Design and Generation of Efficient Hardware Accelerators for Tensor Computations

Nitish Srivastava

Special Committee

Zhiru Zhang — David Albonesi
Christopher Batten — Rajit Manohar

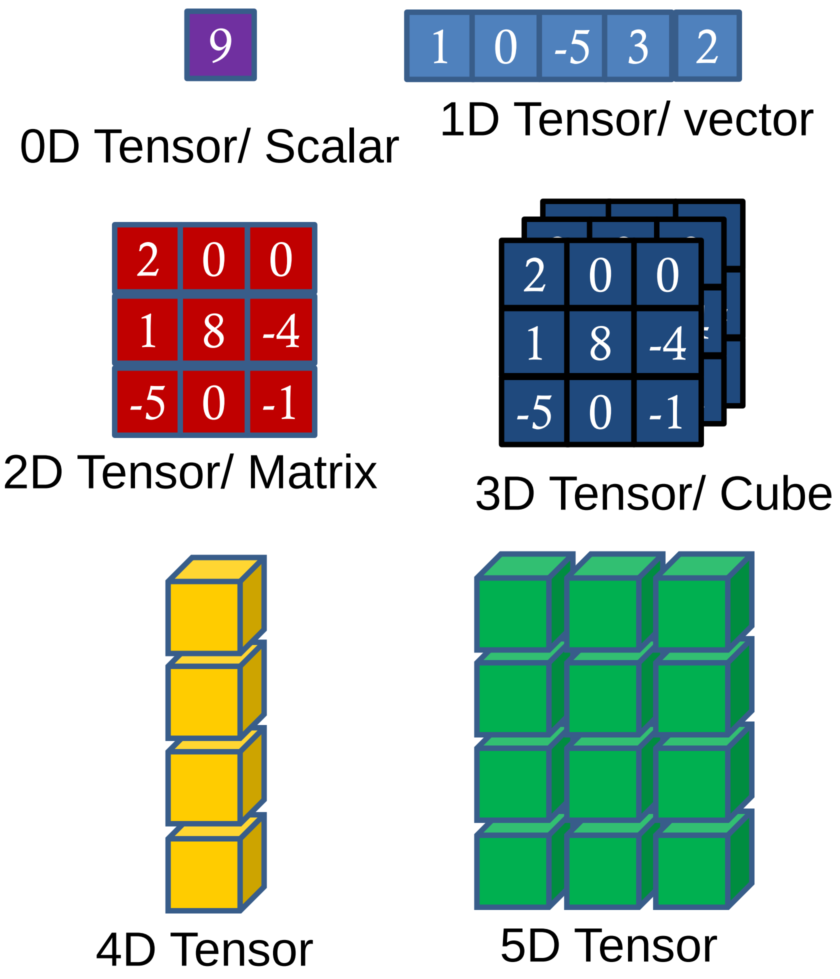
01/14/2020

School of Electrical and Computer Engineering, Cornell University

What is a Tensor?

What is a Tensor?

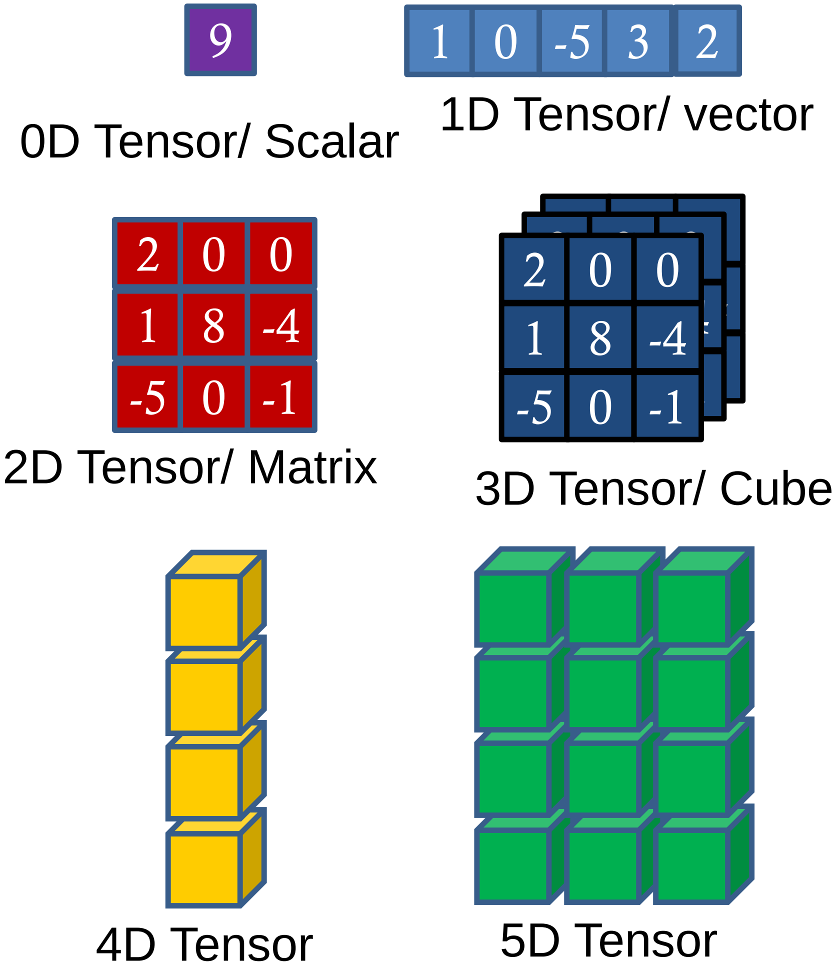
- **Tensors** are generalization of matrices to n dimensions
  - Scalar is tensor with 0 dimensions
  - Vector is tensor with 1 dimension
  - Matrix is tensor with 2 dimensions, and so on



What is a Tensor?

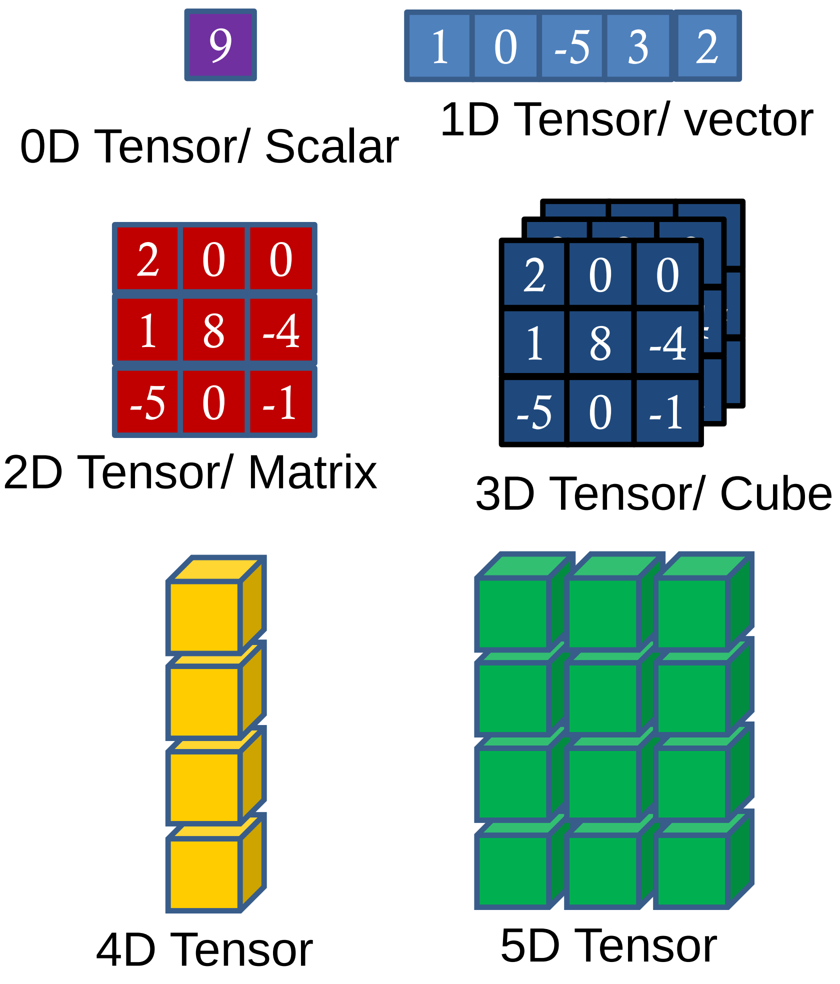
- **Tensors** are generalization of matrices to n dimensions
  - Scalar is tensor with 0 dimensions
  - Vector is tensor with 1 dimension
  - Matrix is tensor with 2 dimensions, and so on

- **Sparse tensor** is a tensor where most of its elements are zeros



What is a Tensor?

- **Tensors** are generalization of matrices to n dimensions
  - Scalar is tensor with 0 dimensions
  - Vector is tensor with 1 dimension
  - Matrix is tensor with 2 dimensions, and so on
- **Sparse tensor** is a tensor where most of its elements are zeros
- **Tensor kernels are both compute and data intensive**



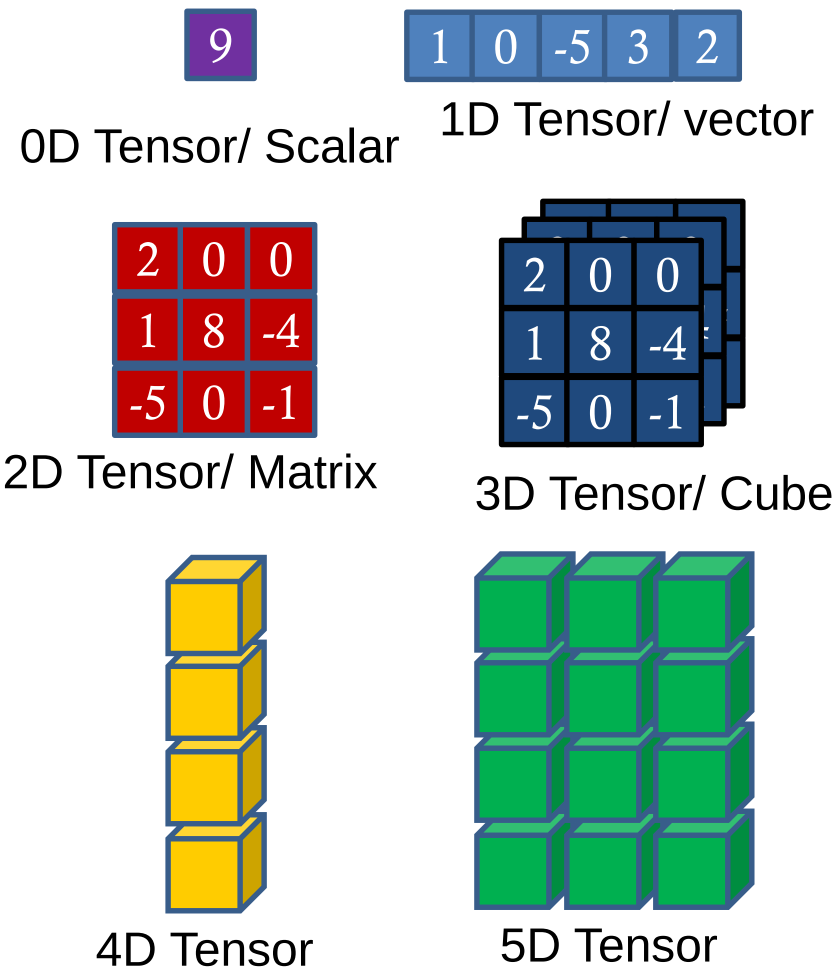
What is a Tensor?

- **Tensors** are generalization of matrices to n dimensions
  - Scalar is tensor with 0 dimensions
  - Vector is tensor with 1 dimension
  - Matrix is tensor with 2 dimensions, and so on

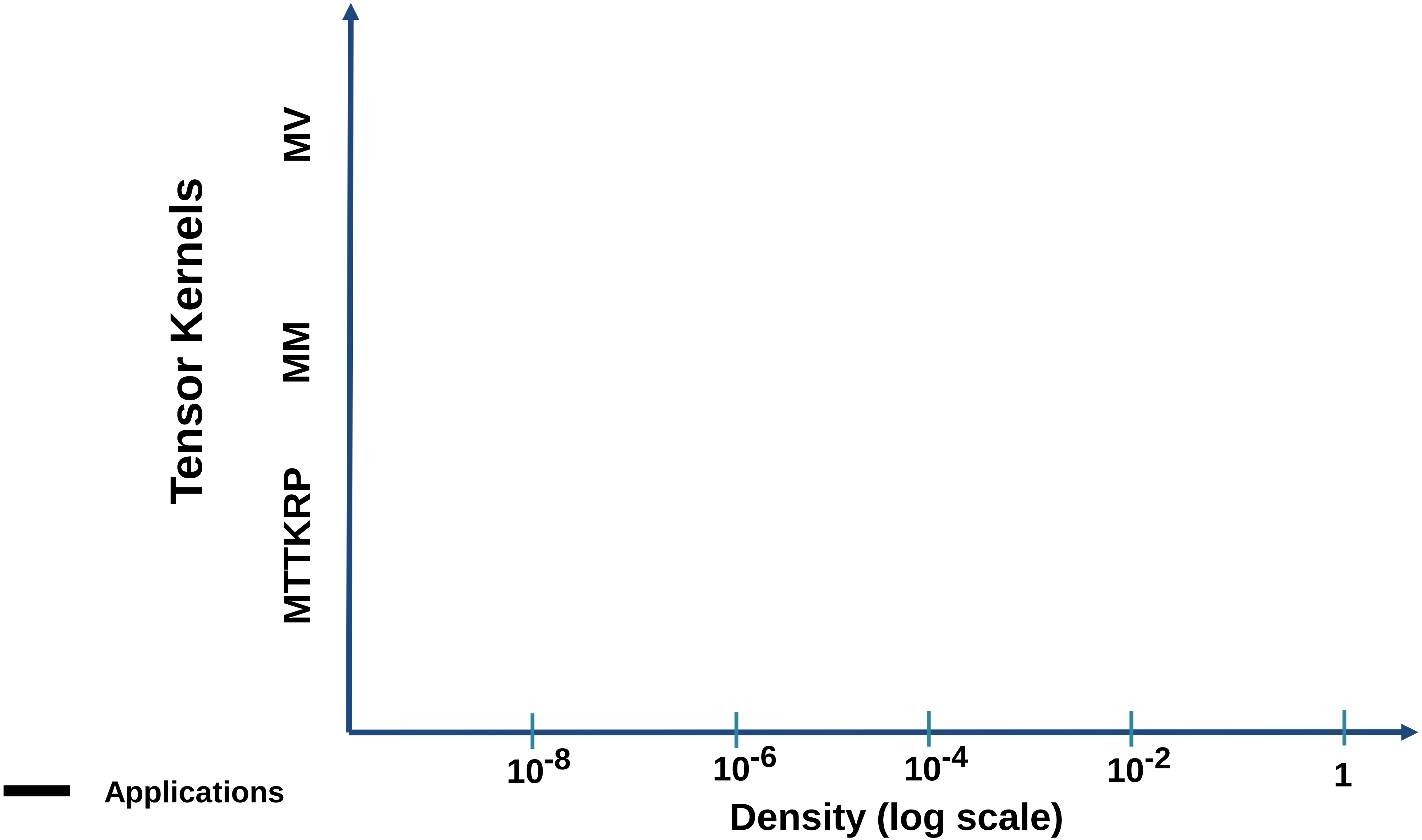
- **Sparse tensor** is a tensor where most of its elements are zeros

- **Tensor kernels are both compute and data intensive**

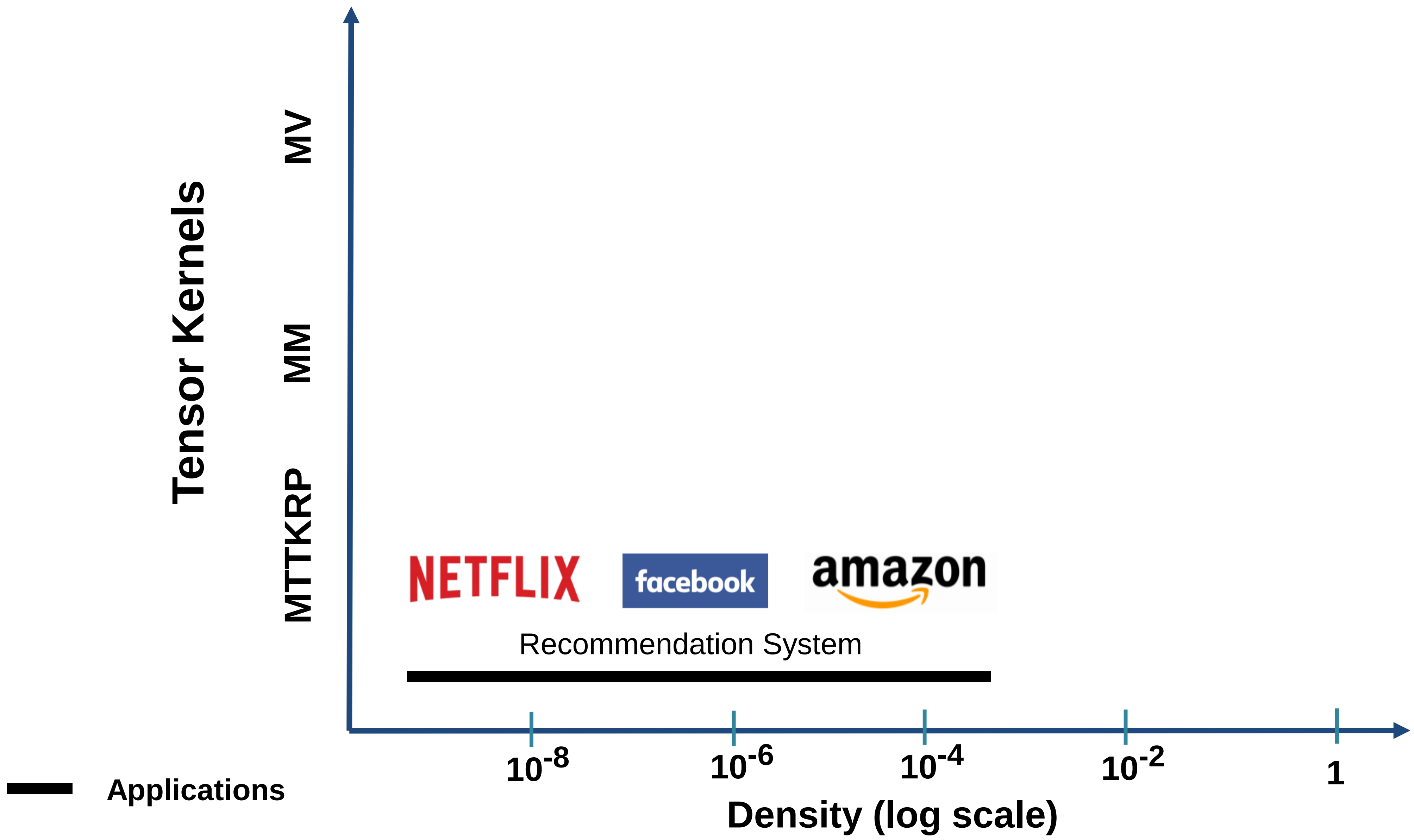
- **Popular tensor kernels**
  - Matricized Tensor Times Khatri-Rao Product (MTTKRP)
  - Matrix-Matrix Multiplication (MM)
  - Matrix-Vector Multiplication (MV), etc



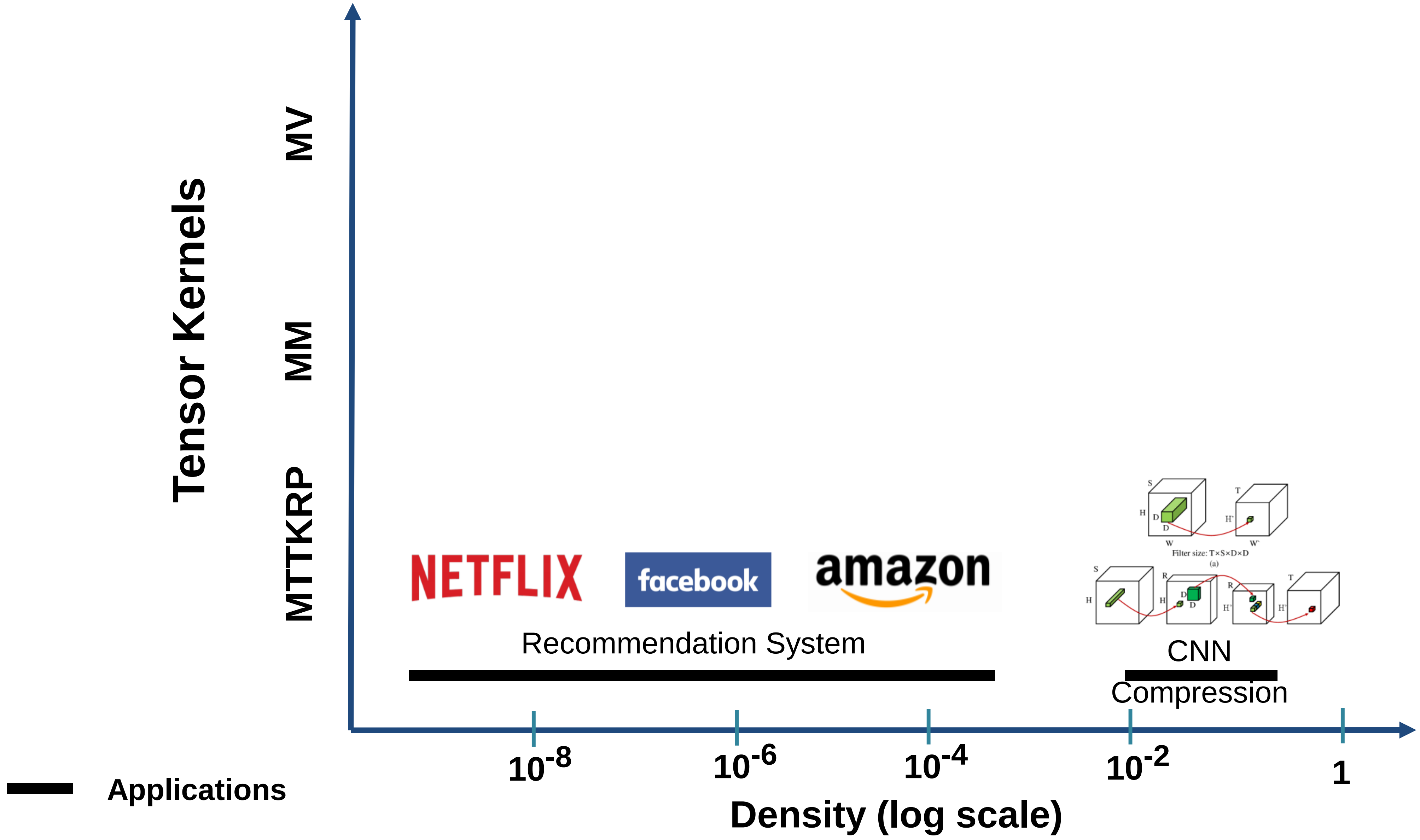
Kernel-Sparsity Spectrum of Tensor Applications



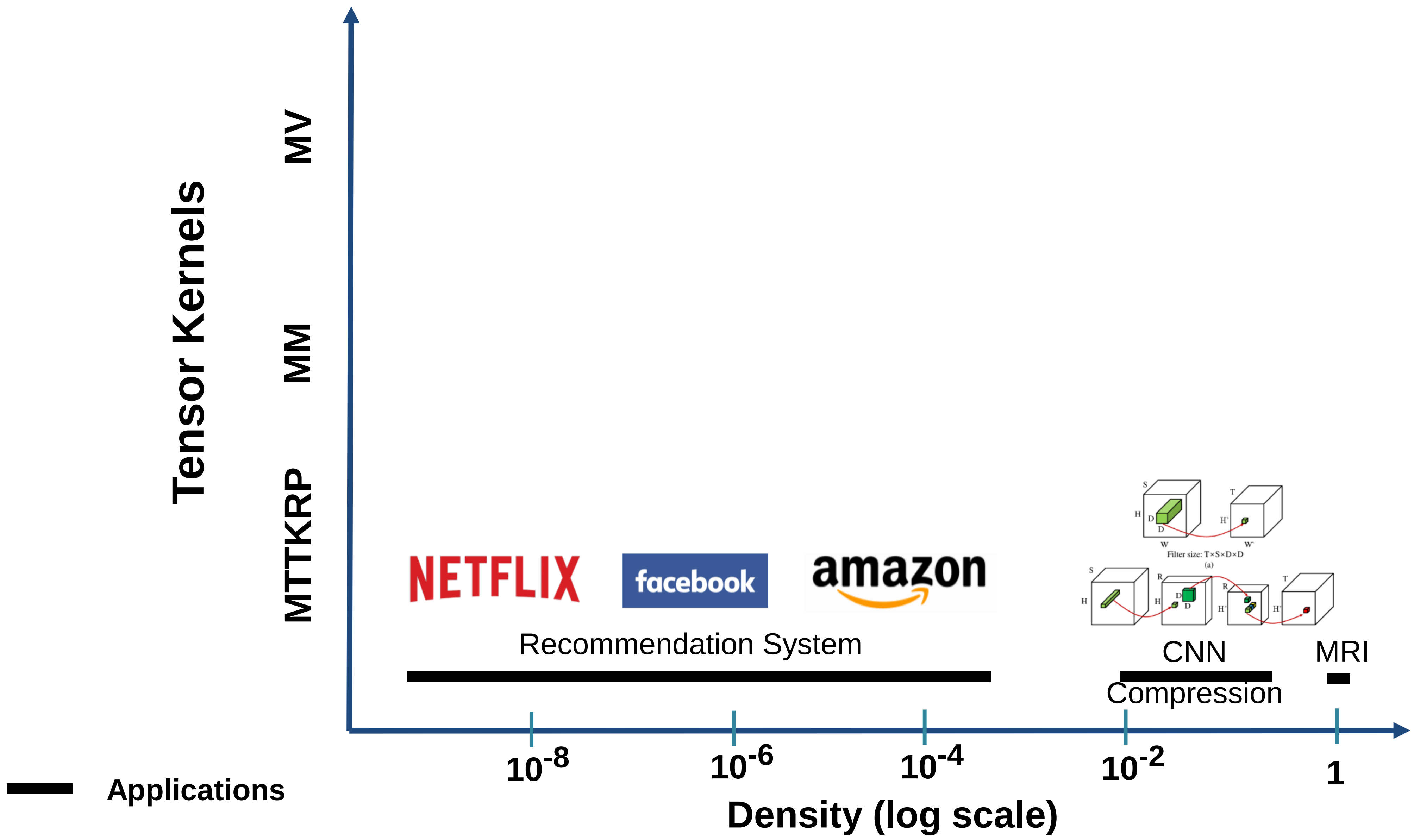
Kernel-Sparsity Spectrum of Tensor Applications



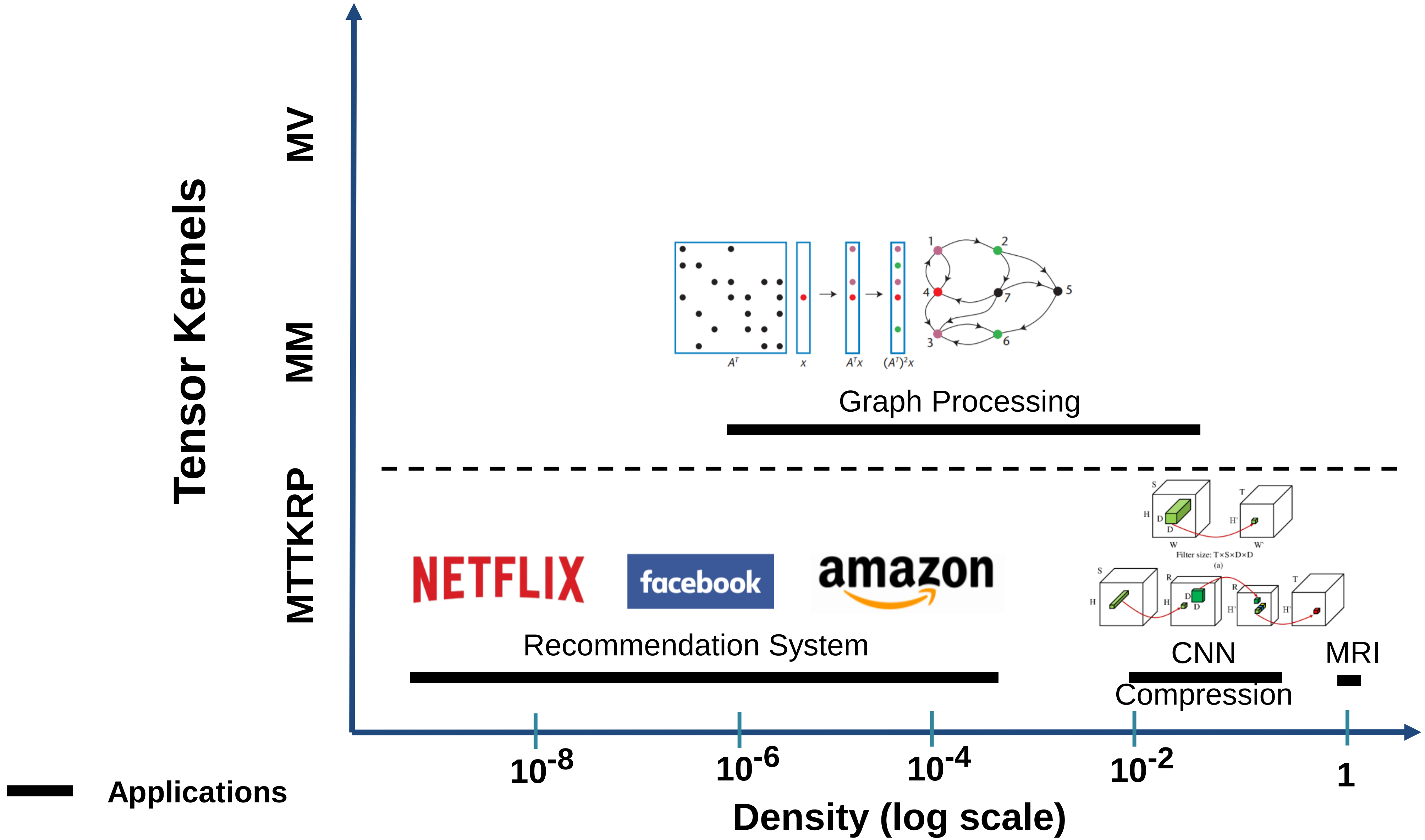
Kernel-Sparsity Spectrum of Tensor Applications



