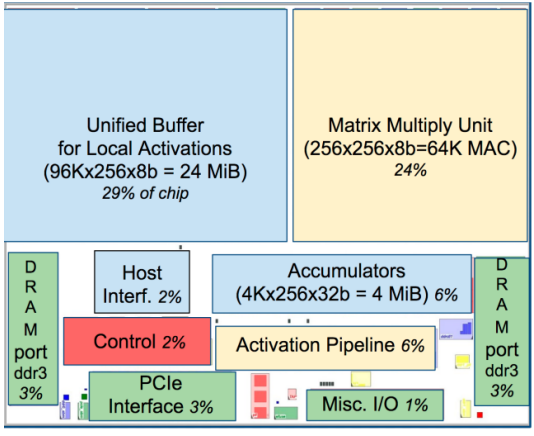
- load all weights in PEs
- multiplies $a \cdot b$
- adds the result to the previous partial result (i.e., c) and stores it locally
- sends the incoming data to the PEs at its right and bottom

Pros and Cons

- + very high throughput (register to register, no memory fetched)
- - registers are constantly switching

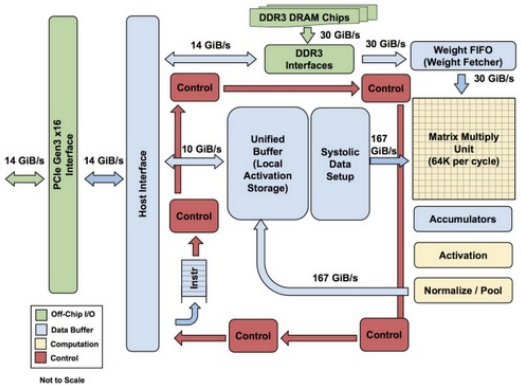
Google TPU architecture





- Exploit systolic array to the extreme (64k MAC / clock cycle)
- TPU = tensor processing unit

Google TPU Architecture

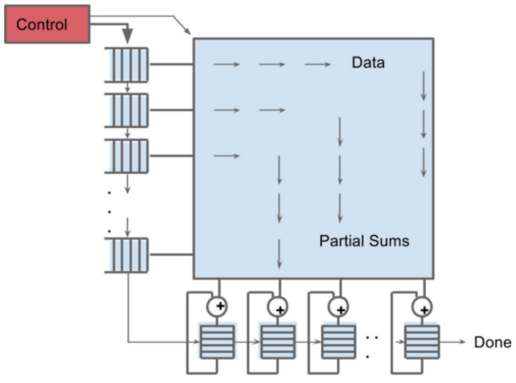


[Jouppi et al., MICRO 2017]

Google TPU v1

- 65,536 MAC units (256x256 systolic array)
- 28 MiB on-chip memory
- Clock frequency: 700 MHz
- Power consumption: 75 W
- Throughput: 92 TOPS (8-bit integer)
- Energy efficiency: **1.23 TOPS/W**

Google TPU architecture



[Google TPU Blog.]

- **MXU** (Matrix Accumulate Unit) is a systolic array that run large-scale matrix multiply accumulate
- One Unit has a systolic array that contains $256x256 = 65,536$ (total) ALUs.
- Can process 65,536 multiply-and-adds for 8-bit integers every cycle.
- Given clock frequency 700MHz, a TPU can compute $65,536700,000,000 = 461012$ multiply-and-add operations or 92 Teraops per second (92 x 1012)

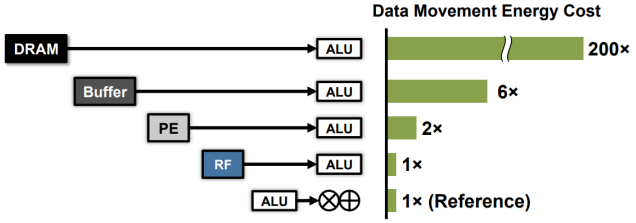
Outline

GeMM & CNN processors enhancements in ASIC, CPU, GP

GeMM & CNN processors enhancements in ASIC, CPU, GPU

- Parallelization (spatial unrolling optimization)
- Stationarity (temporal unrolling optimization)
- Extending spatial and temporal unrolling to higher levels

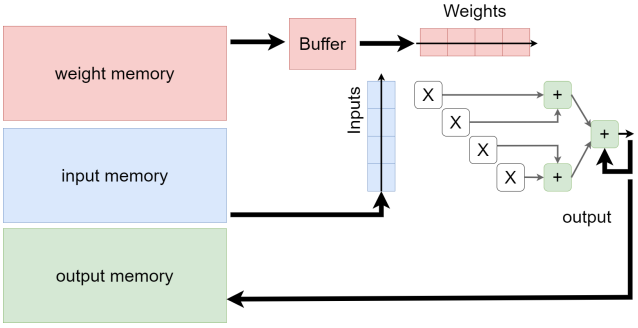
Temporal data reuse at multiple levels



We still need more!