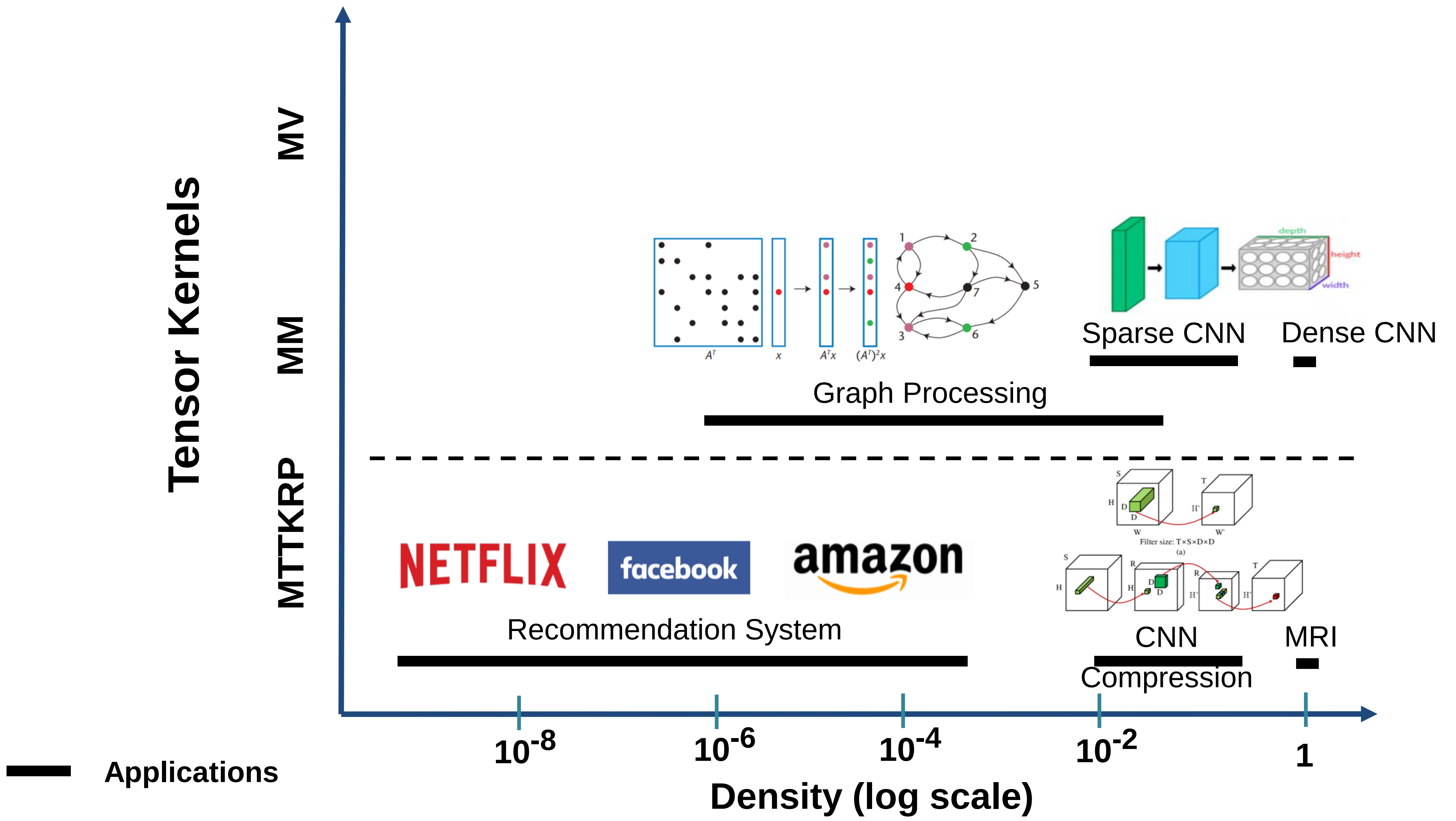
Kernel-Sparsity Spectrum of Tensor Applications



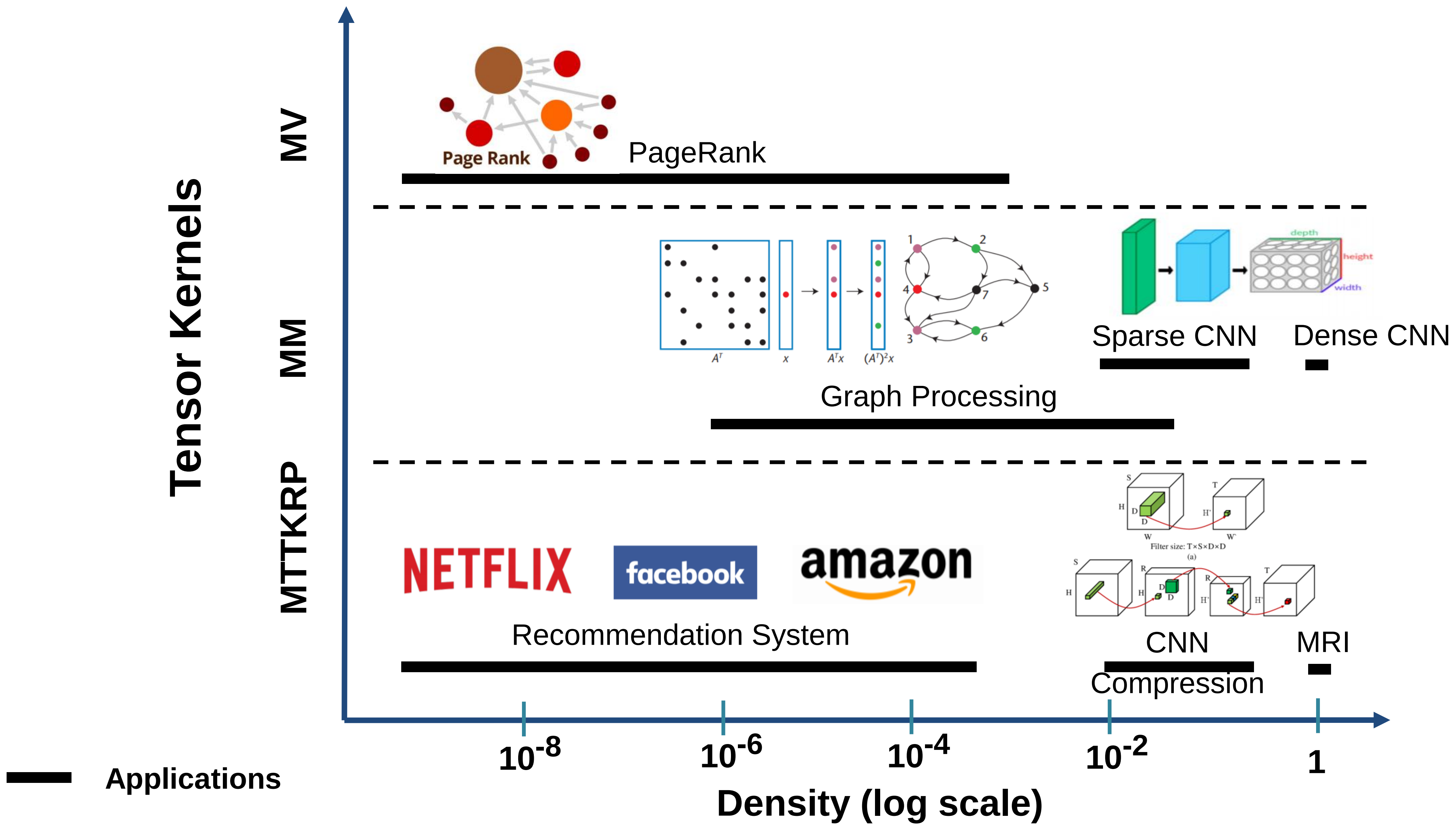
Kernel-Sparsity Spectrum of Tensor Applications



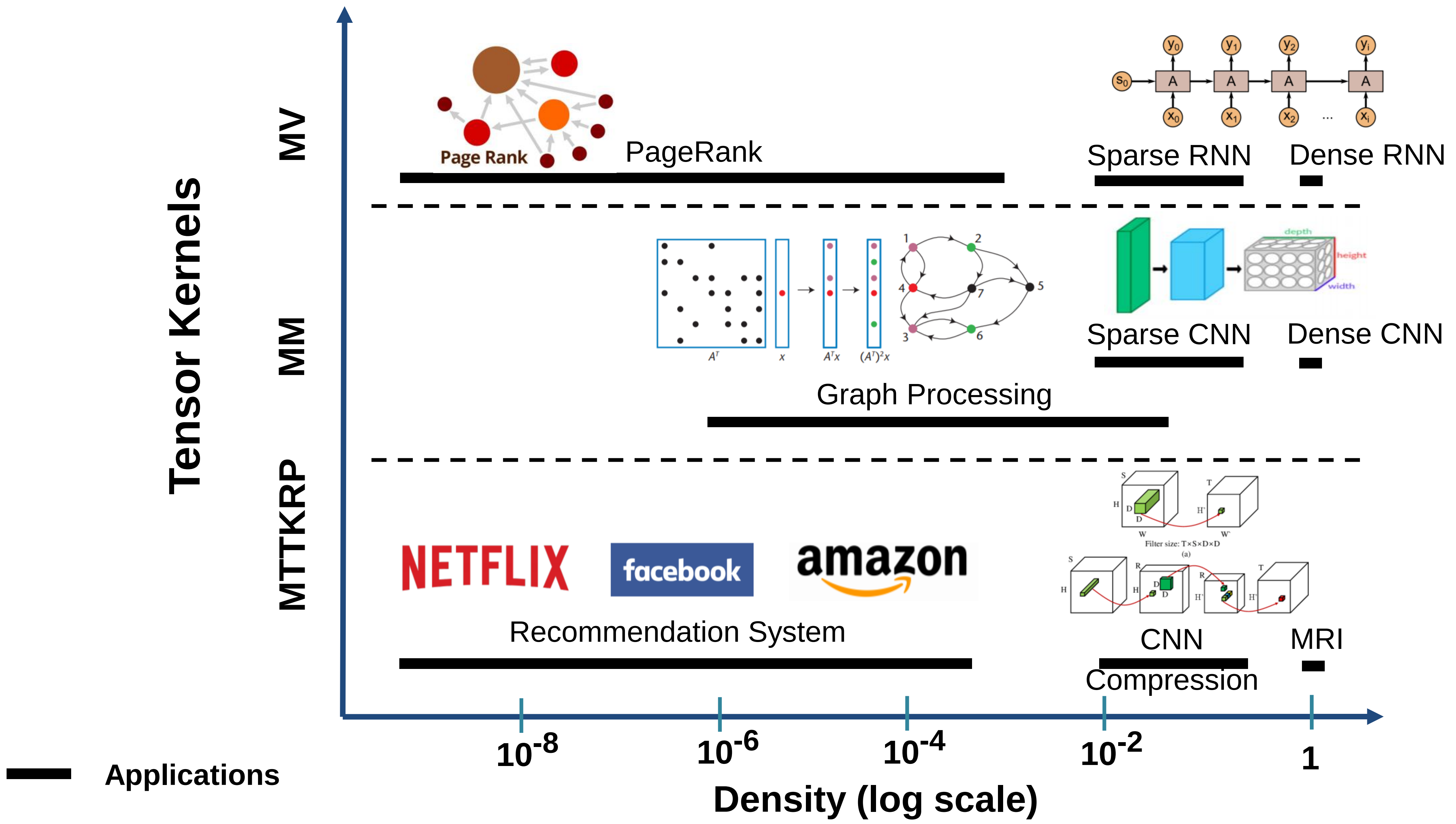
Kernel-Sparsity Spectrum of Tensor Applications



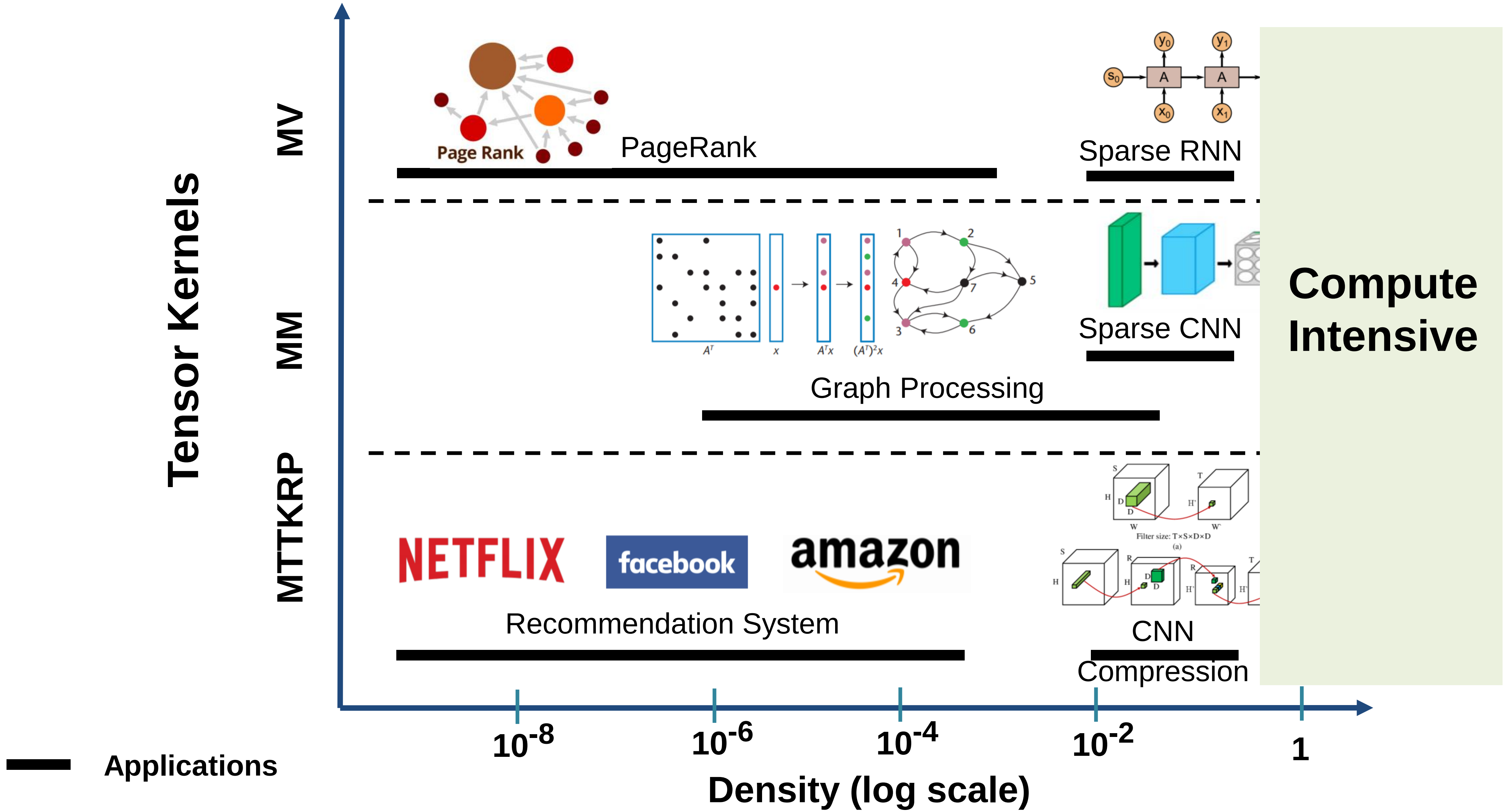
Kernel-Sparsity Spectrum of Tensor Applications



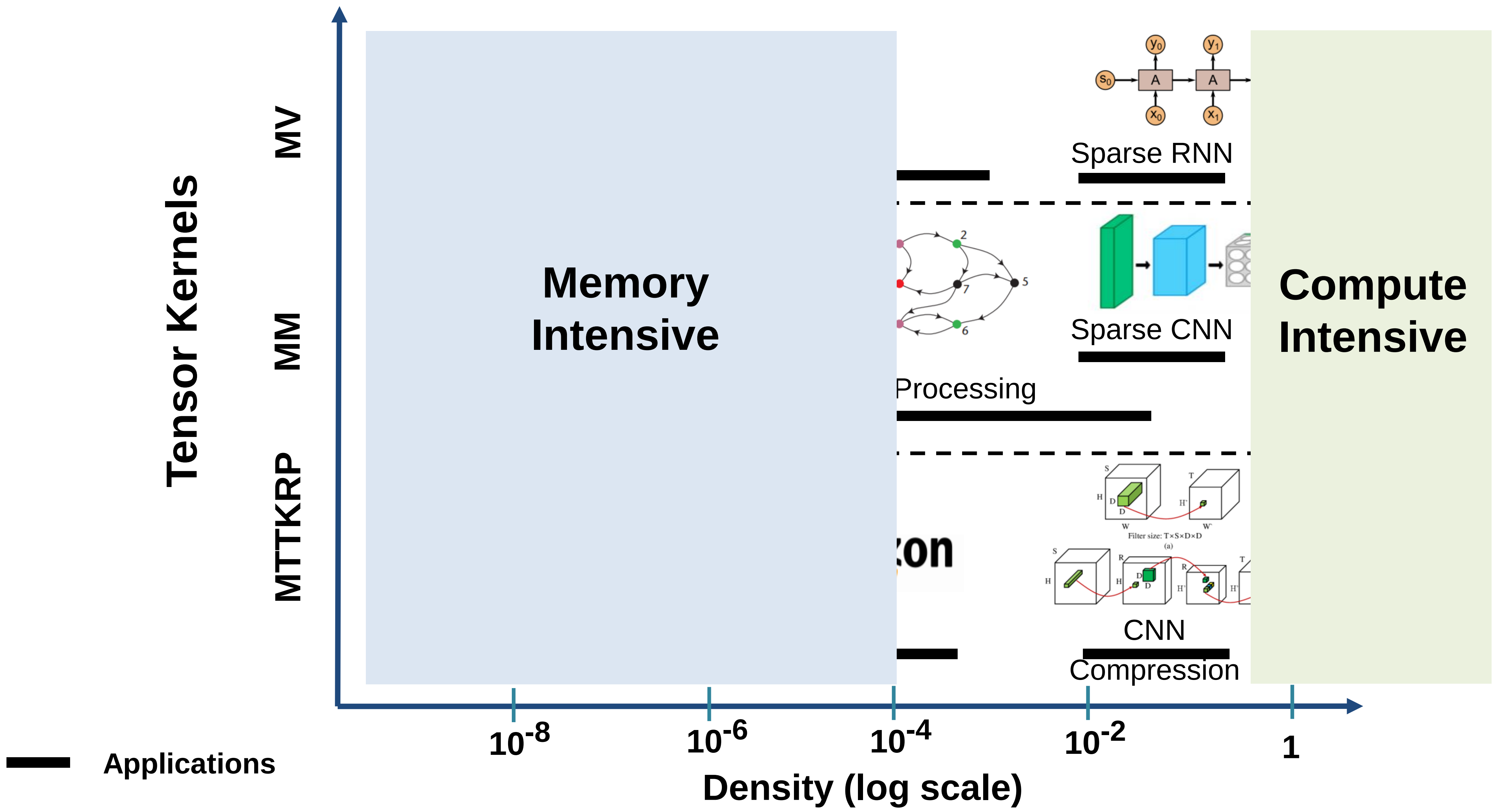
Kernel-Sparsity Spectrum of Tensor Applications



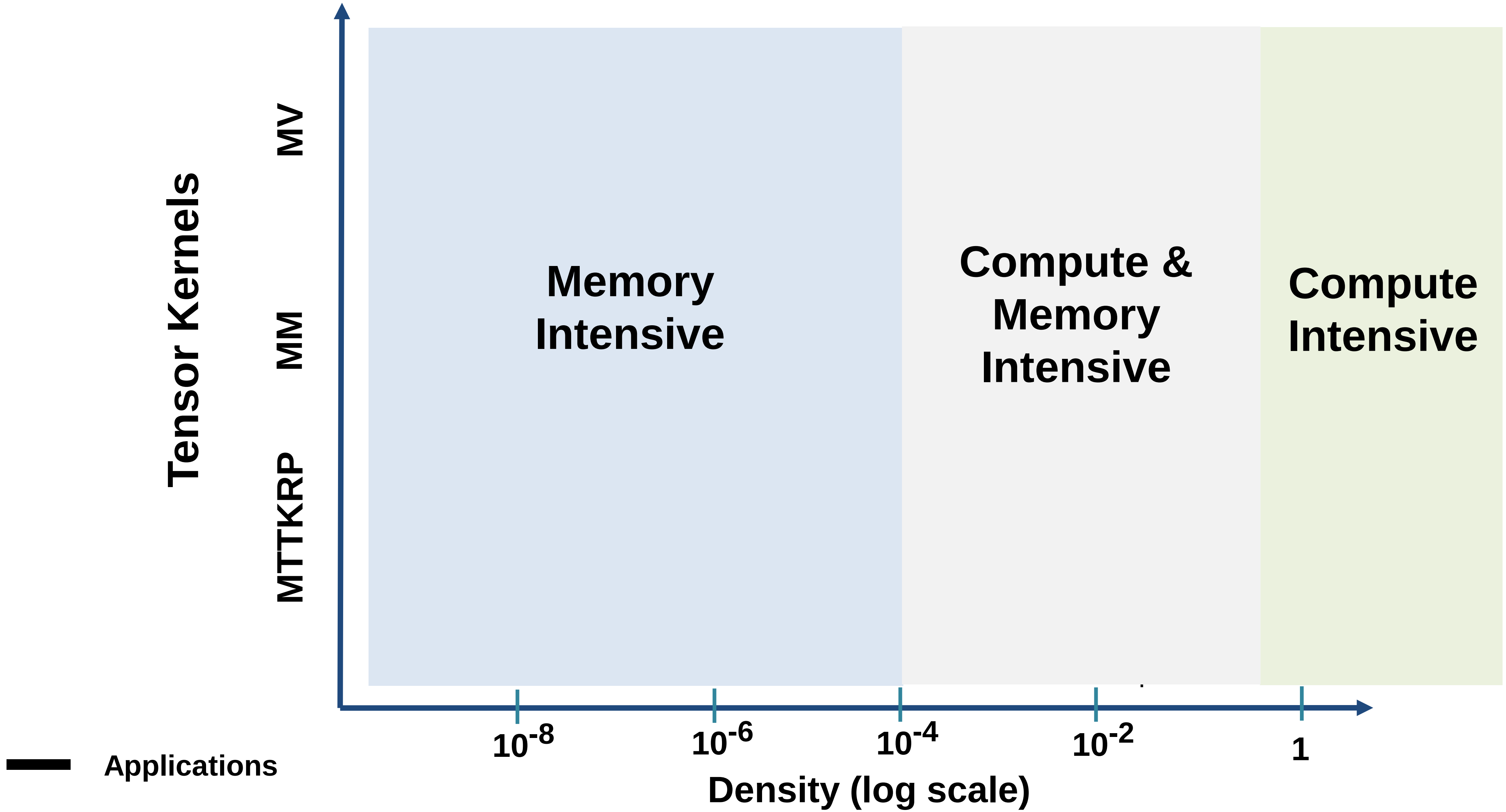
Kernel-Sparsity Spectrum of Tensor Applications



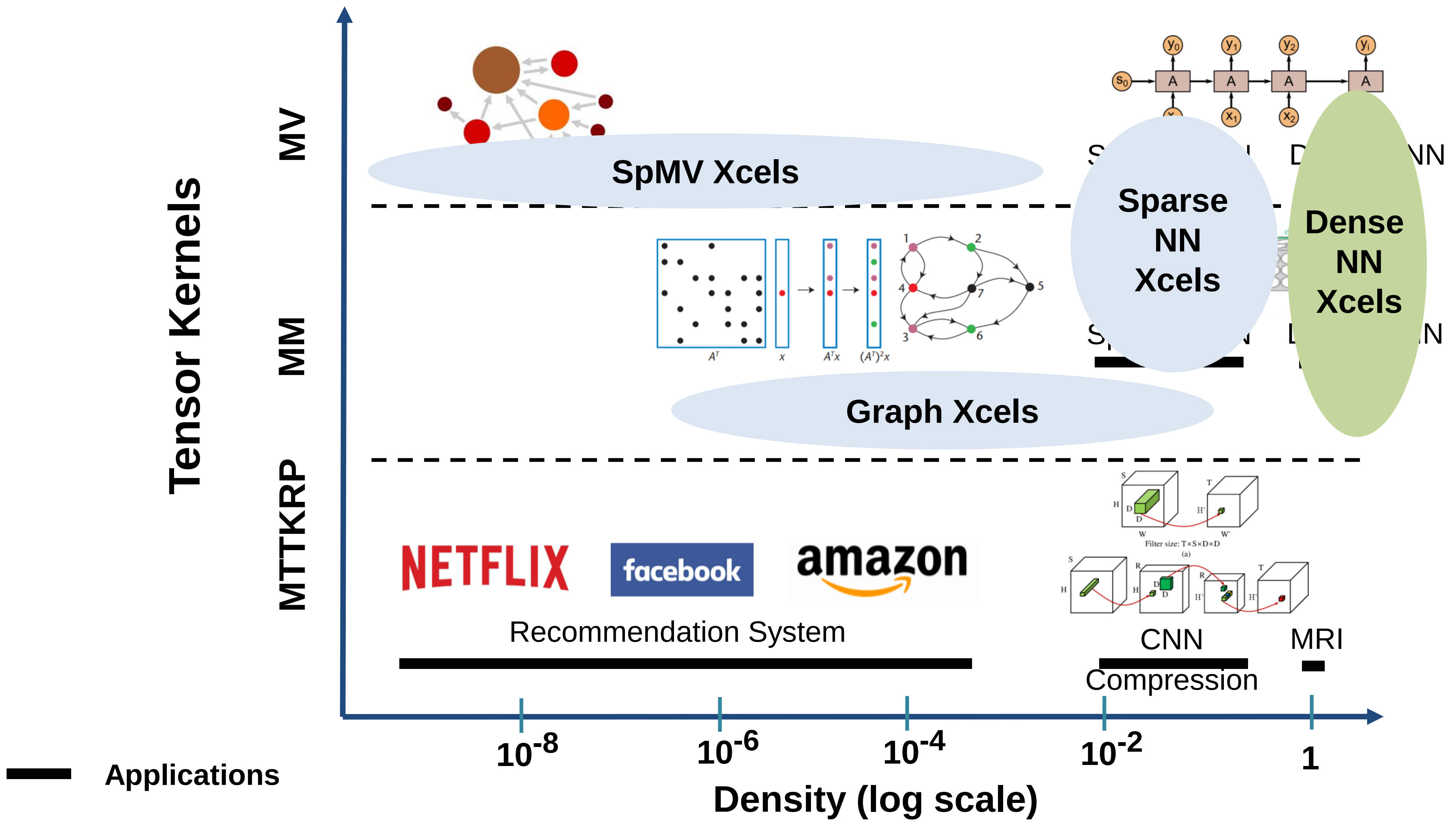
Kernel-Sparsity Spectrum of Tensor Applications



Kernel-Sparsity Spectrum of Tensor Applications



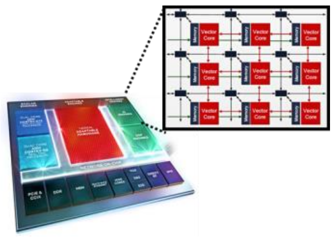
Kernel-Sparsity Spectrum of Tensor Applications



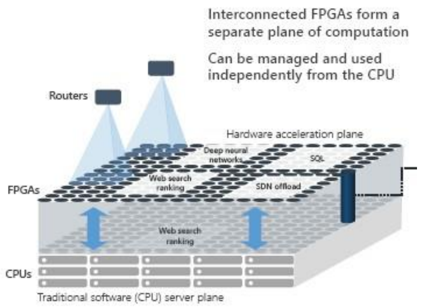
Examples of Tensor Accelerators

Examples of Tensor Accelerators

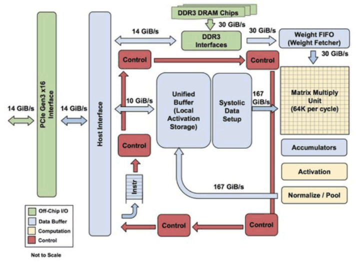
Dense



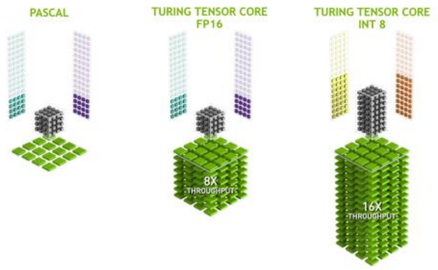
Xilinx Versal ACAP



Microsoft Brainwave



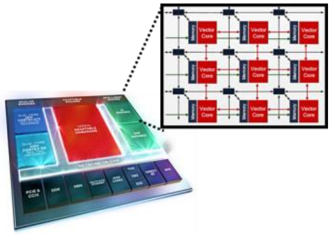
Google TPU



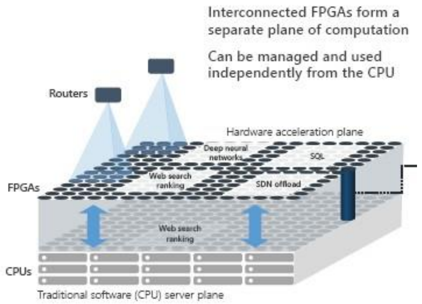
NVIDIA Tensor Core

Examples of Tensor Accelerators

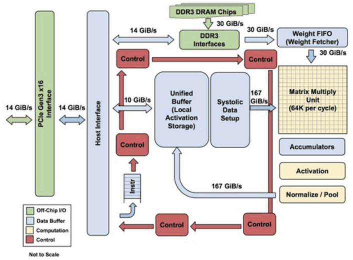
Dense



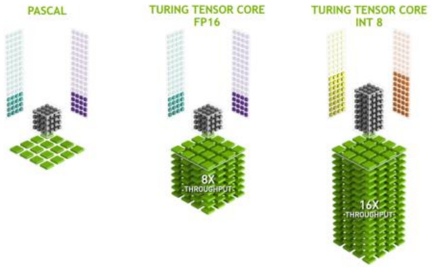
Xilinx Versal ACAP



Microsoft Brainwave

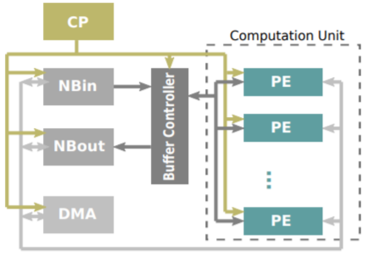


Google TPU

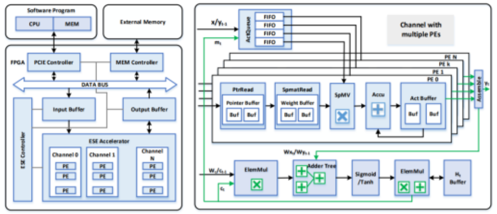


NVIDIA Tensor Core

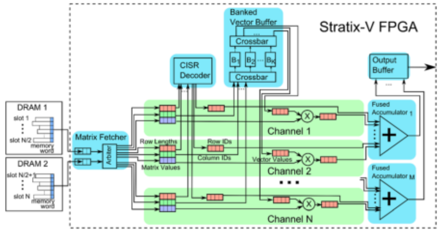
Sparse



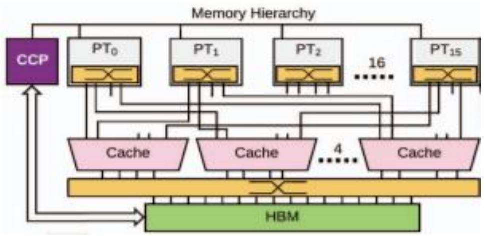
Cambricon-X[1]
(Sparse CNN)



ESE[2]
(Sparse RNN)



SpMV Xcel[3]

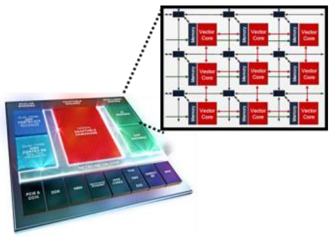


OuterSPACE[4]
(Graph Xcel)

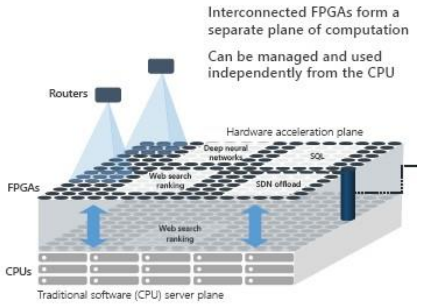
[1] Zhang, Shijin, et al. "Cambricon-x: An accelerator for sparse neural networks." , *Int'l Symp. on Microarchitecture,* 2016.
[2] Han, Song, et al. "Ese: Efficient speech recognition engine with sparse lstm on fpga." *Int'l Symp. on Field-Programmable Gate Arrays,* 2017.
[3] Fowers, Jeremy, et al. "A high memory bandwidth fpga accelerator for sparse matrix-vector multiplication." *Int'l Symp. on Field-Programmable Custom Computing Machines*, 2014.
[4] Pal, Subhankar, et al. "Outerspace: An outer product based sparse matrix multiplication accelerator." *Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2018.

Examples of Tensor Accelerators

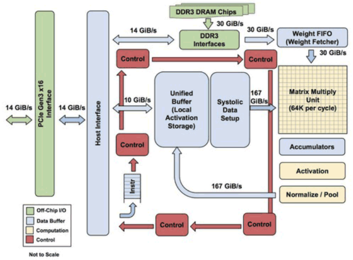
Dense



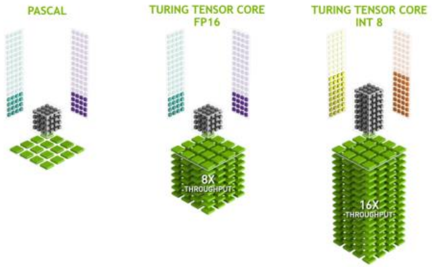
Xilinx Versal ACAP



Microsoft Brainwave

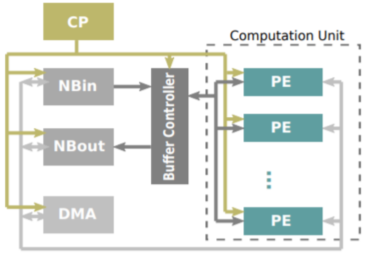


Google TPU

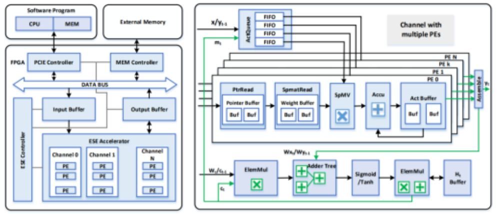


NVIDIA Tensor Core

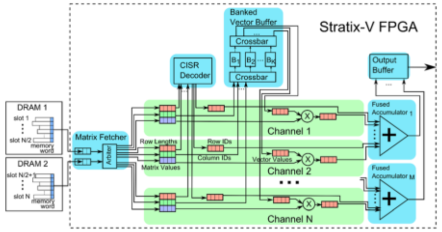
Sparse



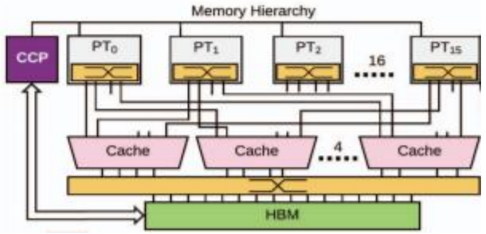
Cambricon-X[1]
(Sparse CNN)



ESE[2]
(Sparse RNN)



SpMV Xcel[3]



OuterSPACE[4]
(Graph Xcel)

All these accelerators either focus on a single tensor kernel or certain class of applications with some assumptions on sparsity

[1] Zhang, Shijin, et al. "Cambricon-x: An accelerator for sparse neural networks." , *Int'l Symp. on Microarchitecture,* 2016.
[2] Han, Song, et al. "Ese: Efficient speech recognition engine with sparse lstm on fpga." *Int'l Symp. on Field-Programmable Gate Arrays,* 2017.
[3] Fowers, Jeremy, et al. "A high memory bandwidth fpga accelerator for sparse matrix-vector multiplication." *Int'l Symp. on Field-Programmable Custom Computing Machines*, 2014.
[4] Pal, Subhankar, et al. "Outerspace: An outer product based sparse matrix multiplication accelerator." *Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2018.

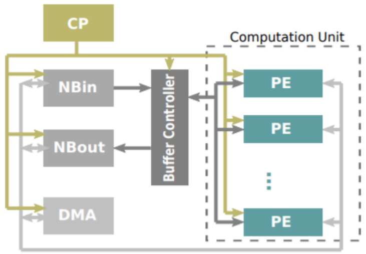
Challenges with Dense and Sparse Tensor Acceleration

Dense

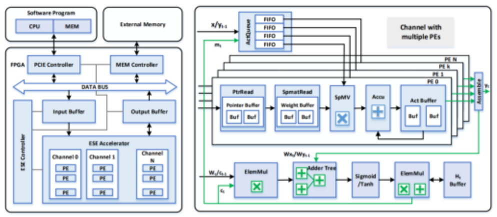
Productivity

Although dense accelerators are well-understood, designing a highly-efficient dense tensor accelerator in short amount of time still remains a challenge

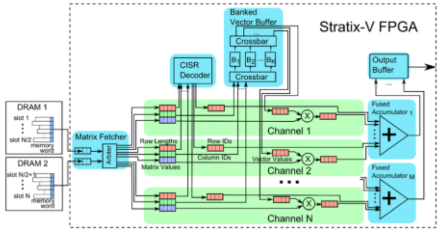
Sparse



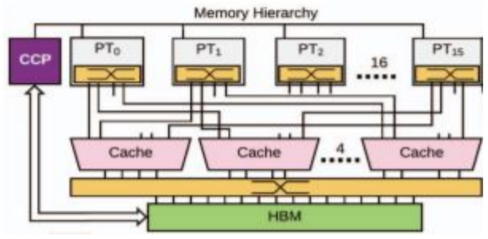
Cambricon-X[1]
(Sparse CNN)



ESE[2]
(Sparse RNN)



SpMV Xcel[3]



OuterSPACE[4]
(Graph Xcel)

Challenges with Dense and Sparse Tensor Acceleration

Dense

Productivity

Although dense accelerators are well-understood, designing a highly-efficient dense tensor accelerator in short amount of time still remains a challenge

Sparse

Flexibility

Existing sparse tensor accelerators focus on a single application while making assumptions on sparsity

Efficiency

Achieving high performance and energy efficiency while maintaining flexibility

Challenges with Dense and Sparse Tensor Acceleration

Dense

Productivity

Although dense accelerators are well-understood, designing a highly-efficient dense tensor accelerator in short amount of time still remains a challenge

Sparse

Flexibility

Existing sparse tensor accelerators focus on a single application while making assumptions on sparsity

Efficiency

Achieving high performance and energy efficiency while maintaining flexibility

In my dissertation, I would attempt to address the productivity challenge in dense tensor acceleration and flexibility and efficiency challenge in sparse tensor acceleration

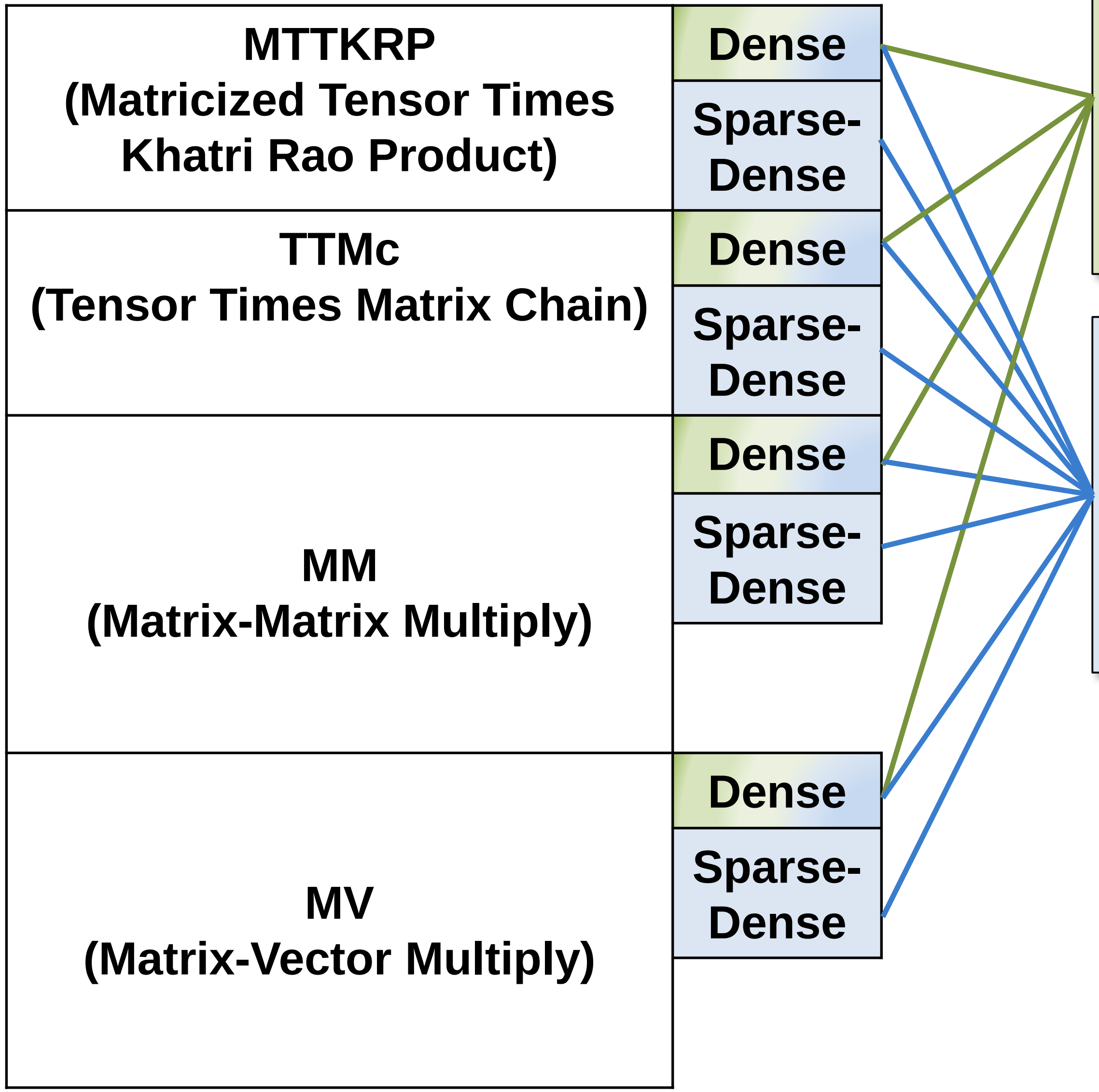
Dissertation Structure

MTTKRP (Matricized Tensor Times Khatri Rao Product)	**Dense**
TTMc (Tensor Times Matrix Chain)	**Dense**
MM (Matrix-Matrix Multiply)	**Dense**
MV (Matrix-Vector Multiply)	**Dense**

Dense

- **T2S-Tensor:** a language and a compilation framework to productively generate high-performance accelerators for dense tensor algebra (FCCM'19) – *Chapter 3*

Dissertation Structure



Dense

- **T2S-Tensor:** a language and a compilation framework to productively generate high-performance accelerators for dense tensor algebra (FCCM'19) – *Chapter 3*

Mixed Sparse-Dense

- **Tensaurus:** a hardware accelerator for mixed sparse-dense tensor computations such as tensor factorizations and matrix multiplications (HPCA'20) – *Chapter 4*

Dissertation Structure

MTTKRP (Matricized Tensor Times Khatri Rao Product)	**Dense**
	Sparse-Dense
TTMc (Tensor Times Matrix Chain)	**Dense**
	Sparse-Dense
MM (Matrix-Matrix Multiply)	**Dense**
	Sparse-Dense
	Sparse-Sparse
MV (Matrix-Vector Multiply)	**Dense**
	Sparse-Dense
	Sparse-Sparse

Dense

- **T2S-Tensor:** a language and a compilation framework to productively generate high-performance accelerators for dense tensor algebra (FCCM'19) – *Chapter 3*

Mixed Sparse-Dense

- **Tensaurus:** a hardware accelerator for mixed sparse-dense tensor computations such as tensor factorizations and matrix multiplications (HPCA'20) – *Chapter 4*

Sparse-Sparse

- **MatRaptor:** a hardware accelerator for the most popular sparse-sparse tensor kernel i.e. sparse-sparse matrix multiply (submitted to ISCA'20) – *Chapter 5*

Chapter 3

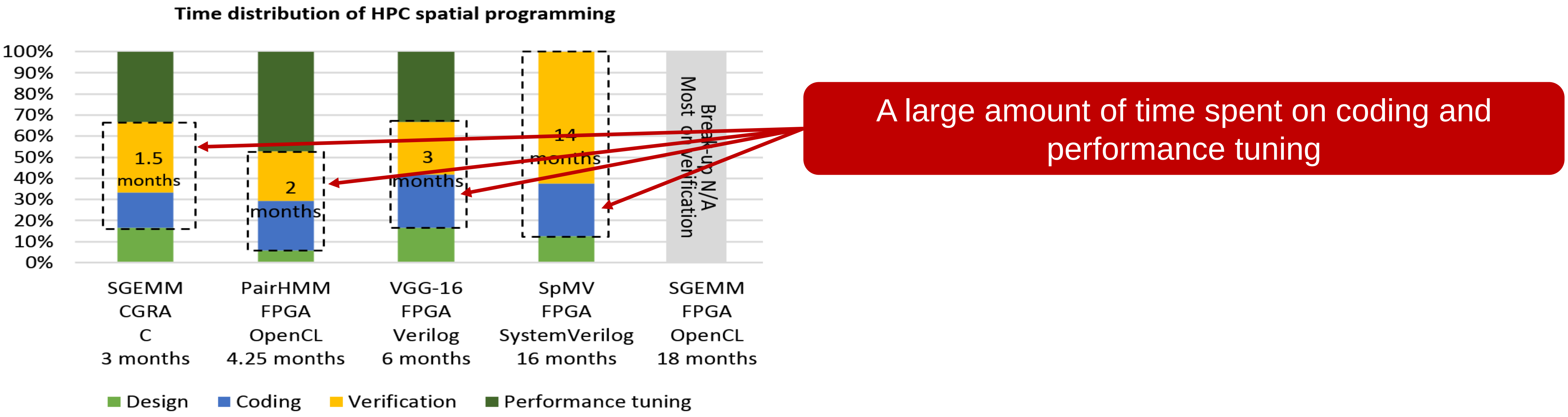
T2S-Tensor : Productively Generating High-Performance Spatial Hardware for Dense Tensor Computations

- Appears in International Symposium on Field-Programmable Custom Computing Machines *(FCCM'19)*

Productively Generating Dense Tensor Systolic Array is Hard

Productively Generating Dense Tensor Systolic Array is Hard

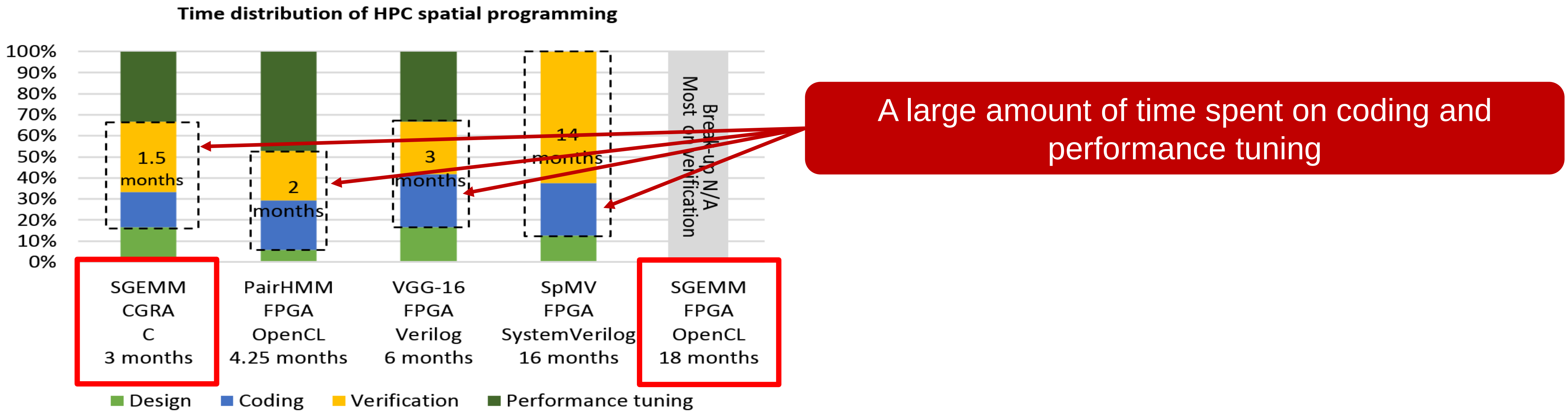
- Designing high-performance dense tensor accelerators is hard



Source of data: Daya Khudia (Intel, SSG), Gorge Powley (Intel, DCG), Yufei Ma (ASU), Jeremy Fowers (Microsoft), Davor Capalija and Tomasz Czajkowski (Intel, PSG)

Productively Generating Dense Tensor Systolic Array is Hard

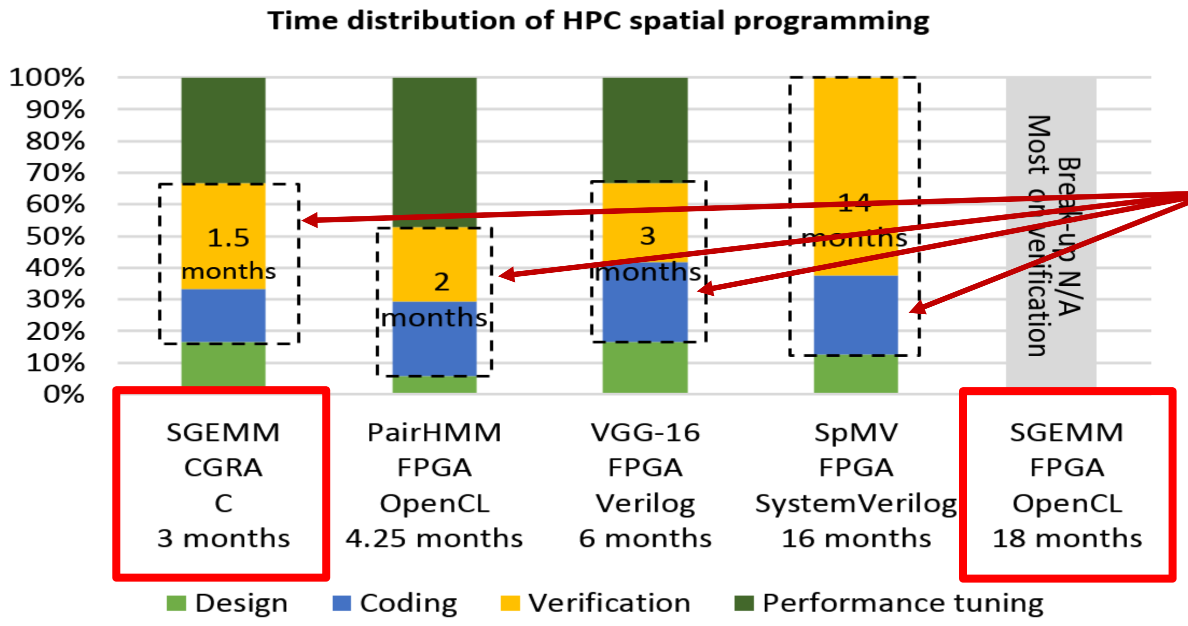
- Designing high-performance dense tensor accelerators is hard
- Even for well-known kernels like MM it takes around 18 months to come up with a high-performance design.



Source of data: Daya Khudia (Intel, SSG), Gorge Powley (Intel, DCG), Yufei Ma (ASU), Jeremy Fowers (Microsoft), Davor Capalija and Tomasz Czajkowski (Intel, PSG)

Productively Generating Dense Tensor Systolic Array is Hard

- Designing high-performance dense tensor accelerators is hard
- Even for well-known kernels like MM it takes around 18 months to come up with a high-performance design.



What to do?

- **Reduce efforts for coding and design space exploration**

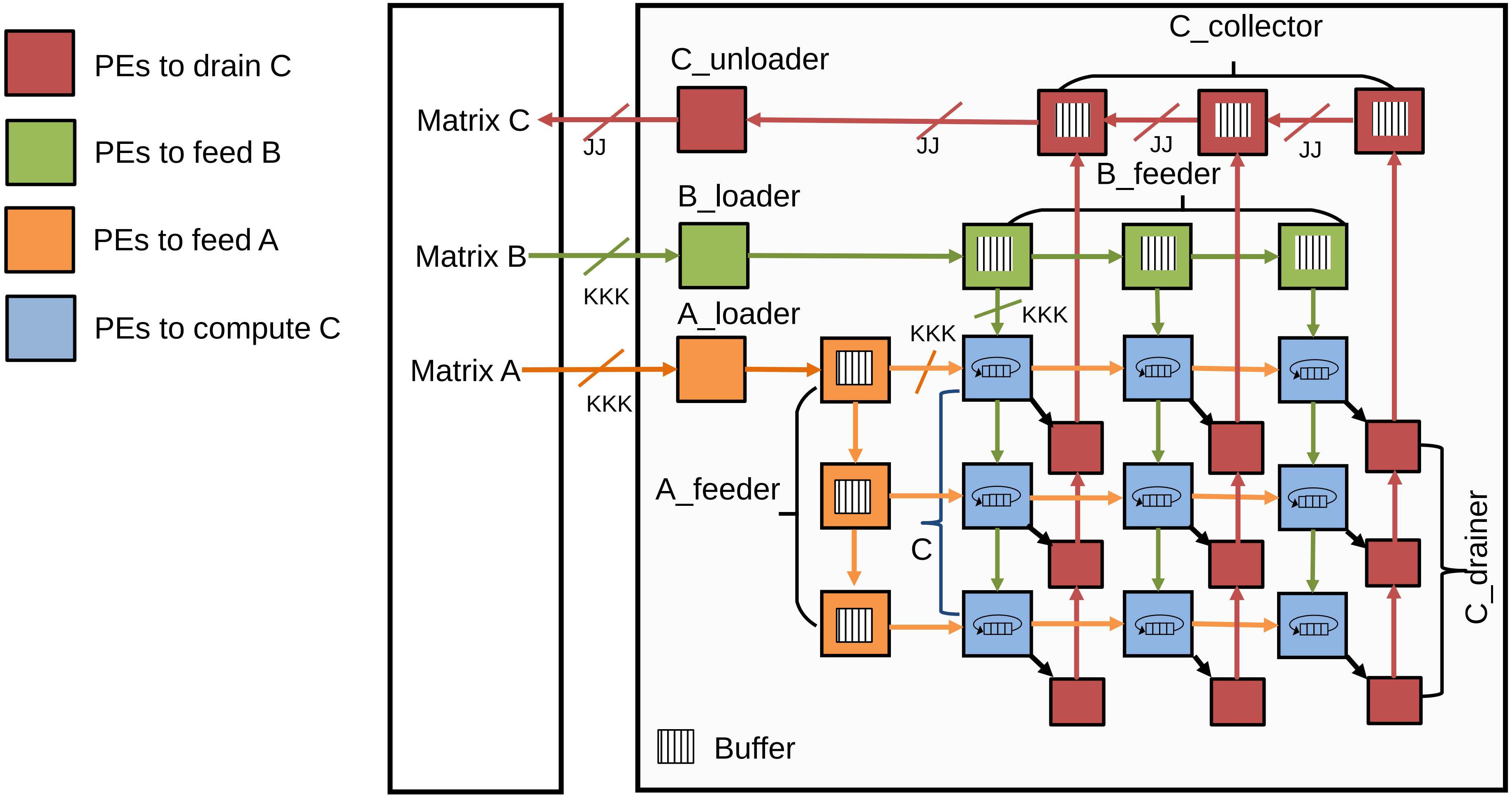
Source of data: Daya Khudia (Intel, SSG), Gorge Powley (Intel, DCG), Yufei Ma (ASU), Jeremy Fowers (Microsoft), Davor Capalija and Tomasz Czajkowski (Intel, PSG)

T2S-Tensor Contributions

- **T2S-Tensor:** A language and compilation framework to productively generate systolic arrays for dense tensor computations

- **Key Features:**
  - Provides a concise yet expressive programming abstraction that **decouples hardware optimizations from algorithm**
  - Provides a **set of key compiler optimizations** that are essential for creating high-performance systolic arrays for dense tensor computations

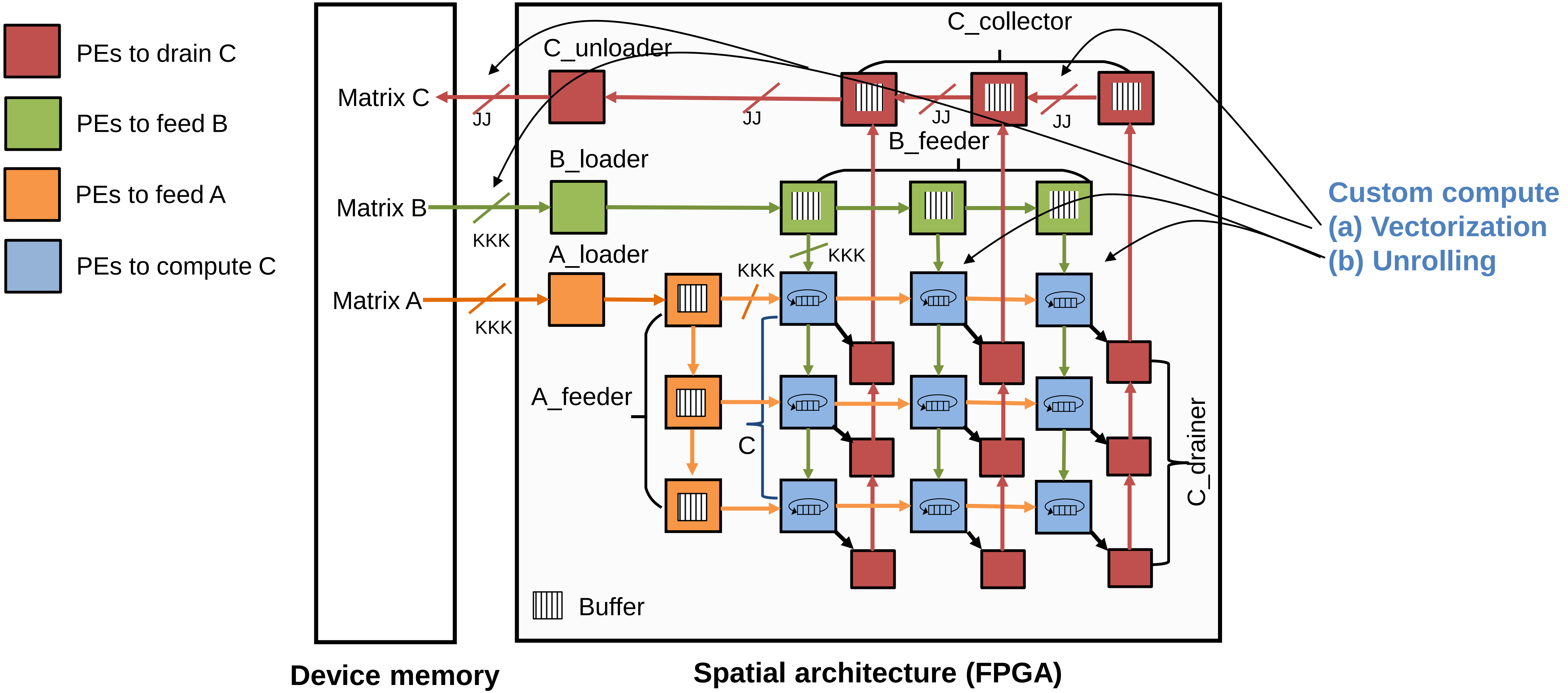
Driving Example – Dense Matrix Multiplication (Dense MM)