- the amount of possible data that we can keep stationary at lowest level is limited
- yet, memory accesses to upper memory levels are expensive (still dominant energy after stationarity optimizations)
- all weights and inputs/outputs together MB's to GB's per frame!
- Only smallest networks fit in (small/cheap) on-chip SRAM memories

Multi level memories and buffers in accelerators



assume temporal output reuse & spatial output reuse

- buffer a subset of weights (locally) to be reused for 1 network layer (or channel)
- combined with smart (optimized) strategy of which channel / pixels to compute first
- maximizing temporal data locality (multi-level stationarity, or chaching)

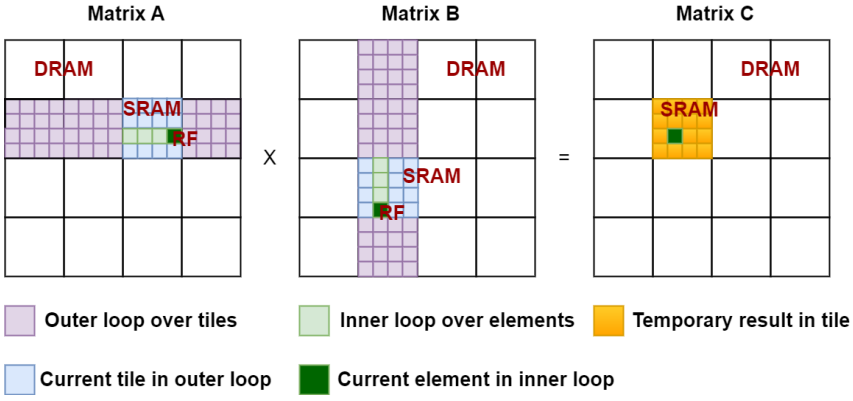
Multi level memories and buffers in accelerators

```
for(b2=0 to 3); //for each image in the batch
 for(k2=0 to 3); //for each output channel
  for(c2=0 to 3); //for each input channel
   for(b1=0 to B/4-1); //for each image in the batch
    for(k1=0 to K/4-1); // for each output channel
     for(c1=0 to C/4-1); // for each input channel
      o[b][k] += i[b][c]*w[c][k]
```

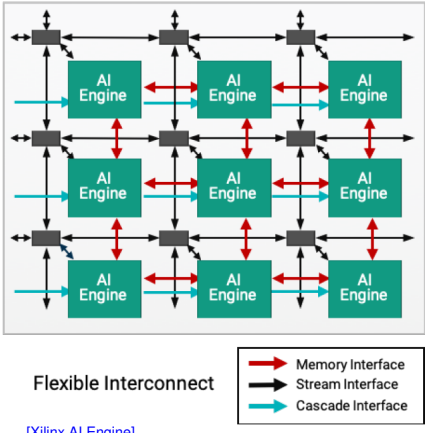
assume temporal output reuse & spatial output reuse

- Tiling and memory usage can again be analyzed using nested for loops (i.e., stationarity at higher levels)
- b2,k2,c2 loops are fed from DRAM
- b1,k1,c1, loops are fed from SRAM
- SRAM can then have some stationarity towards the DRAM (again, just loop tiling!)

Multi level memories and buffers in accelerators



Spatial data reuse at multiple levels



[Xilinx AI Engine]

Beyond the datapath level

- Parallelism can be extended beyond the datapath level
- multicore ML processors
- means that parfor loops are repeated at higher loop level

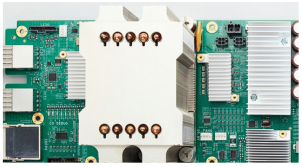
Spatial data reuse at multiple levels

```
parfor(b=0 to B-1); //for each image in the batch
 for(k=0 to K-1); //for each output channel
  for(y=0 to Y-1); //for each input channel
   parfor(c=0 to C-1); //for each image in the batch
    parfor(fx=0 to FX-1); // for each output channel
     parfor(fy=0 to FY-1); // for each input channel
      o[b][k][x][y] += i[b][c][x+fx][y+fy]*w[k][c][fx][fy]
```

Beyond the datapath level

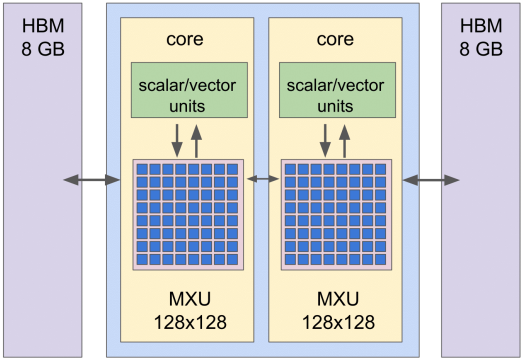
- Parallelism can be extended beyond the datapath level
- multicore ML processors
- means that parfor loops are repeated at higher loop level
- massive spatial output reuse

Google TPU v2



Google TPU v2 Chip

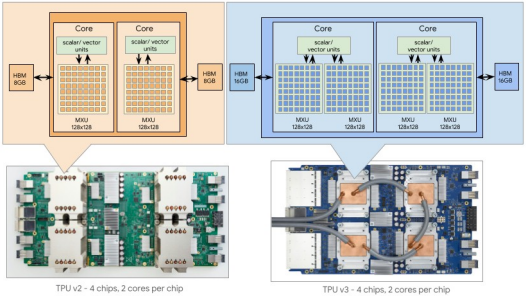
- 16Gb of HBM
- 600 GB/s mem BW
- Scalar Unit 32b float
- MXU: 32bfloat accumulation but reduced precision for multipliers
- 45TFLOPS



Google TPU v2

- multiple processing arrays per chip
- inference & training (2 systolic arrays)

Google TPU v3



TPUs

TPU 3 has 4 systolic arrays, 8x performance vs TPU2, TPU3 is liquid cooled!

Extending spatial and temporal unrolling to higher levels: conclusions

Summary

- What "parfor" loop to push to the other core?
  - other images of the batch?
  - other output channels?
  - other X-Y tiles?
- which one will result in minimal latency or energy?
- what metrics matter?
- Benefits of multicore Vs one large single core?
- Downsides of multicore vs one large single core?
- Too many approaches to cover! It is a hot research field.

Summary

GeMM & CNN processors enhancements in ASIC, CPU, GPU

- Parallelization (spatial unrolling optimization)
- Stationarity (temporal unrolling optimization)
- Extending spatial and temporal unrolling to higher levels

Next Time

- Sparsity in accelerators
- Quantization in accelerators