Device memory **Spatial architecture (FPGA)**

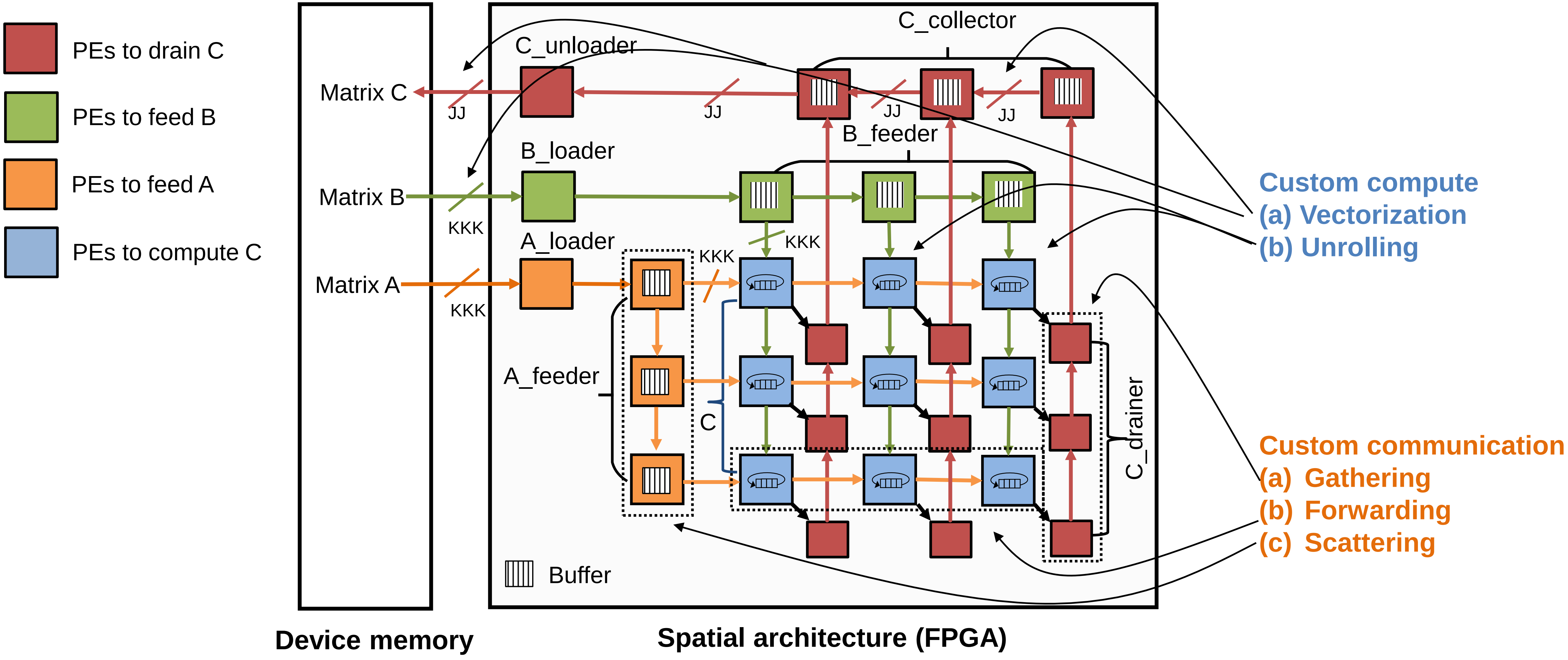
High-performance dense MM design on FPGA

Driving Example – Dense Matrix Multiplication (Dense MM)



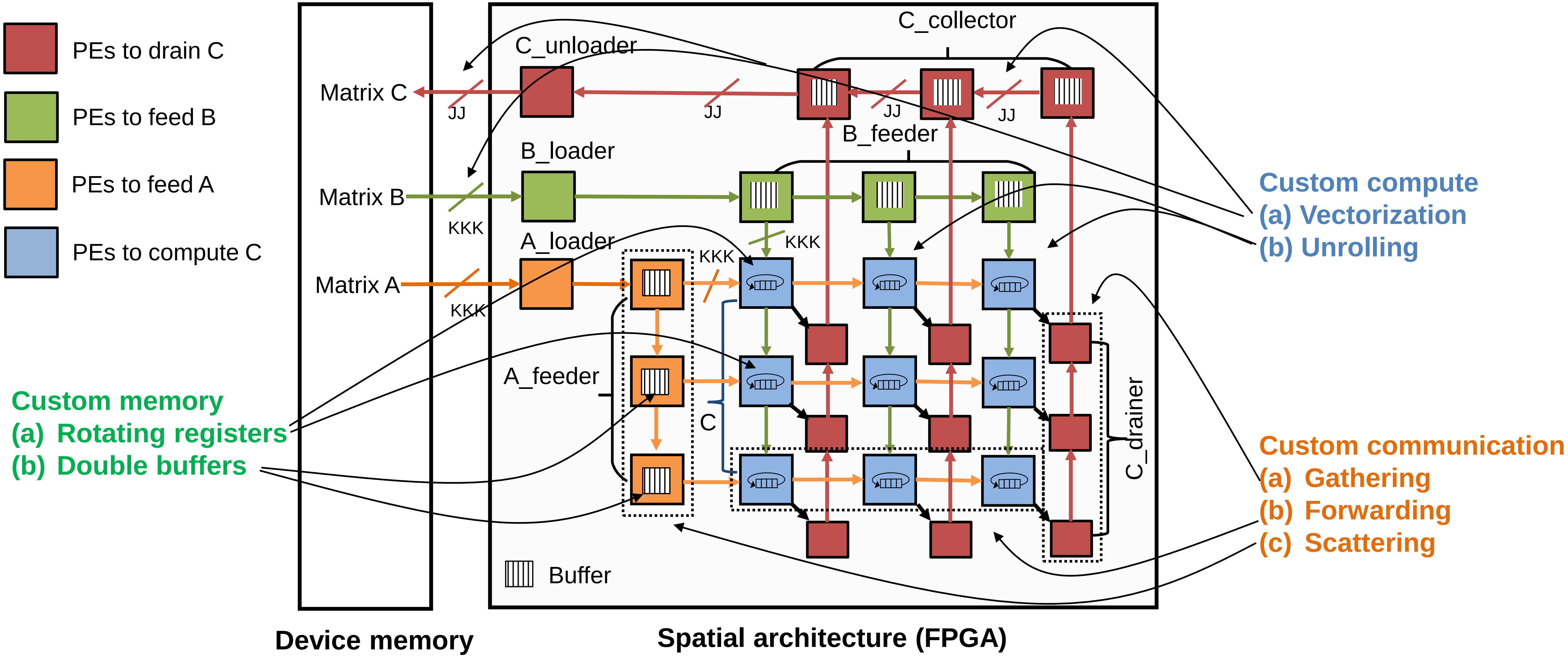
High-performance dense MM design on FPGA

Driving Example – Dense Matrix Multiplication (Dense MM)



High-performance dense MM design on FPGA

Driving Example – Dense Matrix Multiplication (Dense MM)



High-performance dense MM design on FPGA

High-Performance Dense MM Design in HLS

```
unrolled for(int ii = 0; ii < II; ii++) {
unrolled for(int jj = 0; jj < JJ; jj++) {
  for(int i = 0; i < I/II*III; i++) {
    for(int j = 0; j < J/JJ*JJJ; j++) {
      for(int k = 0; k < K/KK; k++) {
        float buffer[III][JJJ]
        for(int iii = 0; iii < III; iii++) {
          for(int jjj = 0; jjj < JJJ; jjj++) {
            8xfloat a = RCH (chA[ii][jj])
            WCH (chA[ii+1][jj], a)
            8xfloat b = RCH (chB[ii][jj])
            WCH (chB[ii][jj+1], b)
            if (drain)
               WCH (chC[ii][jj], buf[iii][jjj])
            #pragma unroll
            for(int kkk = 0; kkk < KKK; kkk++)
               sum += a[kkk]*b[kkk]
            buffer[iii][jjj] += sum;
}}}}}}}
```

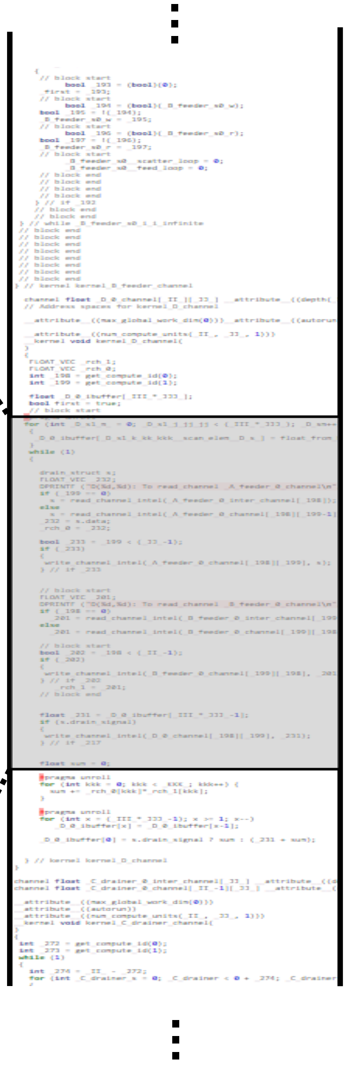
Custom compute (Loop unrolling)

Custom compute (Loop tiling)

Custom comm. (Forwarding)

Custom memory (Buffer)

Custom compute (Vectorization)



750 lines
of high-performance HLS code

High-Performance Dense MM Design in HLS

```
unrolled for(int ii = 0; ii < II; ii++) {
unrolled for(int jj = 0; jj < JJ; jj++) {
 for(int i = 0; i < I/II*III; i++) {
   for(int j = 0; j < J/JJ*JJJ; j++) {
     for(int k = 0; k < K/KK; k++) {
      float buffer[III][JJJ]
       for(int iii = 0; iii < III; iii++) {
          for(int jjj = 0; jjj < JJJ; jjj++) {
             8xfloat a = RCH (chA[ii][jj])
             WCH (chA[ii+1][jj], a)
             8xfloat b = RCH (chB[ii][jj])
             WCH (chB[ii][jj+1], b)
             if (drain)
                 WCH (chC[ii][jj], buf[iii][jjj])
              #pragma unroll
              for(int kkk = 0; kkk < KKK; kkk++)
                  sum += a[kkk]*b[kkk]
              buffer[iii][jjj] += sum;
}}}}}}}
```

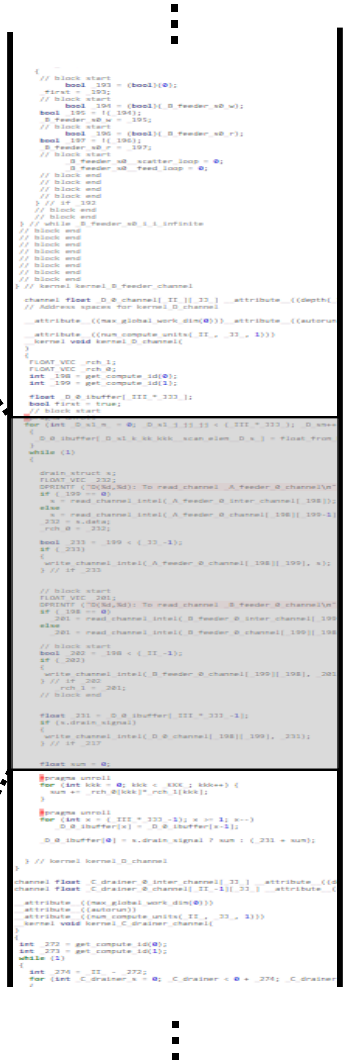
Custom compute (Loop unrolling)

Custom compute (Loop tiling)

Custom comm. (Forwarding)

Custom memory (Buffer)

Custom compute (Vectorization)



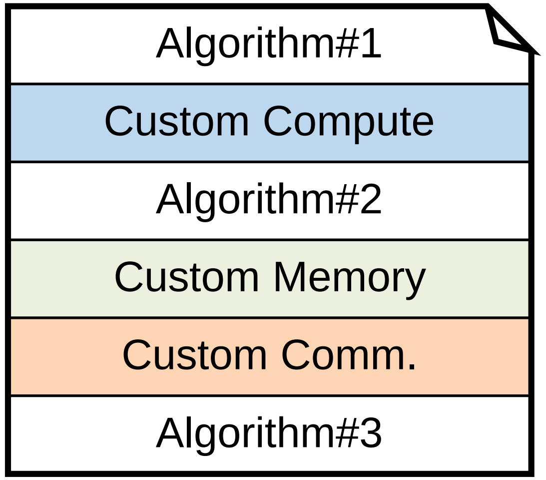
750 lines
of high-performance HLS code

Entangled Hardware Customization and Algorithm
Less Portable, Less Maintainable and Less Productive

Decoupling Hardware Customization and Algorithm

Decoupling Hardware Customization and Algorithm

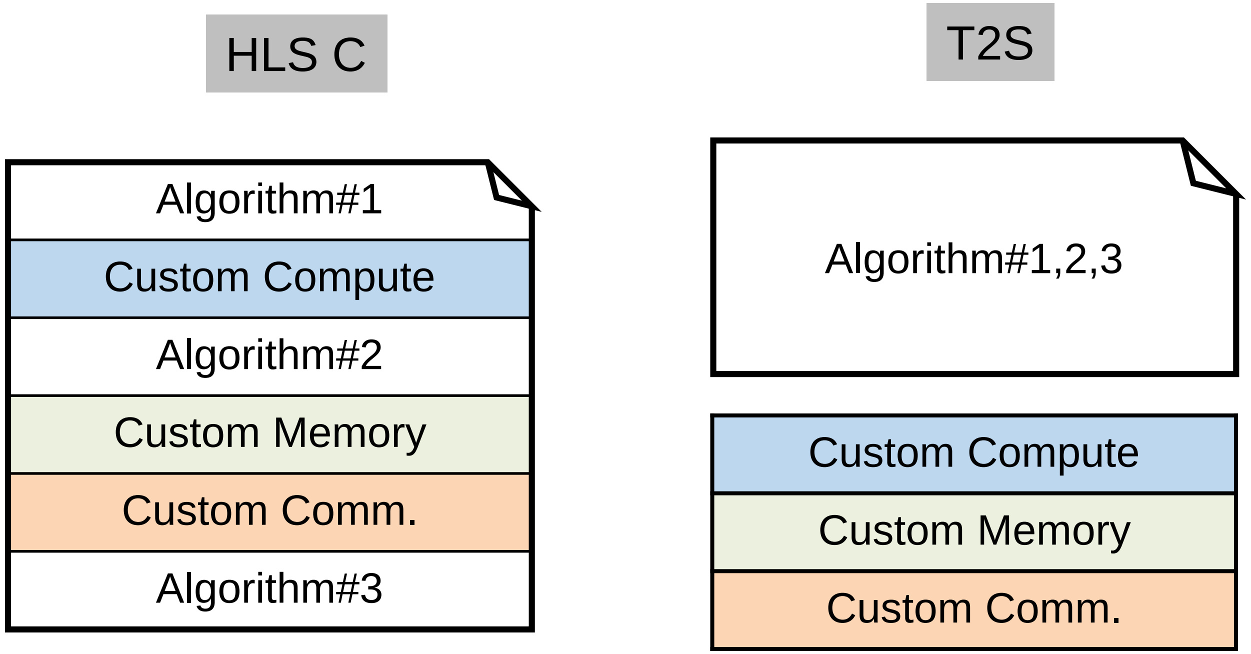
HLS C



Entangled algorithm specification & customization schemes [1,2,3]

[1] Intel HLS
[2] Xilinx Vivado HLS
[3] Canis, et al. FPGA'11

Decoupling Hardware Customization and Algorithm



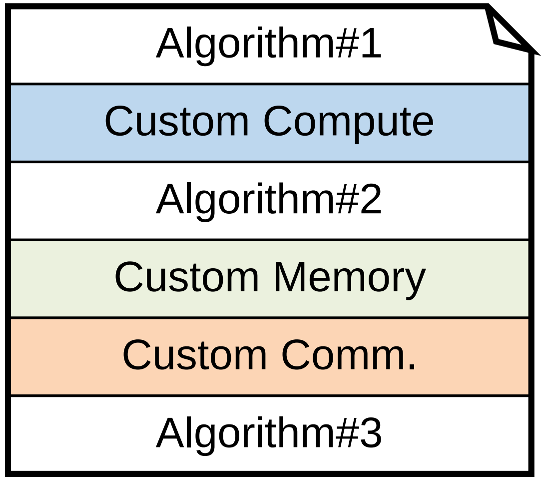
Entangled algorithm specification & customization schemes [1,2,3]

Decoupled customization & clean abstraction

[1] Intel HLS
[2] Xilinx Vivado HLS
[3] Canis, et al. FPGA'11

Decoupling Hardware Customization and Algorithm

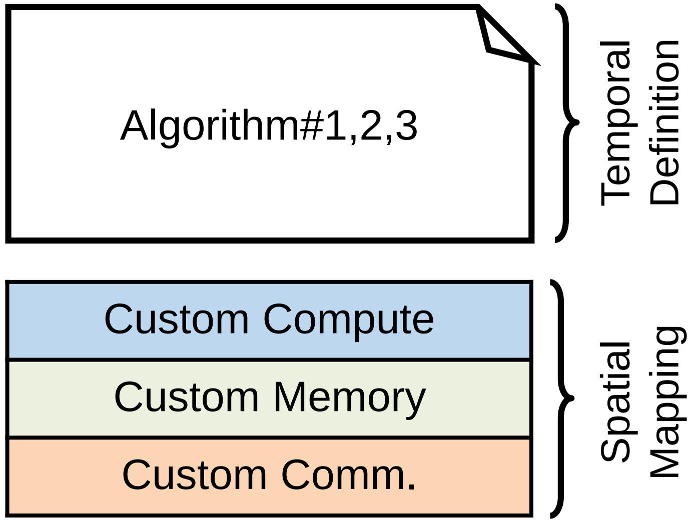
HLS C



Entangled algorithm specification & customization schemes [1,2,3]

[1] Intel HLS
[2] Xilinx Vivado HLS
[3] Canis, et al. FPGA'11

T2S



Decoupled customization & clean abstraction

```
C(i, j) = 0;
C(i, j) += A(i, k) * B(k, j);
C.update().tile(k, j, i, kk, jj, ii, KK, JJ, II);
                .isolate_producer_chain(A, A_loader, A_feeder)
                .isolate_producer_chain(B, B_loader, B_feeder)
                .isolate_consumer_chain(C, C_drainer, C_unloader);
A_loader.unroll(ii).remove(jj).vload(kk);
A_feeder.buffer(ii, Buffer::Double).unroll(ii);
B_loader.unroll(jj).remove(ii).vload(kk);
B_feeder.buffer(k, Buffer::Double).unroll(jj);
C.update().unroll(jj, ii)
              .forward(A_feeder, {1, 0}) .forward(B_feeder, {0, 1});
C_drainer.unroll(jj, ii).gather(C, {1, 0})
C_unloader.buffer(ii).unroll(ii).vstore(jj);
```

Temporal to Spatial ⟶ T2S

T2S is an extension over Halide for spatial architectures

Temporal Definition in T2S

```
Func C
C(i, j) = 0
C(i, j) += A(i, k) * B(k, j)
C.tile(i,j,k,ii,jj,kk,II,JJ,KK)
```

Algorithm

Temporal Definition in T2S

```
Func C
C(i, j) = 0
C(i, j) += A(i, k) * B(k, j)
C.tile(i,j,k,ii,jj,kk,II,JJ,KK)
```

Algorithm

C

```
for i, j, k
   for ii, jj, kk
      i’ = i*II + ii
      j’ = j*JJ + jj
      k’ = k*KK + kk
      C[i’, j’] += A[i’,k’] * B[k’,j’]
```

C

Spatial Mapping in T2S

```
Func C
C(i, j) = 0
C(i, j) += A(i, k) * B(k, j)
C.tile(i,j,k,ii,jj,kk,II,JJ,KK)
C.isolate_producer(A, A_feeder)
```

Algorithm

Spatial Mapping

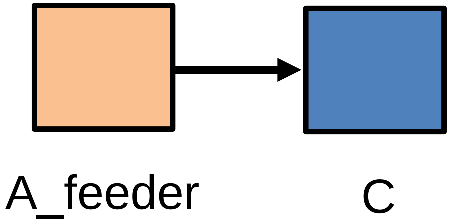
A_feeder

```
for i, j, k
  for ii, jj, kk
    i',j',k' = …
    WCH (ch1, A[i',k'])
```

C

```
for i, j, k
  for ii, jj, kk
    i',j',k' = …
    C[i', j'] +=          * B[k',j']
```

channel1 (ch1)



Spatial Mapping in T2S

```
Func C
C(i, j) = 0
C(i, j) += A(i, k) * B(k, j)
C.tile(i,j,k,ii,jj,kk,II,JJ,KK)
C.isolate_producer(A, A_feeder)
C.unroll(ii, jj)
```

Algorithm

Spatial Mapping

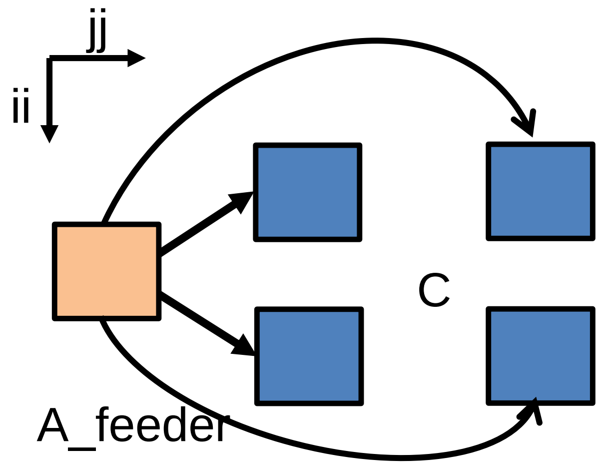
A_feeder

```
for i, j, k
  for ii, jj, kk
    i',j',k' = …
    WCH (ch1[ii][jj], A[i',k'])
```

C

```
unrolled for ii, jj
for i, j, k
  for kk
    i',j',k' = …
    C[i', j'] =            * B[k',j']
```

ch1[ii][jj]



Spatial Mapping in T2S

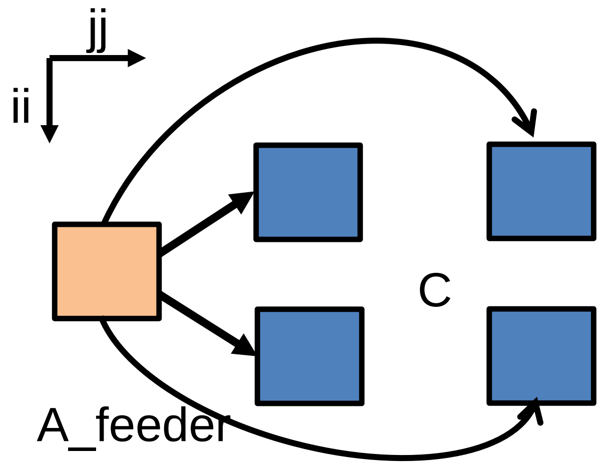
```
Func C
C(i, j) = 0
C(i, j) += A(i, k) * B(k, j)
C.tile(i,j,k,ii,jj,kk,II,JJ,KK)
C.isolate_producer(A, A_feeder)
C.unroll(ii, jj)
```

Algorithm

Spatial Mapping

A_feeder

```
for i, j, k
   for ii, jj, kk
     i',j',k' = …
     WCH (ch1[ii][jj], A[i',k'])
```



Spatial Mapping in T2S

Algorithm

```
Func C
C(i, j) = 0
C(i, j) += A(i, k) * B(k, j)
C.tile(i,j,k,ii,jj,kk,II,JJ,KK)
```

Spatial Mapping

```
C.isolate_producer(A, A_feeder)
C.unroll(ii, jj)
A_feeder.isolate_producer(A, A_loader)
A_loader.remove(jj)
A_feeder.buffer(ii, DOUBLE)
```

A_feeder

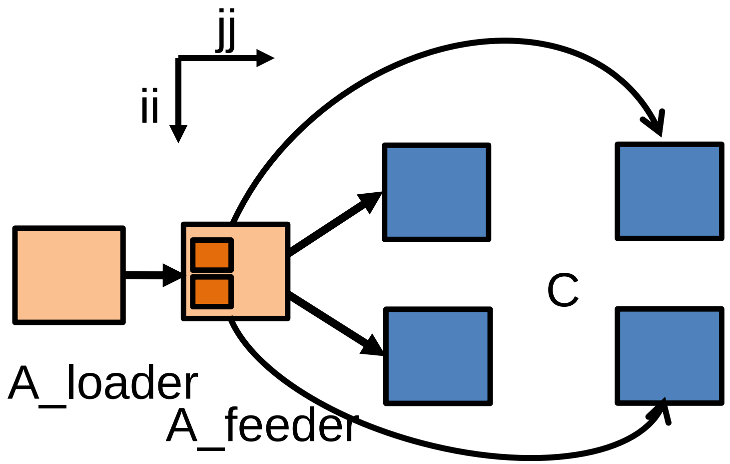
```
for i, j, k, ii
  float buf [KK]
  for kk = 0 .. KK
    buf [kk] = RCH(...)
```

A_loader

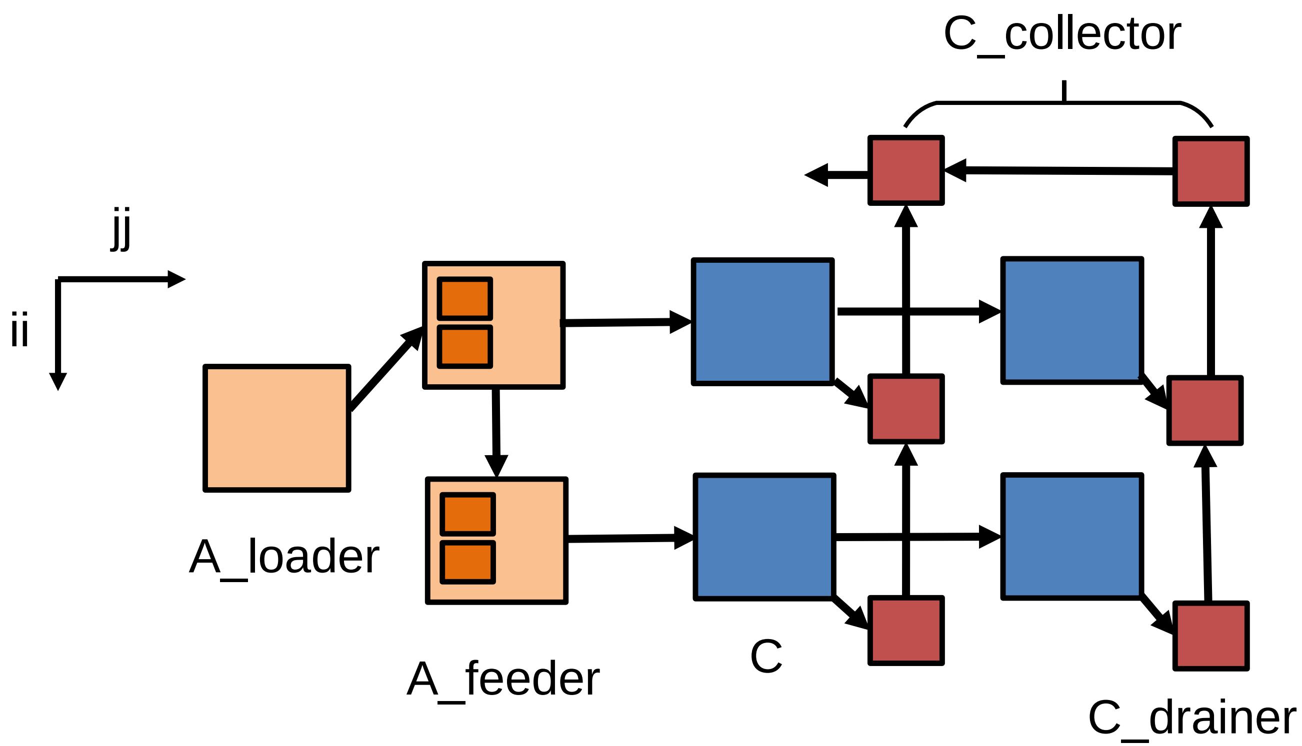
for i, j, k
  for ii, ~~jj,~~ kk
    i',j',k' = …
    WCH (ch2, A[i',k'])

ch2

```
for jj, kk
  i',j',k' = …
  WCH (ch1[ii][jj], buf[kk])
```



Spatial Mapping in T2S




```
Func C
C(i, j) = 0
C(i, j) += A(i, k) * B(k, j)
C.tile(i,j,k,ii,jj,kk,II,JJ,KK)
```

Algorithm

```
C.isolate_producer(A, A_feeder)
C.unroll(ii, jj)
A_feeder.isolate_producer(A, A_loader)
A_loader.remove(jj)
A_feeder.buffer(ii, DOUBLE)
C.forward(A_feeder, +jj)
A_feeder.unroll(ii).scatter(A, +ii)
C.isolate_consumer(C, C_drainer)
C_drainer.isolate_consumer_chain(C, C_collector, C_unloader)
C_drainer.unroll(ii).unroll(jj).gather(C, -ii)
C_collector.unroll(jj).gather(C, -jj)
```

Spatial Mapping

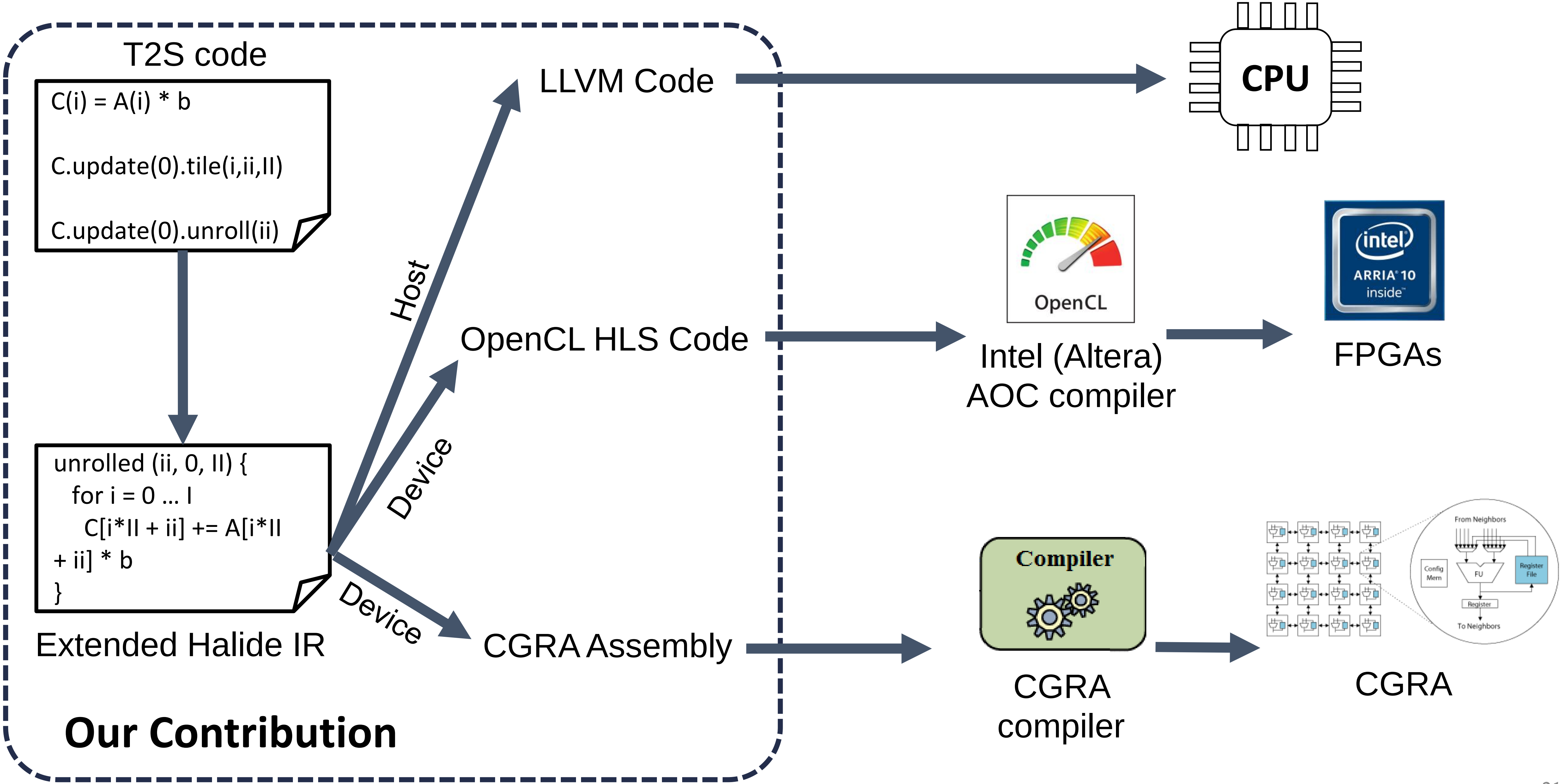
Complete T2S Code for Dense MM

~20 LOC vs 750 lines of HLS code

```
C(j, i) = 0.0f;
C(j, i) += A(k, i) * B(j, k);
C.tile(j, i, jj, ii, JJ, II).tile(jj, ii, jjj, iii, JJJ, III);
C.update(0).tile(k, j, i, kk, jj, ii, KK, JJ, II).tile(kk, jj, ii, kkk, jjj, iii, KKK, JJJ, III);
C.update(0).isolate_producer_chain(A, A_serializer, A_loader, A_feeder)
           .isolate_producer_chain(B, B_serializer, B_loader, B_feeder)
           .isolate_consumer_chain(C, C_drainer, C_collector, C_unloader, C_deserializer);
A_serializer.sread().swrite();
B_serializer.sread().swrite();
C.update(0).vread({A,B});
C.update(0).unroll(ii)
           .unroll(jj)
C.update(0).forward(A_feeder, { 0, 1 })
           .forward(B_feeder, { 1, 0 });
A_serializer.remove(jjj, jj, j);
A_loader.remove(jjj, jj);
A_feeder.buffer(ii, true).unroll(ii);
B_serializer.remove(iii, ii, i);
B_loader.remove(iii, ii);
B_feeder.buffer(k, true).unroll(jj);
A_feeder.scatter(A, {1});
B_feeder.scatter(B, {1});
C_drainer.unroll(ii).unroll(jj).gather(C, jjj, { 1, 0 });
C_collector.unroll(jj).gather(C_drainer, jjj, { 1 });
```

C.tile … / C.update(0).tile …	**Custom compute (Loop Tiling)**
isolate_producer_chain / isolate_consumer_chain	**Custom compute (Compute Partitioning)**
sread().swrite() / vread({A,B})	**Custom compute (Data Vectorization)**
unroll(ii).unroll(jj)	**Custom compute (Loop Unrolling)**
forward(A_feeder…) / forward(B_feeder…)	**Custom comm. (Data Forwarding)**
remove / buffer	**Custom memory (Buffer Insertion)**
scatter	**Custom comm. (Data Scattering)**
gather	**Custom comm. (Data Gathering)**

T2S Compilation Flow



Dense MM on Arria 10

Dense MM on Arria 10

- Baseline
  - Open-source NDRange-style OpenCL code, tuned on the specific FPGA
- Ninja
  - Hand-written and manually optimized design from industry

Dense MM on Arria 10

- Baseline
  - Open-source NDRange-style OpenCL code, tuned on the specific FPGA
- Ninja
  - Hand-written and manually optimized design from industry

	Baseline	T2S	Ninja
LOC	70	**20**	750
Systolic array size	--	10 x 8	10 x 8
Vector length	16 x float	16 x float	16 x float
# Logic elements	131 K (31%)	214 K (50%)	230 K (54%)
# DSPs	1,032 (68%)	**1,282 (84%)**	1,280 (84%)
# RAMs	1,534 (57%)	1,384 (51%)	1,069 (39%)
Frequency (MHz)	189	215	245
Throughput (GFLOPS)	311	**549**	626

1.8x speedup over the baseline with 3.5x less code
82% performance of ninja implementation with 3% code

Tensor Kernels on FPGA & CGRA

- Tensor decomposition kernels
  - **MTTKRP**: $Y(i,f) \quad += A(i,j,k) * B(j,f) * C(k,f)$
  - **TTM**: $\quad Y(i,j,f) += A(i,j,k) * B(k,f)$
  - **TTMc**: $\quad Y(i,f_1,f_2) += A(i,j,k) * B(j,f_1) * C(k,f_2)$

Evaluation on CGRA

	LOC	Throughput wrt. Ninja GEMM	FMA Usage
MM	40	92 %	**100 %**
MTTKRP	32	99 %	**100 %**
TTM	47	104 %	**100 %**
TTMc	38	103 %	**95 %**

Evaluation on Arria-10 FPGA

Benchmark	LOC	Systolic Array Size	Logic Usage	DSP Usage	RAM Usage	Frequency (MHz)	Throughput (GFLOPS)
MTTKRP	28	8 x 9	53 %	81 %	56 %	204	**700**
TTM	30	8 x 11	64 %	93 %	88 %	201	**562**
TTMc	37	8 x 10	54 %	90 %	62 %	205	**738**

~100 % FMA usage for CGRA
~80-90 % DSP utilization and 560-740 GFLOPS for FPGA

Chapter 4

Tensaurus : A Versatile Accelerator for Mixed Sparse-Dense Tensor Computations

- Appears in International Symposium on High-Performance Computer Architecture (*HPCA'20*)

Tensaurus Contributions

Tensaurus Contributions

- **Tensaurus:** First accelerator for sparse tensor factorizations

Tensaurus Contributions

- **Tensaurus:** First accelerator for sparse tensor factorizations

- **Key Idea:** Co-design **sparse format** and **computation pattern**

Tensaurus Contributions

- **Tensaurus:** First accelerator for sparse tensor factorizations

- **Key Idea:** Co-design **sparse format** and **computation pattern**

- **Key Features:**
  - **Versatile:**
    - NOT limited to tensor factorizations. Also supports **common matrix operations**
  - **Adaptable:**
    - Also **accelerates dense kernels** (tensor factorizations and matrix ops)
    - Easily **adapts to different levels of sparsity** found in various domains

Importance of Accelerator Friendly Formats

$$
\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & a_{00} & & a_{02} & \\
1 & & a_{11} & & \\
2 & & a_{21} & & a_{23} \\
3 & & & & a_{33} \\
\hline
\end{array}
\bullet
\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}
$$

Importance of Accelerator Friendly Formats

$$\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & a_{00} & & a_{02} & \\
1 & & a_{11} & & \\
2 & & a_{21} & & a_{23} \\
3 & & & & a_{33} \\
\hline
\end{array} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Compressed Sparse Row Format (CSR)

Importance of Accelerator Friendly Formats

$$\begin{array}{c|cccc|} & 0 & 1 & 2 & 3 \\ \hline 0 & a_{00} & & a_{02} & \\ 1 & & a_{11} & & \\ 2 & & a_{21} & & a_{23} \\ 3 & & & & a_{33} \\ \hline \end{array} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Compressed Sparse Row Format (CSR)

Values	a_{00}	a_{02}	a_{11}	a_{21}	a_{23}	a_{33}

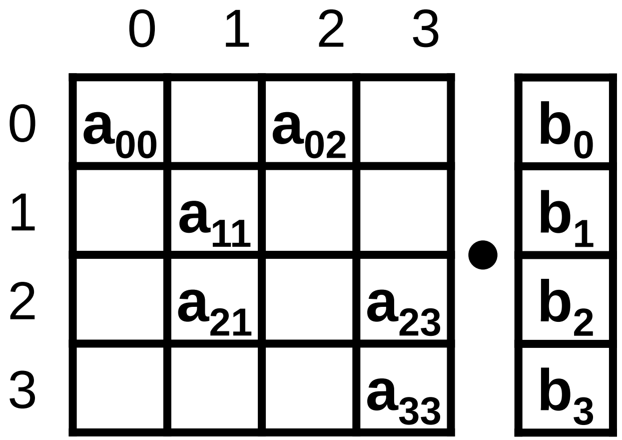
Importance of Accelerator Friendly Formats

$$\begin{array}{c c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} a_{00} & & a_{02} & \\ & a_{11} & & \\ & a_{21} & & a_{23} \\ & & & a_{33} \end{bmatrix} \end{array} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

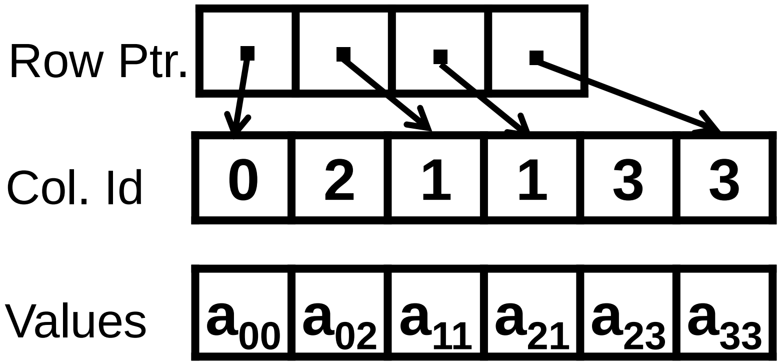
Compressed Sparse Row Format (CSR)

Col. Id	0	2	1	1	3	3
Values	a_{00}	a_{02}	a_{11}	a_{21}	a_{23}	a_{33}

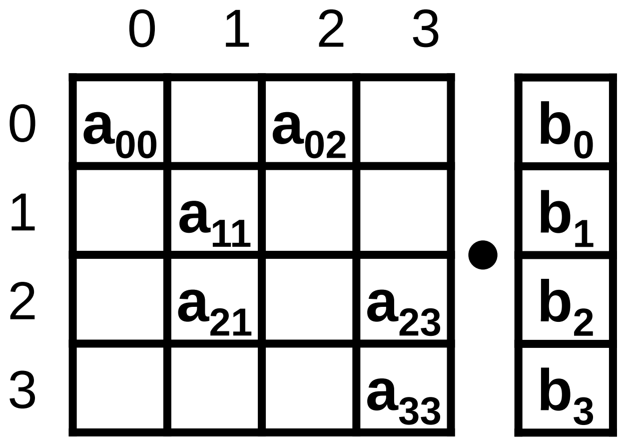
Importance of Accelerator Friendly Formats

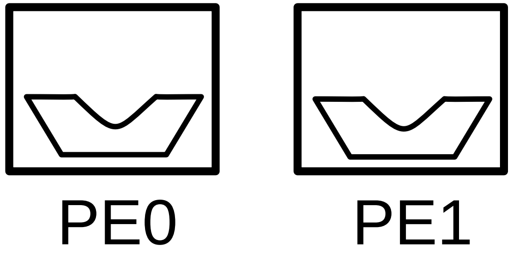


Compressed Sparse Row Format (CSR)



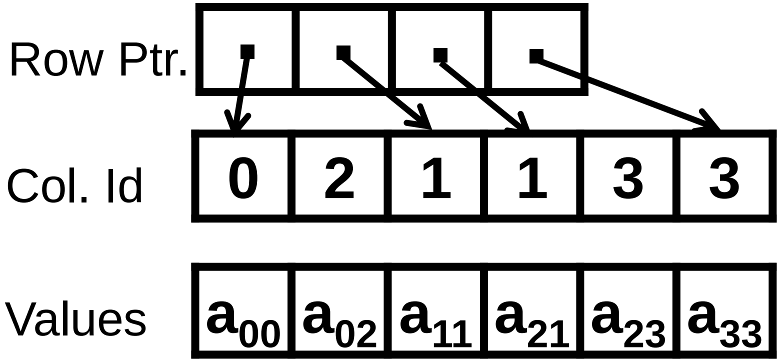
Importance of Accelerator Friendly Formats





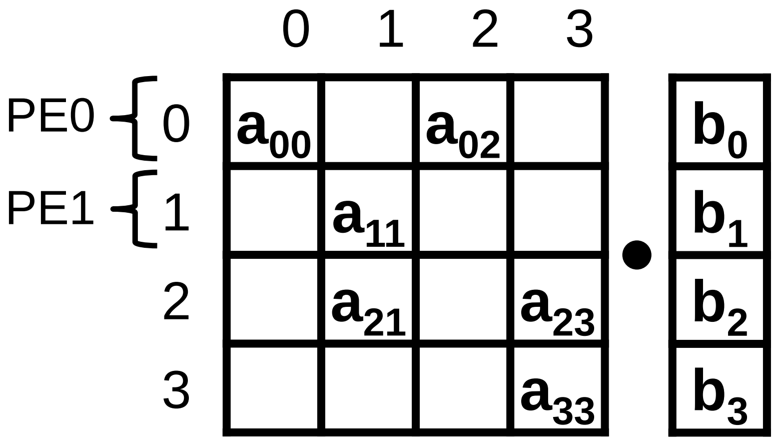
Compressed Sparse Row Format (CSR)

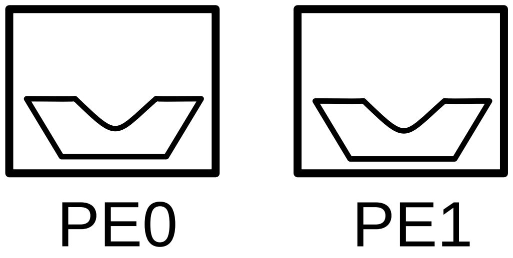
Compressed Interleaved Sparse Row (CISR) [1]



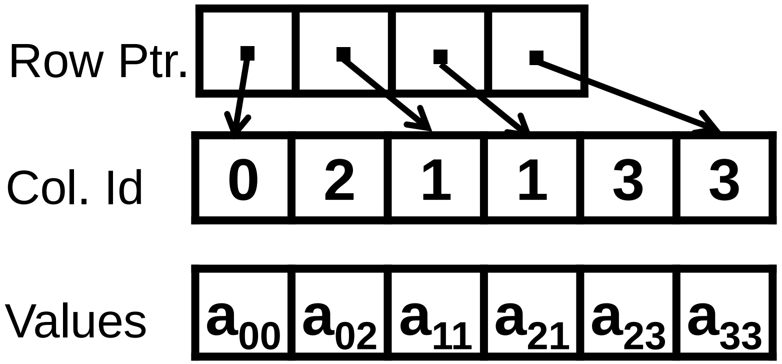
[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Importance of Accelerator Friendly Formats

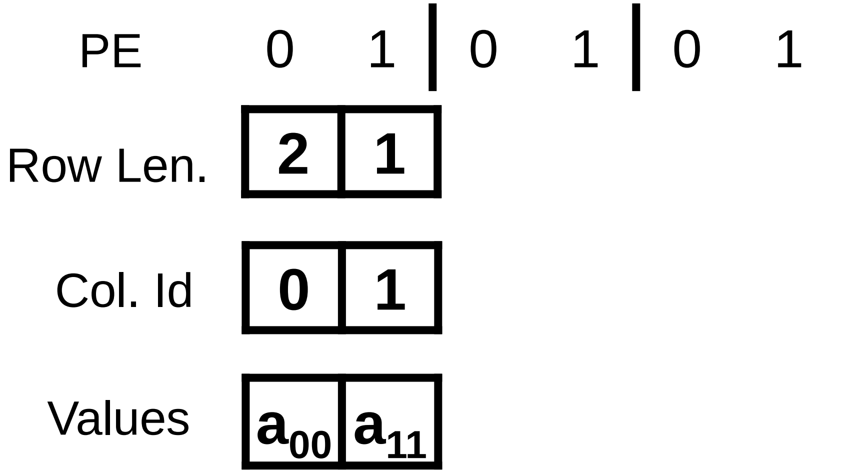




Compressed Sparse Row Format (CSR)

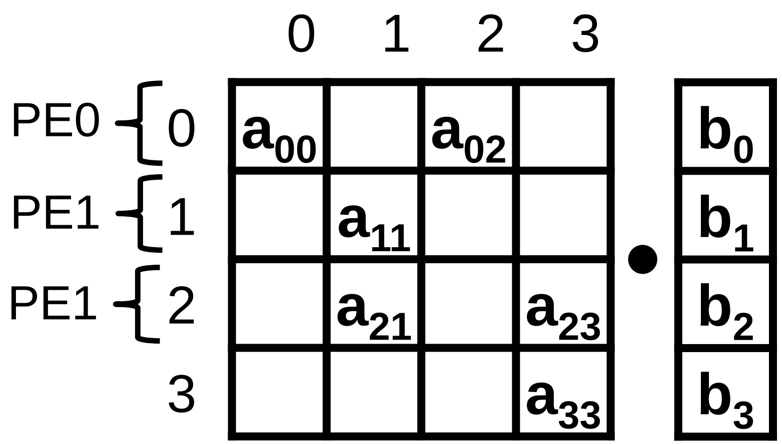


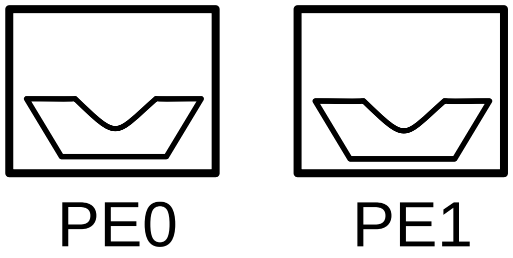
Compressed Interleaved Sparse Row (CISR) [1]



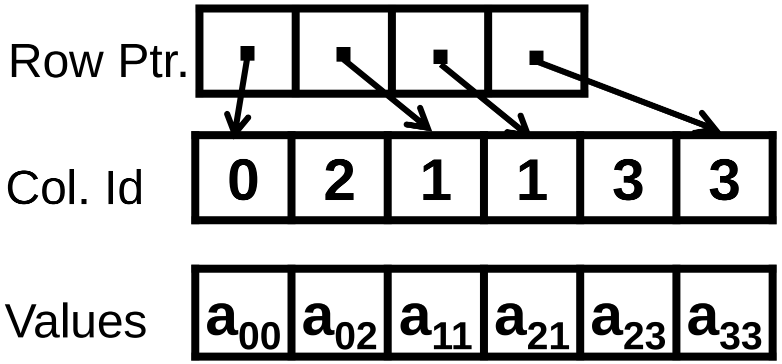
[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Importance of Accelerator Friendly Formats

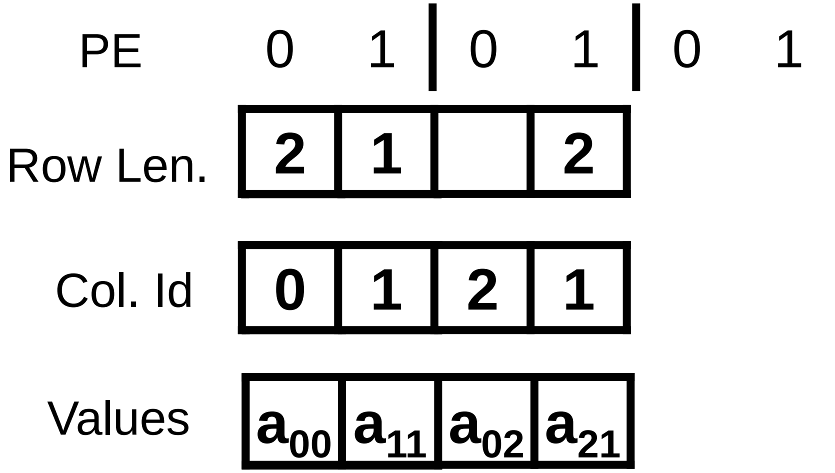




Compressed Sparse Row Format (CSR)



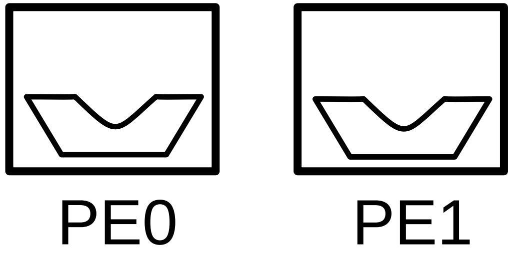
Compressed Interleaved Sparse Row (CISR) [1]



[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Importance of Accelerator Friendly Formats

		0	1	2	3		
PE0	0	a_{00}		a_{02}			b_0
PE1	1		a_{11}			•	b_1
PE1	2		a_{21}		a_{23}		b_2
PE0	3				a_{33}		b_3



Compressed Sparse Row Format (CSR)

Row Ptr.	▪	▪	▪	▪		
Col. Id	0	2	1	1	3	3
Values	a_{00}	a_{02}	a_{11}	a_{21}	a_{23}	a_{33}

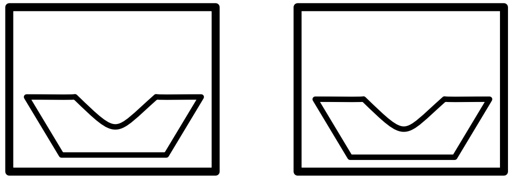
Compressed Interleaved Sparse Row (CISR) [1]

PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

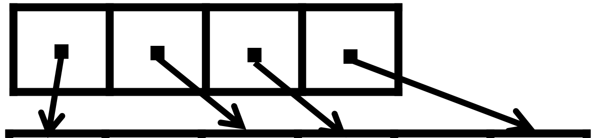
Importance of Accelerator Friendly Formats

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & a_{00} & & a_{02} & \\ 1 & & a_{11} & & \\ 2 & & a_{21} & & a_{23} \\ 3 & & & & a_{33} \end{array} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$



Compressed Sparse Row Format (CSR)

Row Ptr.



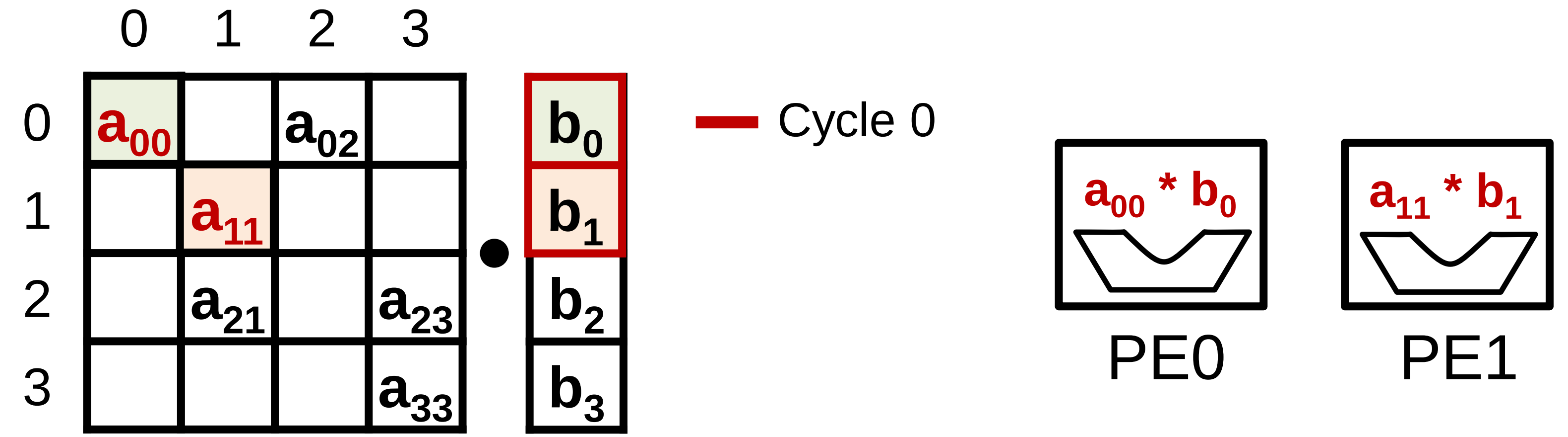
Col. Id	0	2	1	1	3	3
Values	a_{00}	a_{02}	a_{11}	a_{21}	a_{23}	a_{33}

Compressed Interleaved Sparse Row (CISR) [1]

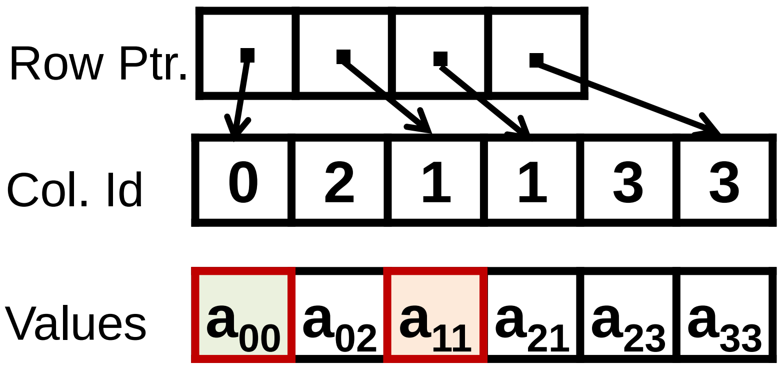
PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Importance of Accelerator Friendly Formats



Compressed Sparse Row Format (CSR)

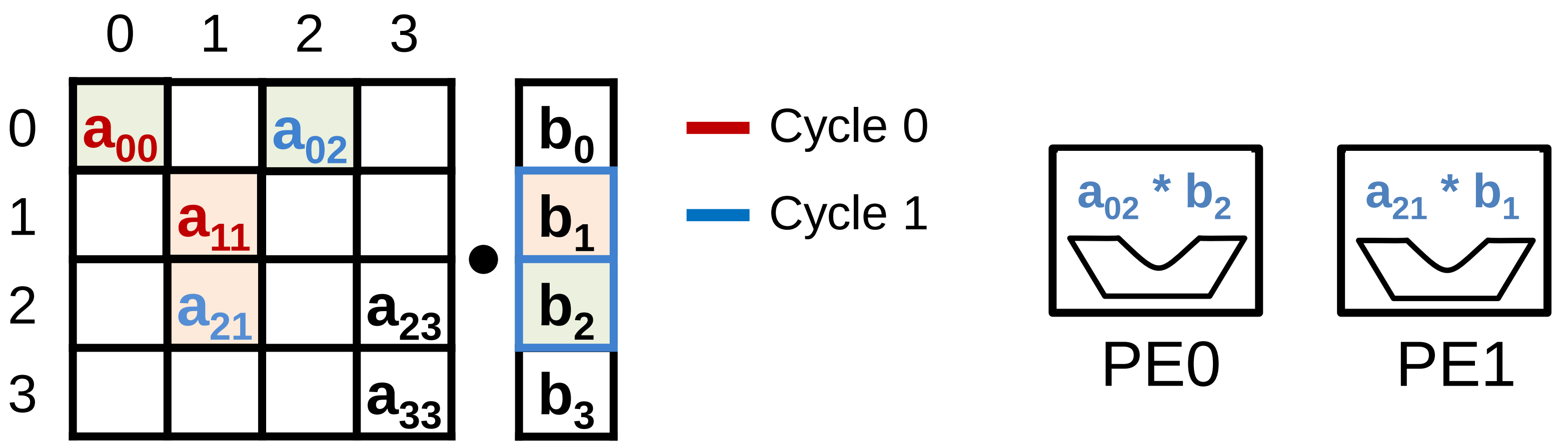


Compressed Interleaved Sparse Row (CISR) [1]

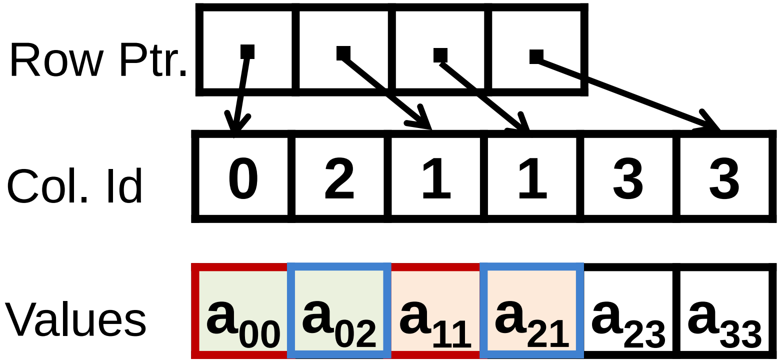
PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

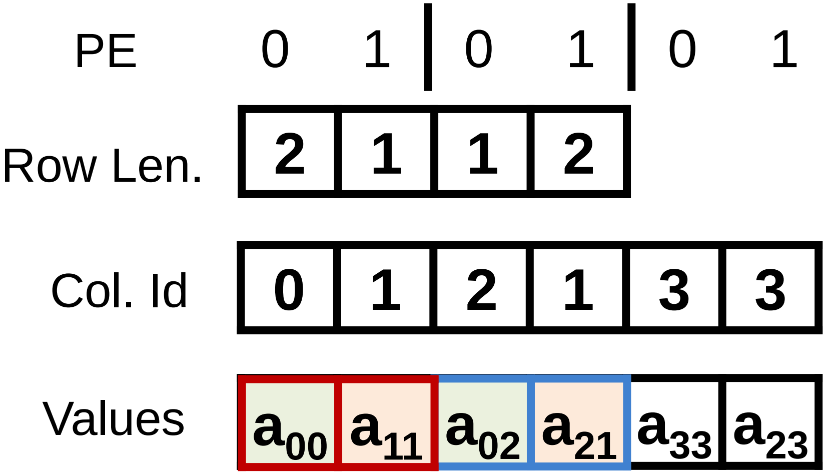
Importance of Accelerator Friendly Formats



Compressed Sparse Row Format (CSR)

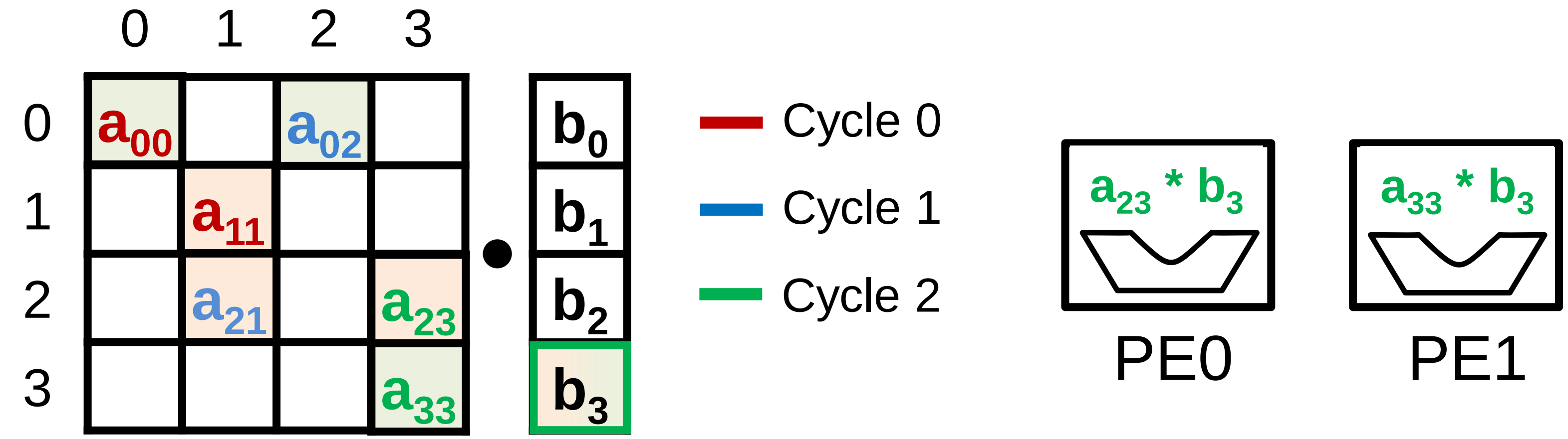


Compressed Interleaved Sparse Row (CISR) [1]

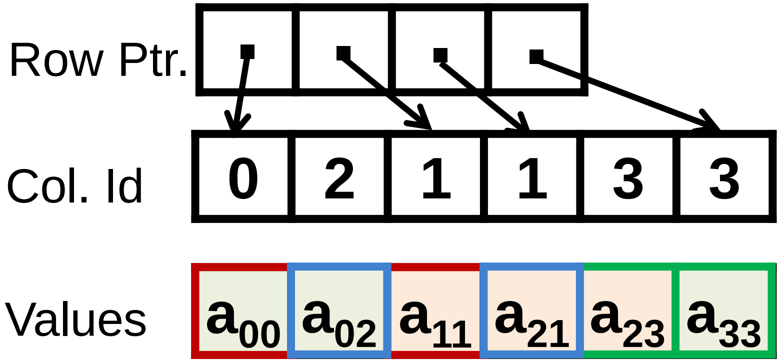


[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Importance of Accelerator Friendly Formats



Compressed Sparse Row Format (CSR)

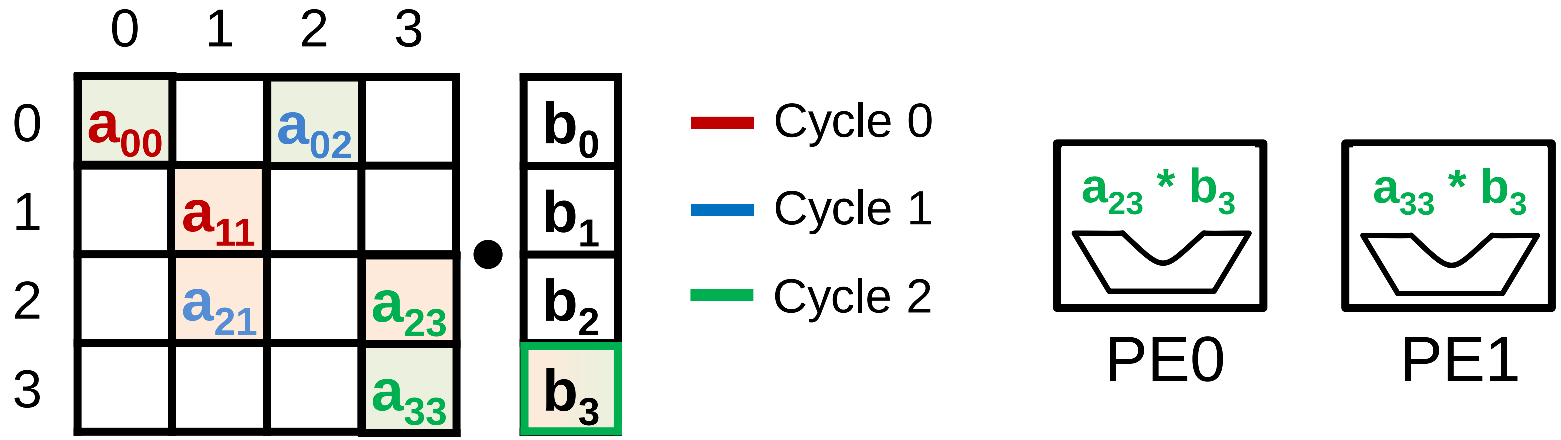


Compressed Interleaved Sparse Row (CISR) [1]

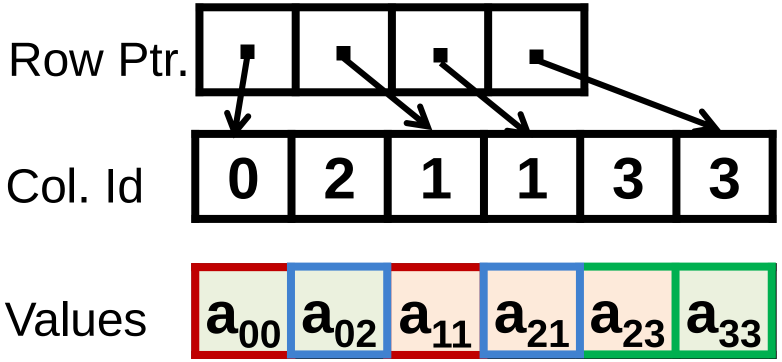
PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

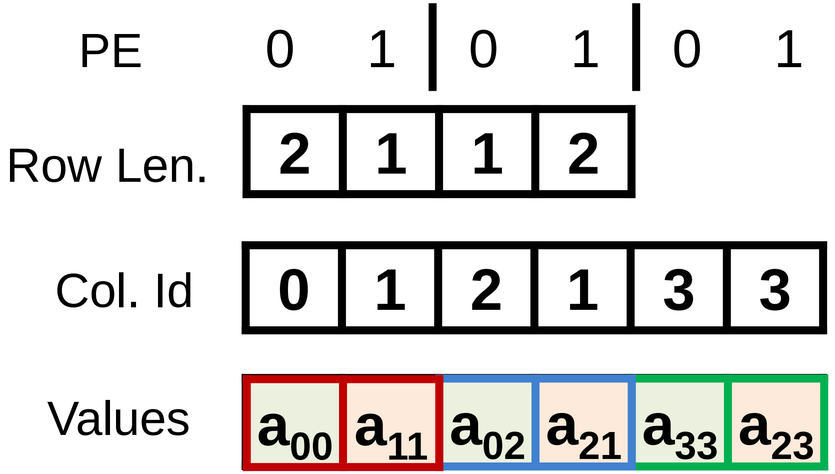
Importance of Accelerator Friendly Formats



Compressed Sparse Row Format (CSR)



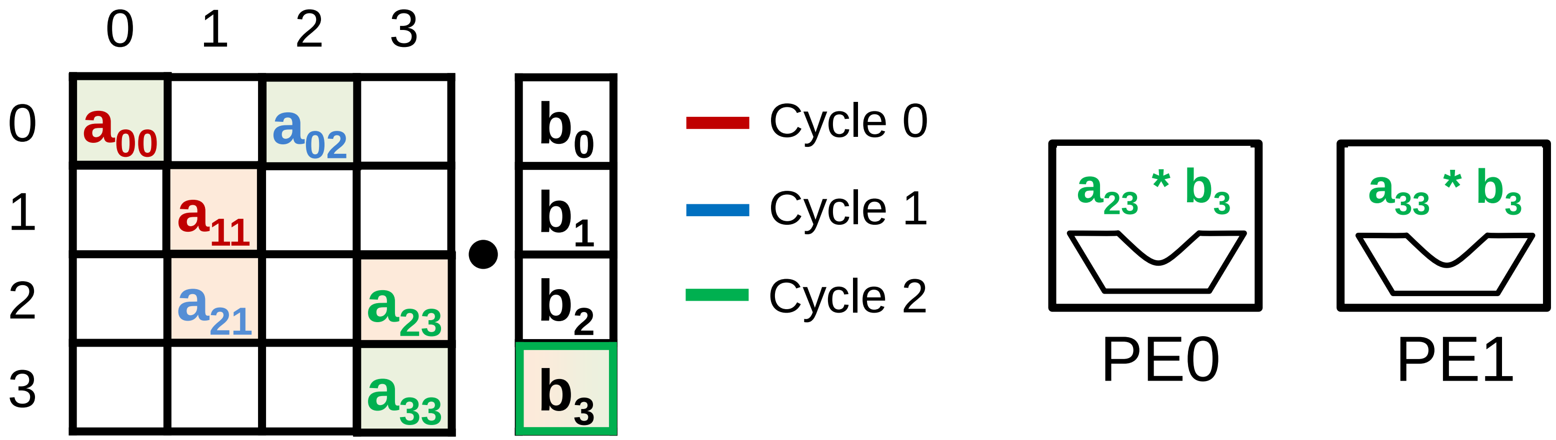
Compressed Interleaved Sparse Row (CISR) [1]



Streaming and vectorized accesses

[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Importance of Accelerator Friendly Formats



Compressed Sparse Row Format (CSR)

Row Ptr.	•	•	•	•		
Col. Id	0	2	1	1	3	3
Values	a_{00}	a_{02}	a_{11}	a_{21}	a_{23}	a_{33}

Non-streaming & non-vectorized accesses

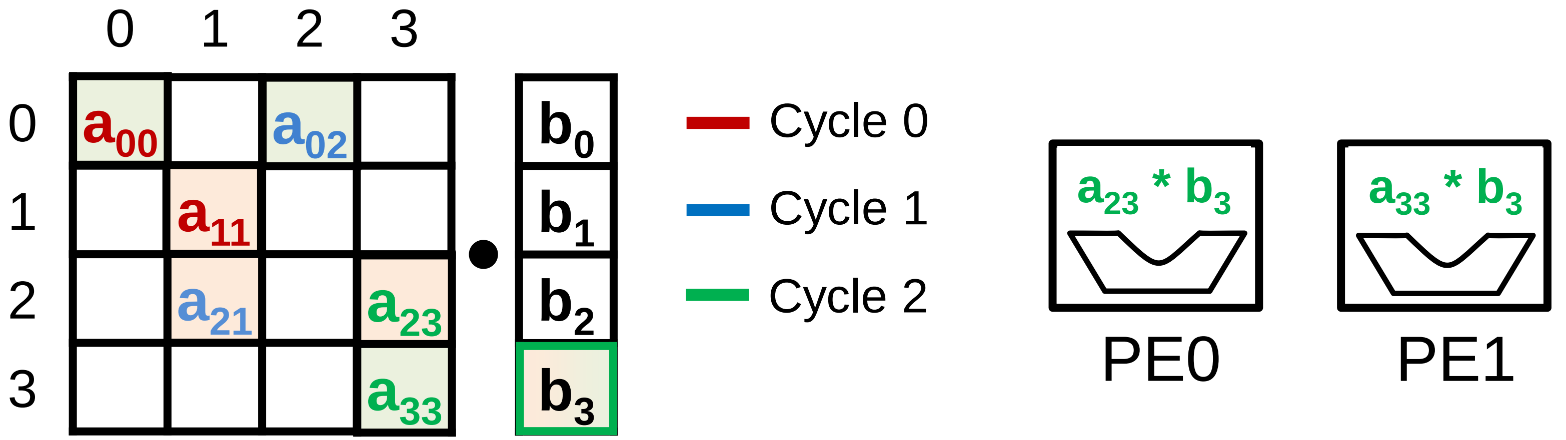
Compressed Interleaved Sparse Row (CISR) [1]

PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

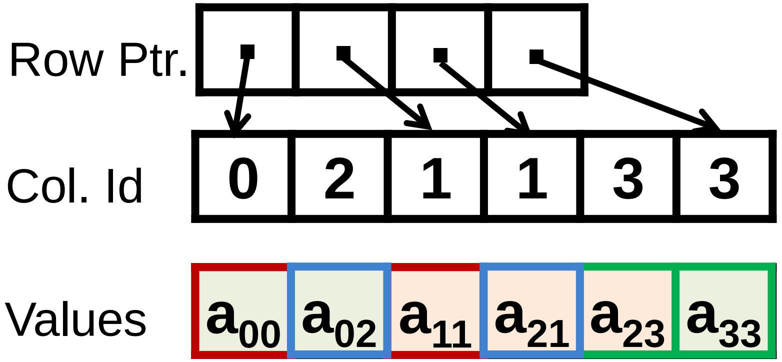
Streaming and vectorized accesses

[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Importance of Accelerator Friendly Formats

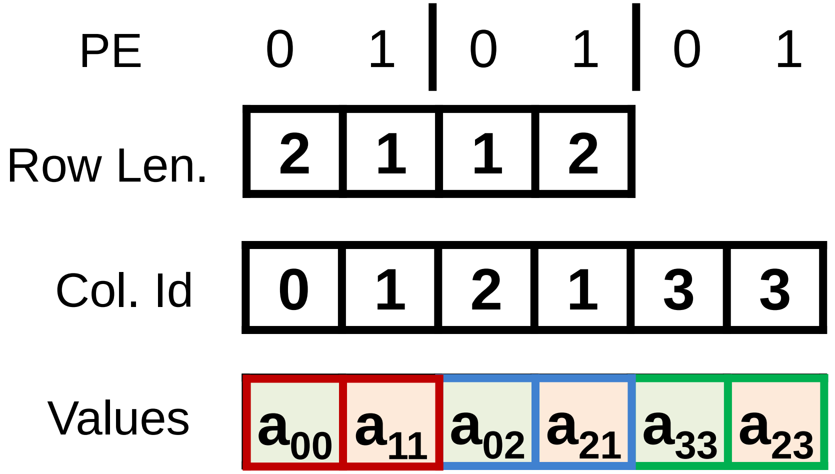


Compressed Sparse Row Format (CSR)



Non-streaming & non-vectorized accesses

Compressed Interleaved Sparse Row (CISR) [1]

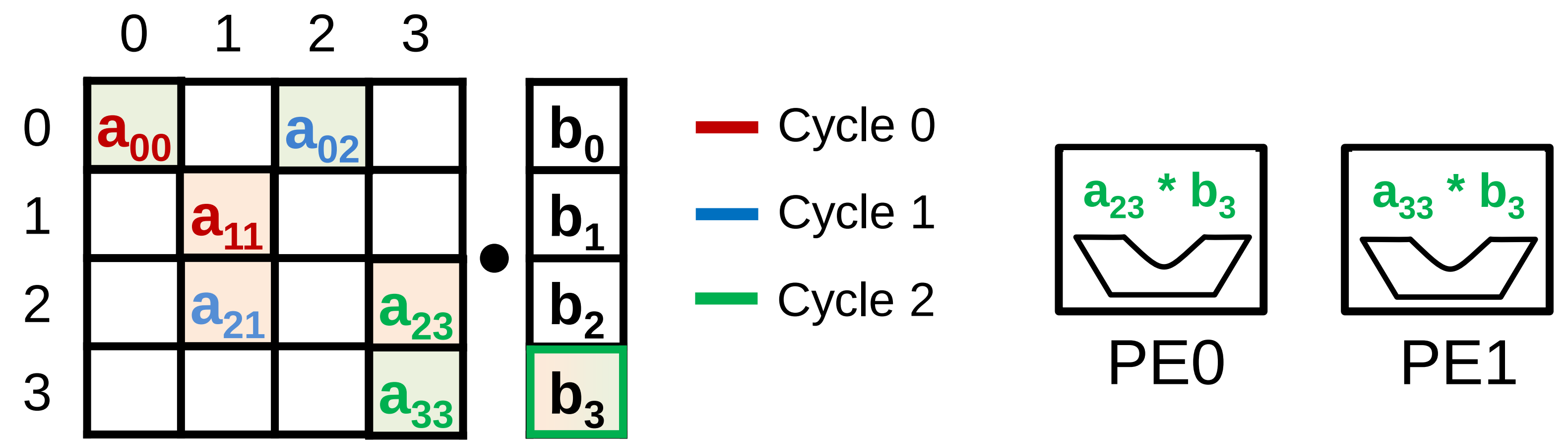


Streaming and vectorized accesses

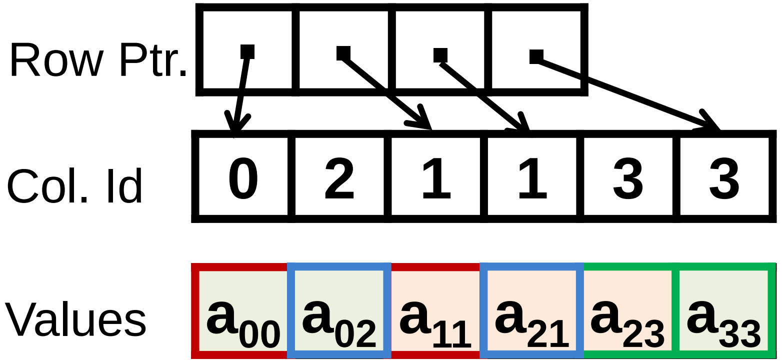
Utilized Bandwidth vs. # PEs

[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Importance of Accelerator Friendly Formats

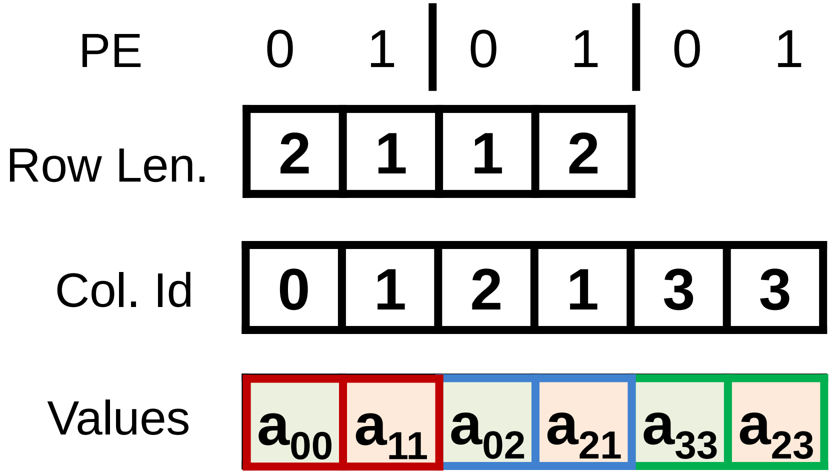


Compressed Sparse Row Format (CSR)

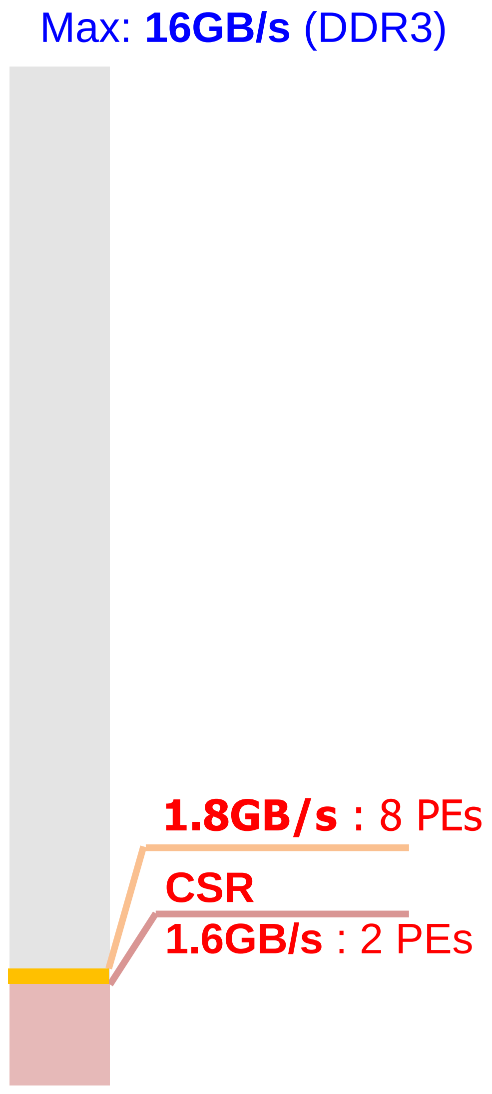


Non-streaming & non-vectorized accesses

Compressed Interleaved Sparse Row (CISR) [1]



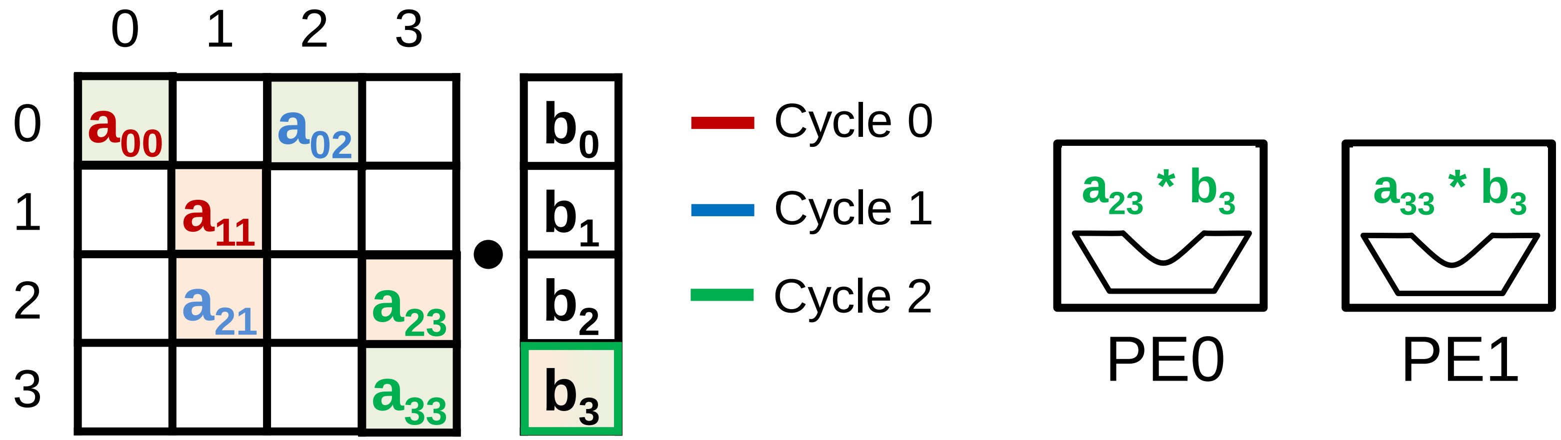
Streaming and vectorized accesses



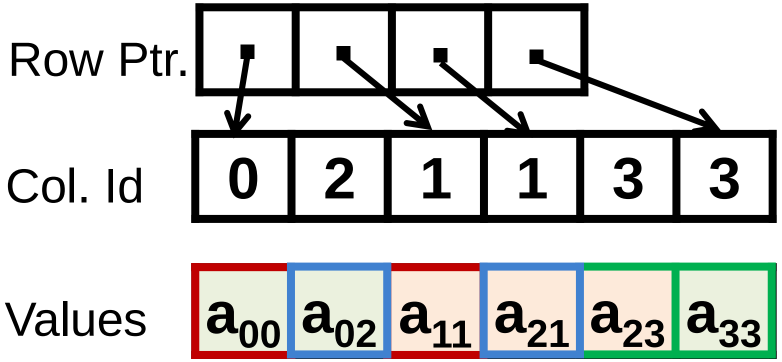
Utilized Bandwidth vs. # PEs

[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Importance of Accelerator Friendly Formats

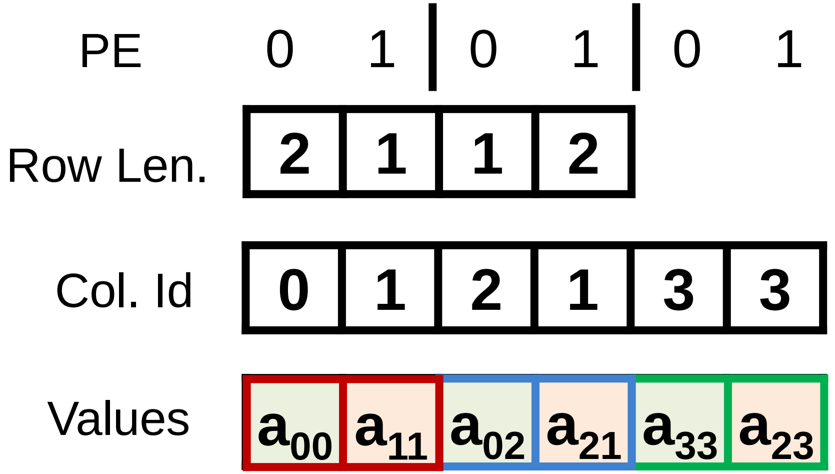


Compressed Sparse Row Format (CSR)

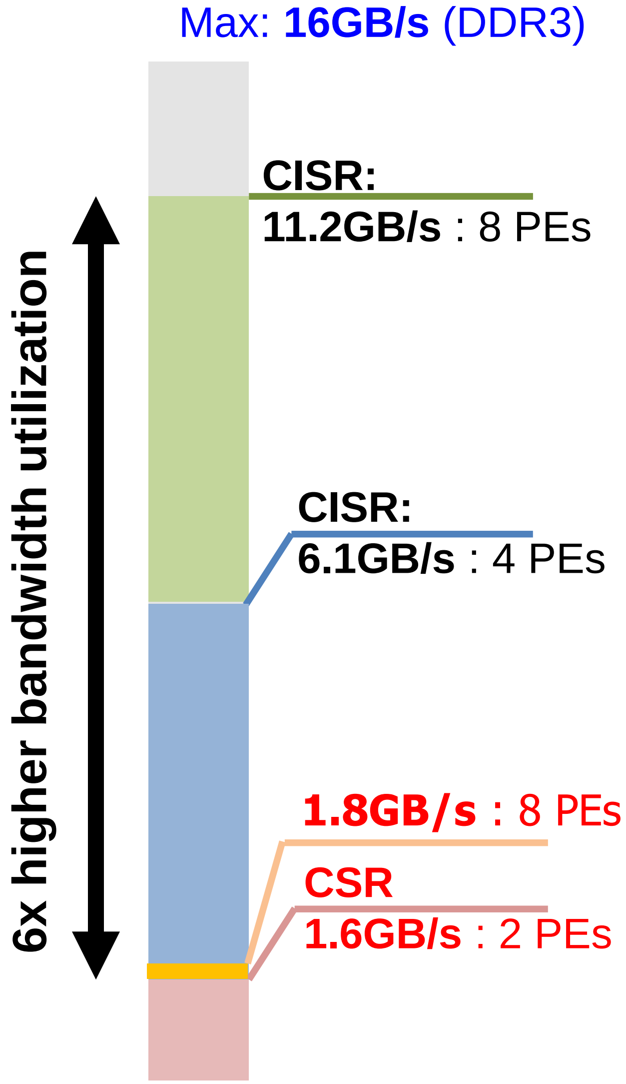


Non-streaming & non-vectorized accesses

Compressed Interleaved Sparse Row (CISR) [1]



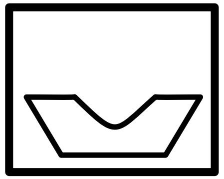
Streaming and vectorized accesses



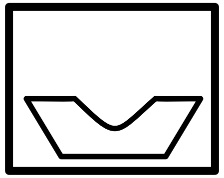
Utilized Bandwidth vs. # PEs

[1] Fowers, et al. A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication, Int'l Symp. On *Field-Programmable Custom Computing Machines (FCCM), 2014*

Making CISR Scalable with CISR+



PE0

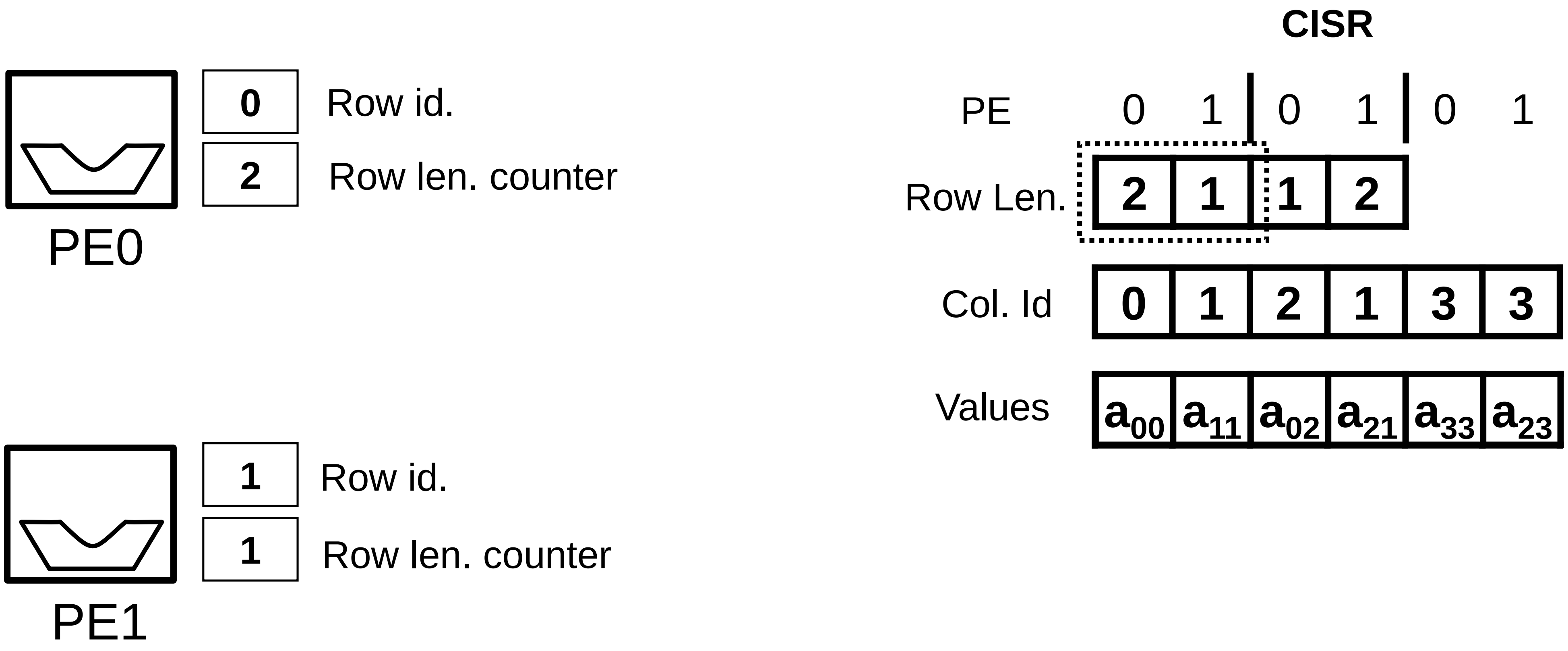


PE1

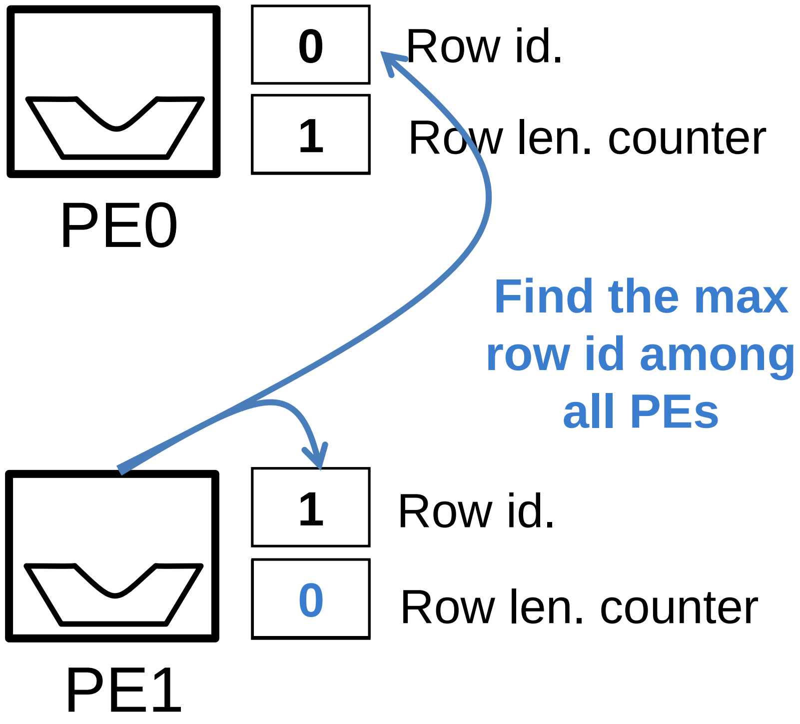
CISR

PE	0	1	0	1	0	1
Row Len.	**2**	**1**	**1**	**2**		
Col. Id	**0**	**1**	**2**	**1**	**3**	**3**
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

Making CISR Scalable with CISR+



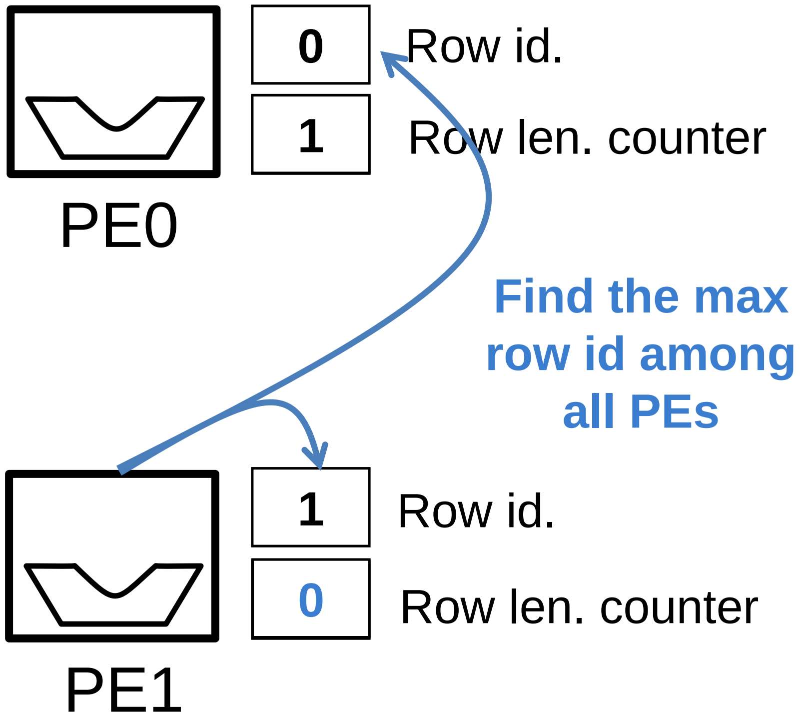
Making CISR Scalable with CISR+



CISR

PE	0	1	0	1	0	1
Row Len.	**2**	**1**	**1**	**2**		
Col. Id	**0**	**1**	**2**	**1**	**3**	**3**
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

Making CISR Scalable with CISR+

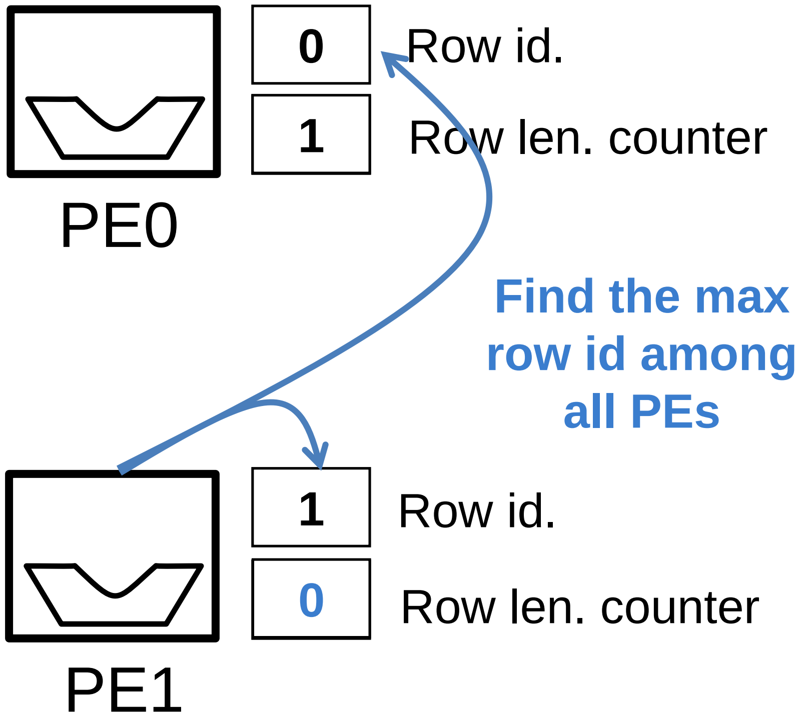


Next row ID = max(row ID) + 1
= 1 + 1
= 2

CISR

PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

Making CISR Scalable with CISR+



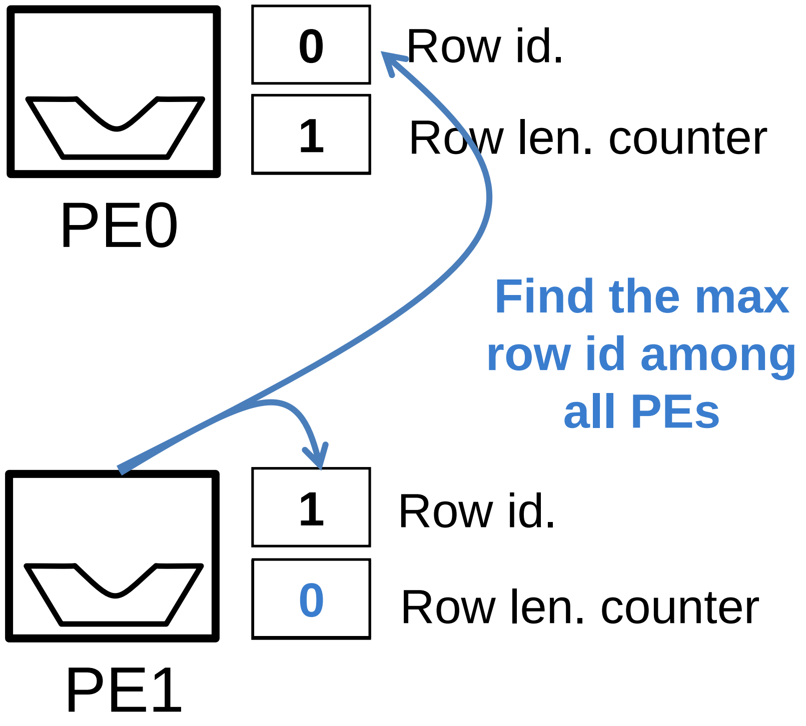
CISR

PE	0	1	0	1	0	1
Row Len.	**2**	**1**	**1**	**2**		
Col. Id	**0**	**1**	**2**	**1**	**3**	**3**
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

Next row ID = max(row ID) + 1
= 1 + 1
= 2

CISR requires centralized row decoding
→ Not scalable

Making CISR Scalable with CISR+



Next row ID = max(row ID) + 1
= 1 + 1
= 2

CISR

PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

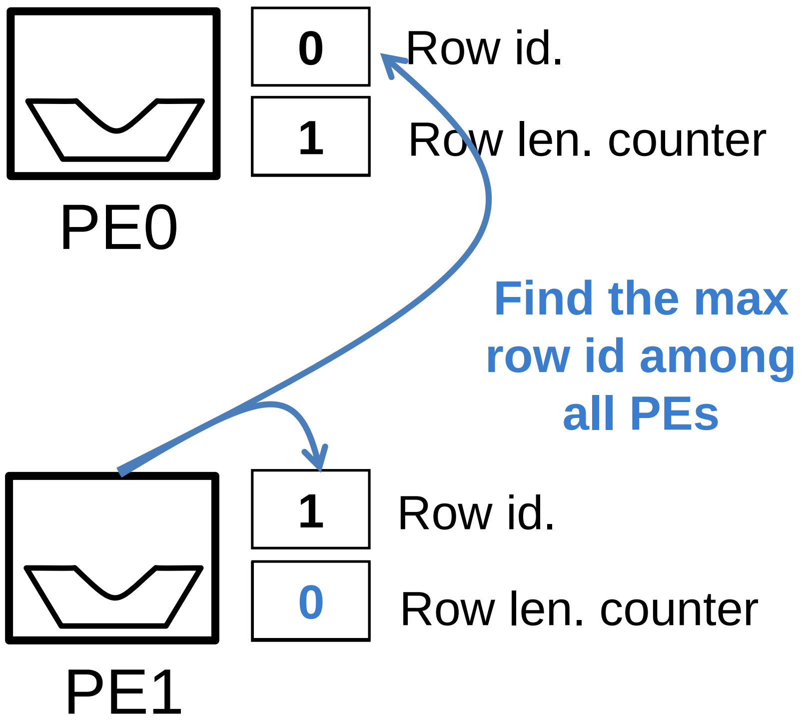
Make Scalable

CISR+

PE	0	1	0	1	0	1	0	1	0	1
Row Id/ Col. Id	0	1	0	1	2	2	3	1	3	3
Values	0	0	a_{00}	a_{11}	a_{02}	0	0	a_{21}	a_{33}	a_{23}

CISR requires centralized row decoding
→ Not scalable

Making CISR Scalable with CISR+



Next row ID = max(row ID) + 1
= 1 + 1
= 2

CISR

PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

Make Scalable

CISR+

PE	0	1	0	1	0	1	0	1	0	1
Row Id/ Col. Id	0	1	0	1	2	2	3	1	3	3
Values	0	0	a_{00}	a_{11}	a_{02}	0	0	a_{21}	a_{33}	a_{23}

CISR requires centralized row decoding
→ Not scalable

CISR+ has negligible storage overhead as an extra zero is sent only once per row

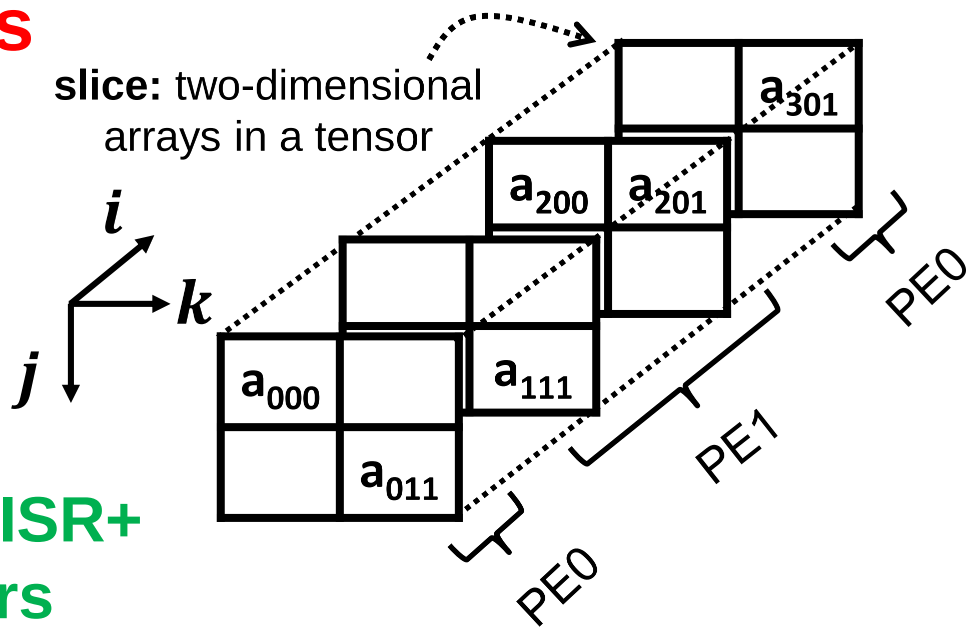
Extending CISR+ to CISS for Tensors

PE	Row	0	1	2	3
PE0	0	a_{00}		a_{02}	
PE1	1		a_{11}		
PE1	2		a_{21}		a_{23}
PE0	3				a_{33}

CISR+

PE	0	1	0	1	0	1	0	1	0	1
Row Id/ Col. Id	**0**	**1**	**0**	**1**	**2**	**2**	**3**	**1**	**3**	**3**
Values	**0**	**0**	a_{00}	a_{11}	a_{02}	**0**	**0**	a_{21}	a_{33}	a_{23}

Extending CISR+ to Tensors



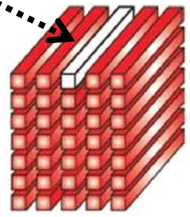
Compressed Interleaved Sparse Slice (CISS)

PE	0	1	0	1	0	1	0	1	0	1
k	x	x	0	1	1	x	x	0	1	1
i/j	**0**	**1**	**0**	**1**	**1**	**2**	**3**	**0**	**0**	**0**
Values	**0**	**0**	a_{000}	a_{111}	a_{011}	**0**	**0**	a_{200}	a_{301}	a_{201}

Computation Pattern for Tensor Kernels

Computation Pattern for Tensor Kernels

fiber: one-dimensional arrays along a tensor dimension

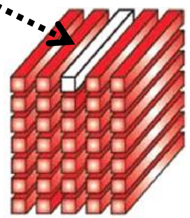


Scalar-Fiber product followed by Fiber-Fiber product (SF^3) Pattern

$$\boxed{\boldsymbol{fibers_{out}} = \sum_{D_1} \boldsymbol{fiber_1}\ op \sum_{D_0} (\boldsymbol{scalar} \,.\, \boldsymbol{fiber_0})}$$

Computation Pattern for Tensor Kernels

fiber: one-dimensional arrays along a tensor dimension



Scalar-Fiber product followed by Fiber-Fiber product (SF³) Pattern

$$\boxed{\boldsymbol{fibers}_{\boldsymbol{out}} = \sum_{D_1} \boldsymbol{fiber}_{\mathbf{1}}\ op \sum_{D_0} (\boldsymbol{scalar} \,.\, \boldsymbol{fiber}_{\mathbf{0}})}$$

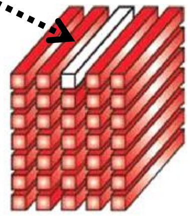
MTTKRP $\forall i \quad \boldsymbol{Y}(\boldsymbol{i},:) = \sum_{D_1} \boldsymbol{B}(\boldsymbol{j},:) \circ \sum_{D_0} (\boldsymbol{A}(\boldsymbol{i},\boldsymbol{j},\boldsymbol{k}) \,.\, \boldsymbol{C}(\boldsymbol{k},:))$

D_1: $[0, J)$ **or** $\{j \mid \exists k\ st.\ A(i,j,k) \neq 0\}$

D_0: $[0, K)$ **or** $\{k \mid A(i,j,k) \neq 0\}$

Computation Pattern for Tensor Kernels

fiber: one-dimensional arrays along a tensor dimension



Scalar-Fiber product followed by Fiber-Fiber product (SF³) Pattern

$$\boxed{\boldsymbol{fibers_{out}} = \sum_{D_1} \boldsymbol{fiber_1}\ op \sum_{D_0} (\boldsymbol{scalar} \,.\, \boldsymbol{fiber_0})}$$

MTTKRP $\forall i \quad \boldsymbol{Y(i,:)} = \sum_{D_1} \boldsymbol{B(j,:)} \circ \sum_{D_0} (\boldsymbol{A(i,j,k)} \,.\, \boldsymbol{C(k,:)})$

D_1: $[0, J)$ **or** $\{j \mid \exists k\ st.\ A(i,j,k) \neq 0\}$

D_0: $[0, K)$ **or** $\{k \mid A(i,j,k) \neq 0\}$

Similarly

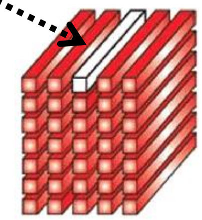
TTMc $\forall i \quad \boldsymbol{Y(i,:)} = \sum_{D_1} \boldsymbol{B(j,:)} \otimes \sum_{D_0} (\boldsymbol{A(i,j,k)} \,.\, \boldsymbol{C(k,:)})$

MM $\forall i \quad \boldsymbol{Y(i,:)} = \sum_{\phi} \boldsymbol{null}\ \ op \sum_{D_0} (\boldsymbol{A(i,j)} \,.\, \boldsymbol{B(j,:)})$

MV $\forall i \quad \boldsymbol{Y(i,:)} = \sum_{\phi} \boldsymbol{null}\ \ op \sum_{D_0} (\boldsymbol{A(i,j)} \,.\, \boldsymbol{b(j,:)})$

Computation Pattern for Tensor Kernels

fiber: one-dimensional arrays along a tensor dimension



Scalar-Fiber product followed by Fiber-Fiber product (SF³) Pattern

$$\boldsymbol{fibers_{out}} = \sum_{D_1} \boldsymbol{fiber_1}\ op \sum_{D_0} (\boldsymbol{scalar} \,.\, \boldsymbol{fiber_0})$$

MTTKRP $\quad \forall i \;\; \boldsymbol{Y(i,:)} = \sum_{D_1} \boldsymbol{B(j,:)} \circ \sum_{D_0} (\boldsymbol{A(i,j,k)} \,.\, \boldsymbol{C(k,:)})$

D_1: $[0, J)$ **or** $\{j \mid \exists k\ st.\ A(i,j,k) \neq 0\}$

D_0: $[0, K)$ **or** $\{k \mid A(i,j,k) \neq 0\}$

Similarly

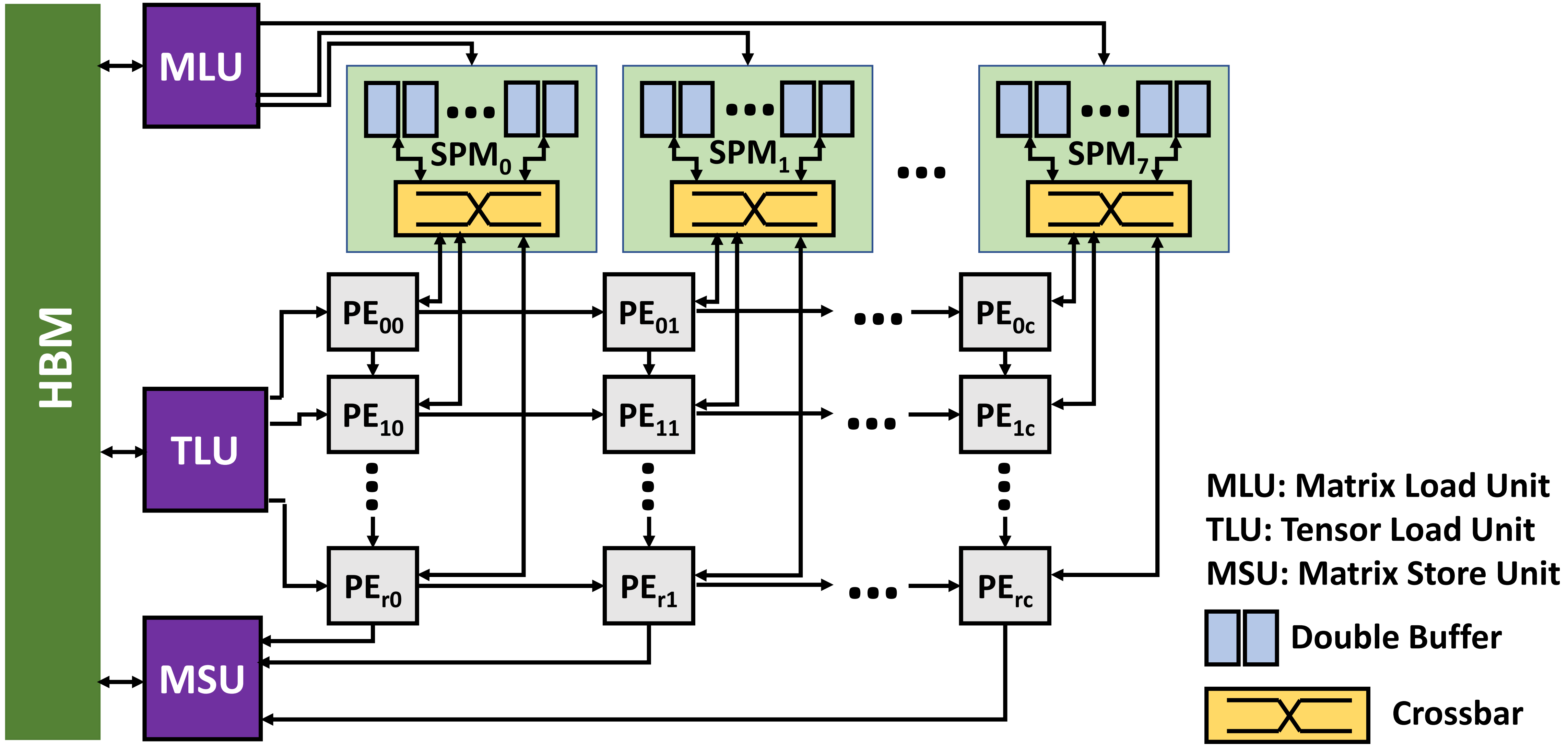
TTMc $\quad \forall i \;\; \boldsymbol{Y(i,:)} = \sum_{D_1} \boldsymbol{B(j,:)} \otimes \sum_{D_0} (\boldsymbol{A(i,j,k)} \,.\, \boldsymbol{C(k,:)})$

MM $\quad \forall i \;\; \boldsymbol{Y(i,:)} = \sum_{\phi} \boldsymbol{null}\ \ op \sum_{D_0} (\boldsymbol{A(i,j)} \,.\, \boldsymbol{B(j,:)})$

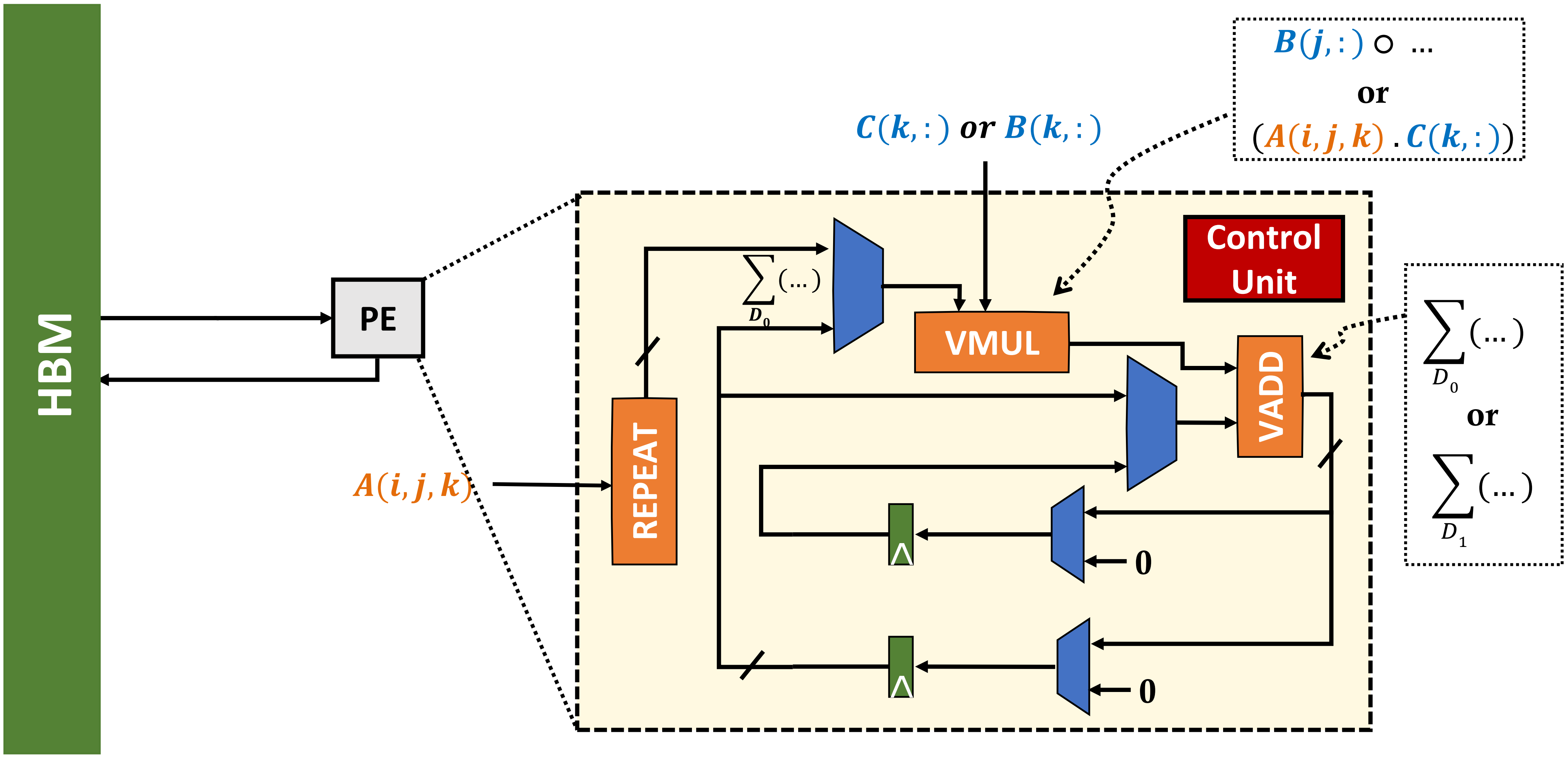
MV $\quad \forall i \;\; \boldsymbol{Y(i,:)} = \sum_{\phi} \boldsymbol{null}\ \ op \sum_{D_0} (\boldsymbol{A(i,j)} \,.\, \boldsymbol{b(j,:)})$

SF³ compute pattern can express all the common dense and mixed sparse-dense tensor kernels

Tensaurus Micro-architecture

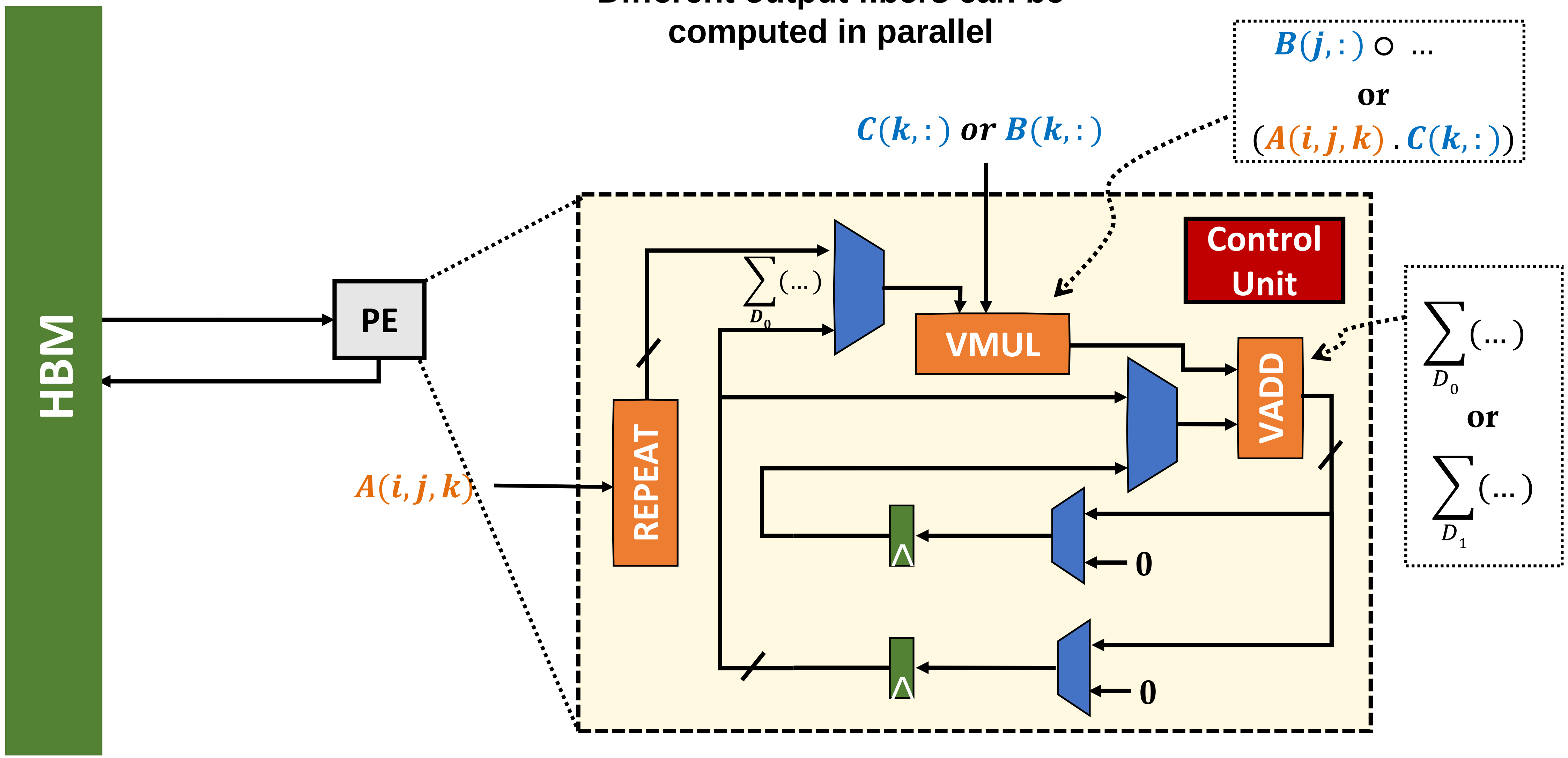


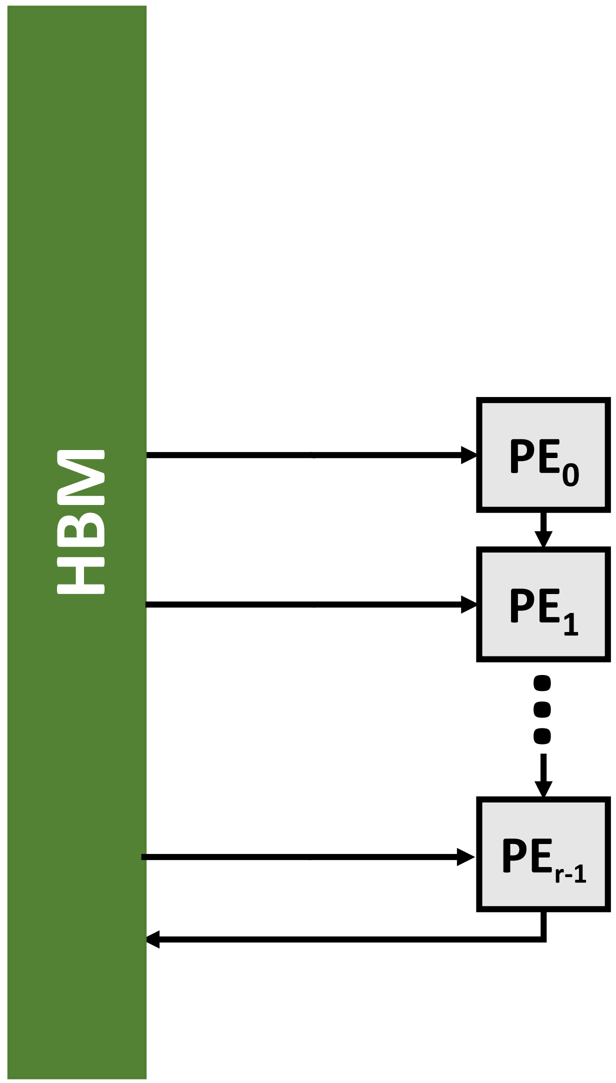
PE: $\forall i \in [0, I)\ \ \boldsymbol{Y(i,:)} = \sum_{D_1} \boldsymbol{B(j,:)} \circ \sum_{D_0} (\boldsymbol{A(i,j,k)} \cdot \boldsymbol{C(k,:)})$



PE: $\underbrace{\forall i \in [0, I) \;\; Y(i,:)}_{} = \sum_{D_1} B(j,:) \circ \sum_{D_0} (A(i,j,k) \,.\, C(k,:))$

Different output fibers can be computed in parallel



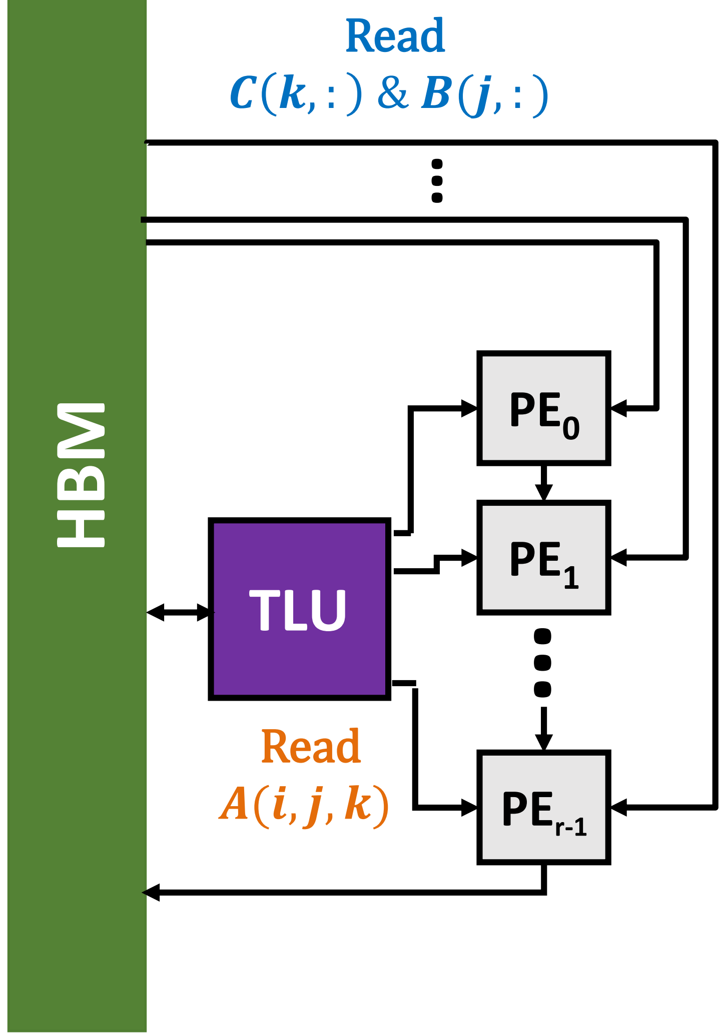


$$\mathbf{PE_0}: \quad \forall i \in \left[0, \frac{I}{r}\right) \qquad Y(i,:) = \sum_{D_1} B(j,:) \circ \sum_{D_0} \left(A(i,j,k) \,.\, C(k,:)\right)$$

$$\mathbf{PE_1}: \quad \forall i \in \left[\frac{I}{r}, \frac{2I}{r}\right) \qquad Y(i,:) = \sum_{D_1} B(j,:) \circ \sum_{D_0} \left(A(i,j,k) \,.\, C(k,:)\right)$$

$$\vdots$$

$$\mathbf{PE_{r-1}}: \quad \forall i \in \left[\frac{(r-1)I}{r}, I\right) \qquad Y(i,:) = \sum_{D_1} B(j,:) \circ \sum_{D_0} \left(A(i,j,k) \,.\, C(k,:)\right)$$



PE_0: $\forall i \in \left[0, \frac{I}{r}\right)$ $\quad Y(i,:) = \sum_{D_1} B(j,:) \circ \sum_{D_0} (A \quad) . C(k,:))$

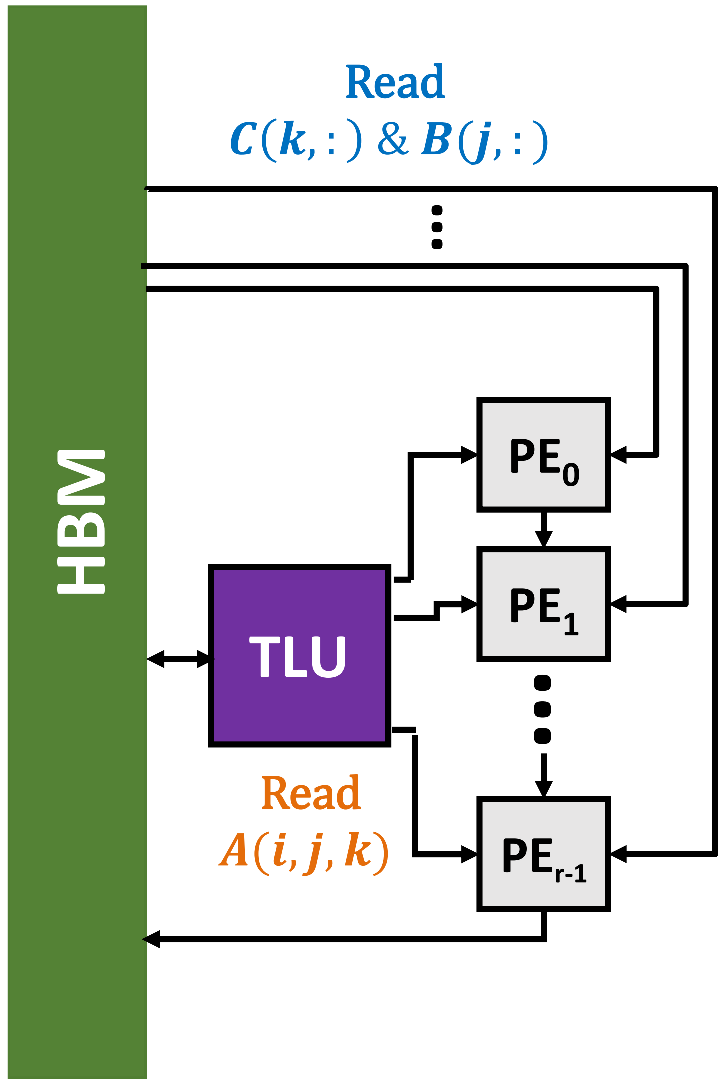
PE_1: $\forall i \in \left[\frac{I}{r}, \frac{2I}{r}\right)$ $\quad Y(i,:) = \sum_{D_1} B(j,:) \circ \sum_{D_0} (\quad . C(k,:))$

Use CISS

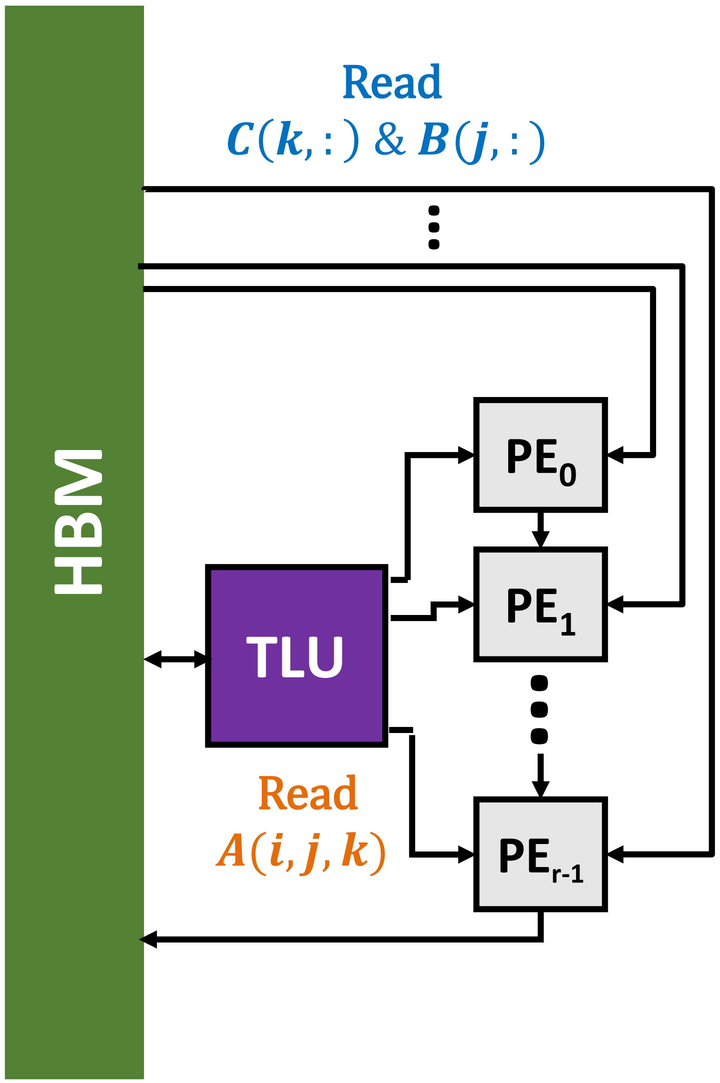
⋮

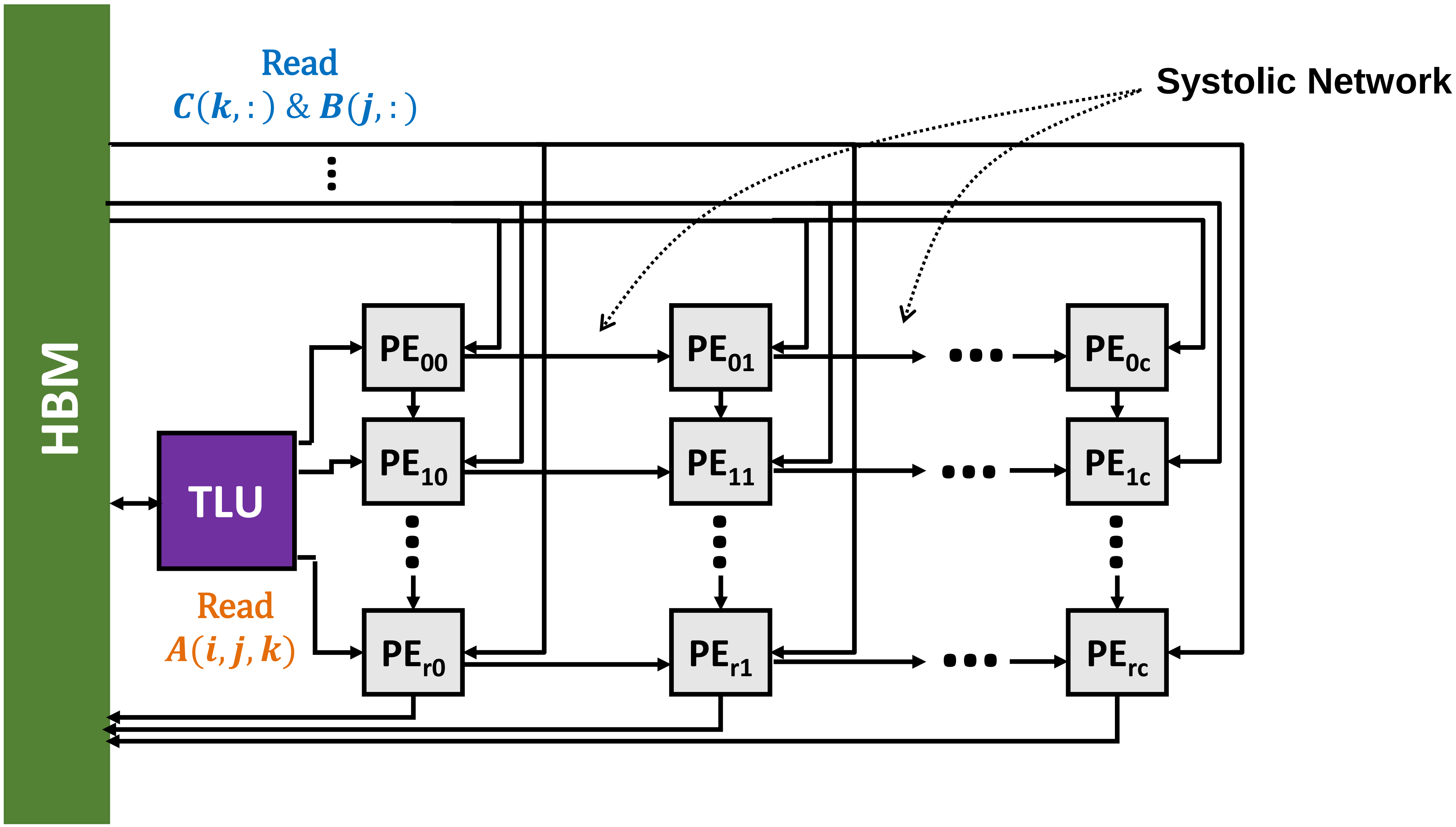
PE_{r-1}: $\forall i \in \left[\frac{(r-1)I}{r}, I\right)$ $\quad Y(i,:) = \sum_{D_1} B(j,:) \circ \sum_{D_0} (\quad) . C(k,:))$

$$\mathbf{PE_0:} \quad \forall i \in \left[0, \frac{I}{r}\right) \quad \boldsymbol{Y}(\boldsymbol{i},:) = \sum_{D_1} \boldsymbol{B}(\boldsymbol{j},:) \circ \sum_{D_0} (\boldsymbol{A}(\boldsymbol{i},\boldsymbol{j},\boldsymbol{k}) \,.\, \boldsymbol{C}(\boldsymbol{k},:))$$



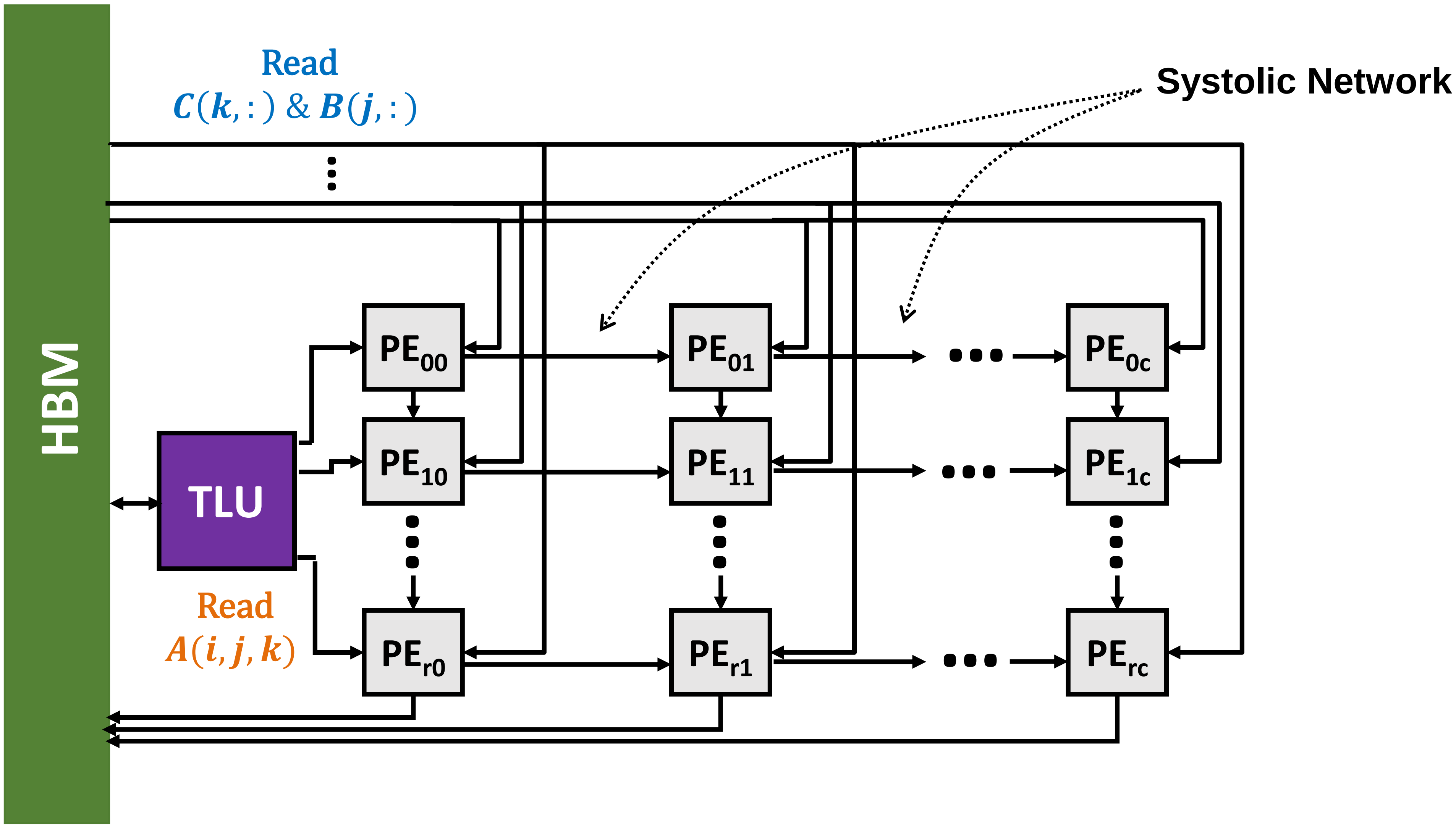
$$\mathbf{PE_0:}\quad \forall i \in \left[0, \frac{I}{r}\right) \quad \underbrace{Y(i,:) = \sum_{D_1} B(j,:) \circ \sum_{D_0} \left(A(i,j,k) \,.\, C(k,:)\right)}_{\text{Can split into small SIMD computations}}$$



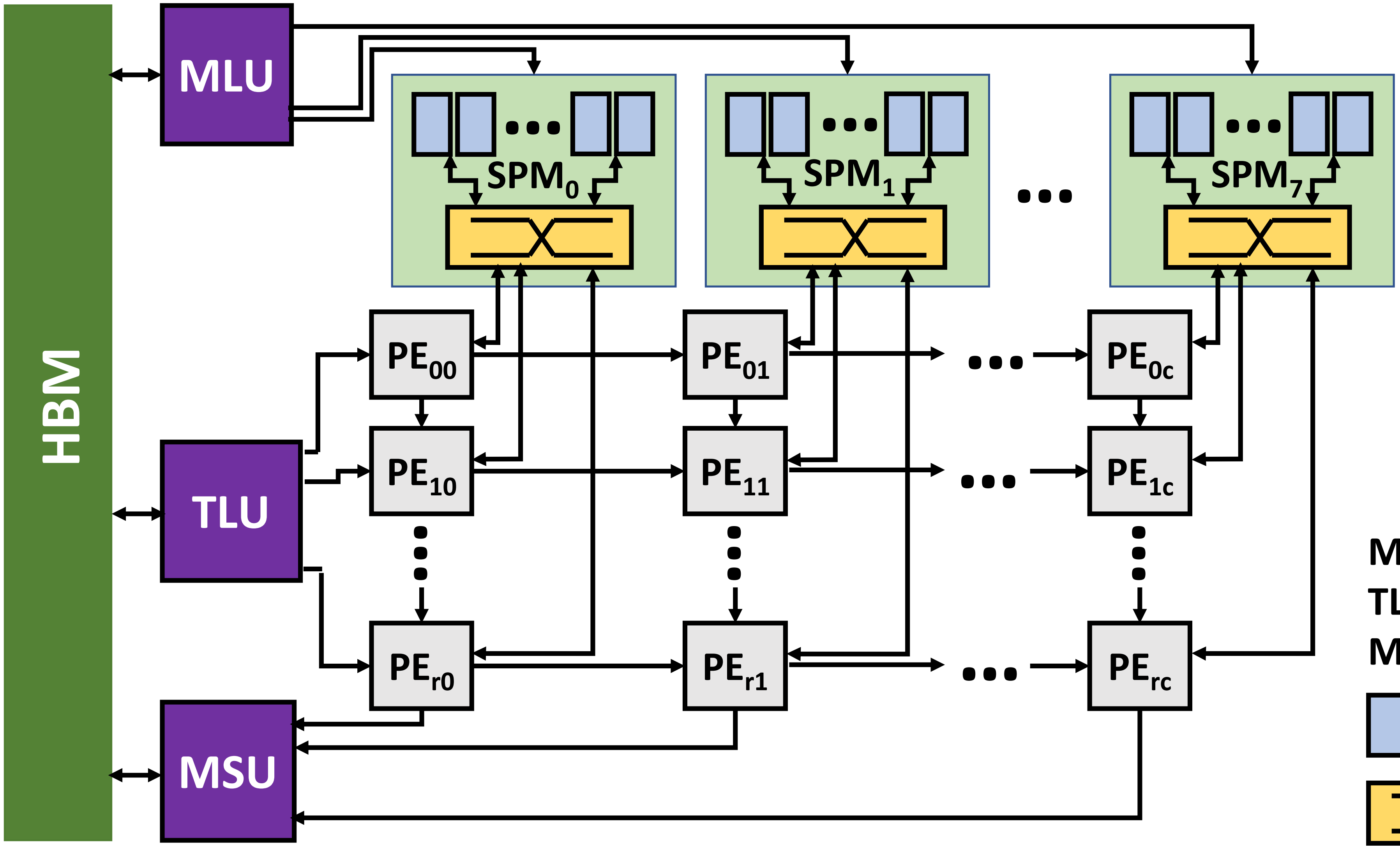


Read
$C(k,:)$ & $B(j,:)$
Systolic Network
HBM
TLU
Read
$A(i,j,k)$
PE_{00}
PE_{01}
PE_{0c}
PE_{10}
PE_{11}
PE_{1c}
PE_{r0}
PE_{r1}
PE_{rc}

B & C are frequently reused → cache on-chip



Tensaurus Micro-architecture



Tensaurus can also accelerate dense tensor kernels

MLU: Matrix Load Unit
TLU: Tensor Load Unit
MSU: Matrix Store Unit

Double Buffer

Crossbar

Evaluation Methodology

- **Cycle-level simulation in gem5**
  - 8 x 8 PE array, VLEN = 8
  - 8 16KB RAMs per SPM
  - HBM: 8 128-bit physical channels (128 GB/s peak bandwidth)

- **RTL Modeling of a PE using PyMTL**

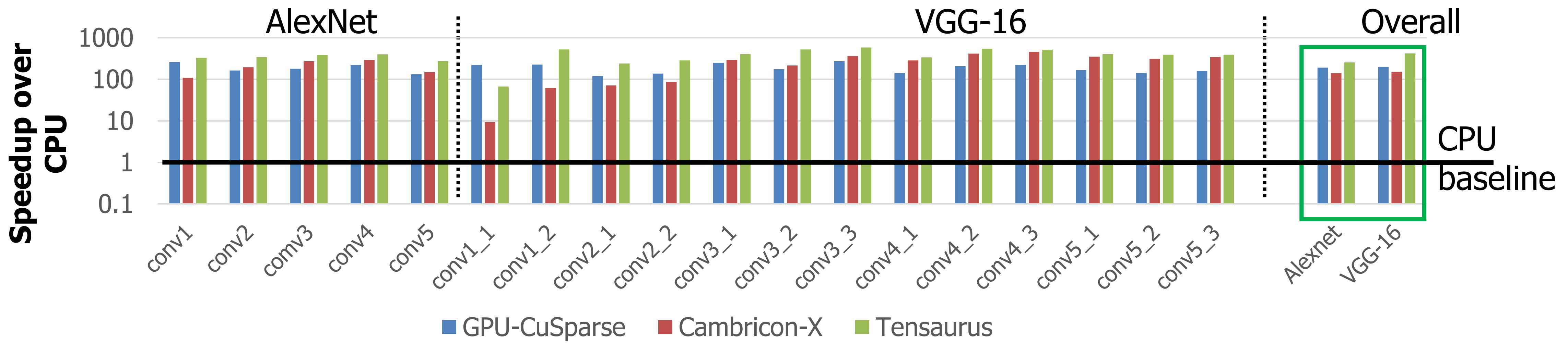
- **Baselines**
  - CPU: Intel(R) Xeon(R) CPU E7-8867
    - SparseBLAS and SPLATT
  - GPU: Titan XP
    - CuSparse, PaRTI
  - Accelerator:
    - Cambricon-X

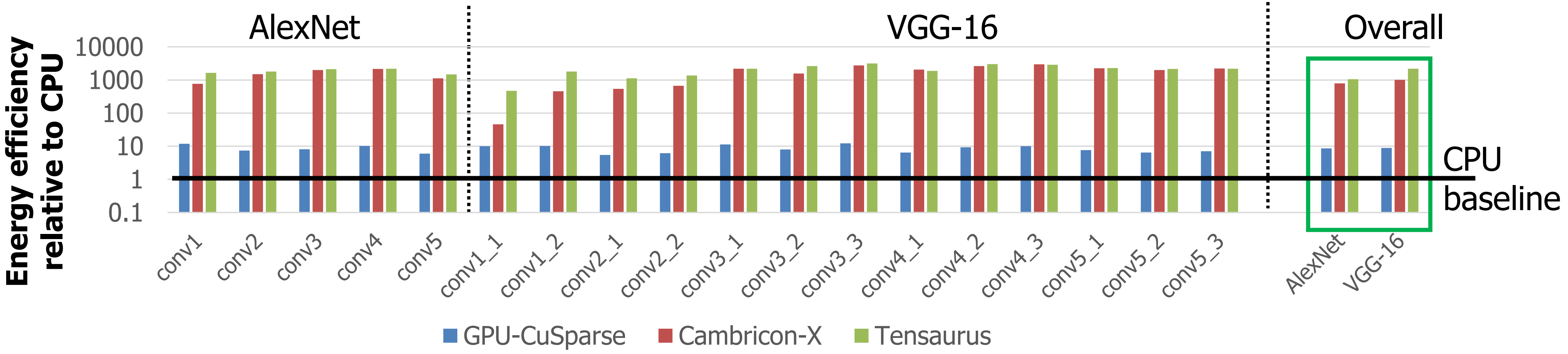
Area and Power Breakdown

Component	Area(mm^2)	%	Power (mW)	%
PE	0.625	27.2 %	402.30	40.9 %
Xbar	0.066	2.8 %	24.27	2.5%
SPM	0.832	36.2 %	296.05	30.1 %
MSU	0.759	33.0 %	247.03	25.2 %
TLU	0.009	0.4 %	6.28	0.6%
MLU	0.009	0.4 %	6.28	0.6 %
Total	2.3	100 %	982.21	100 %

- **Datasets**
  - FROSTT Tensors, Florida Sparse Matrices, AlexNet, VGG-16

Results on Sparse Neural Nets

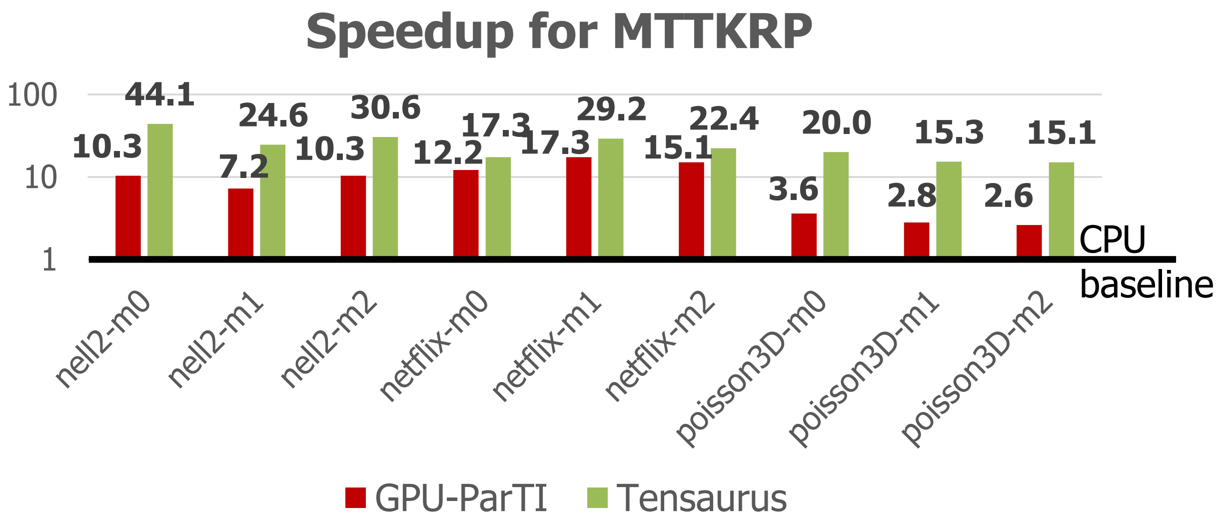


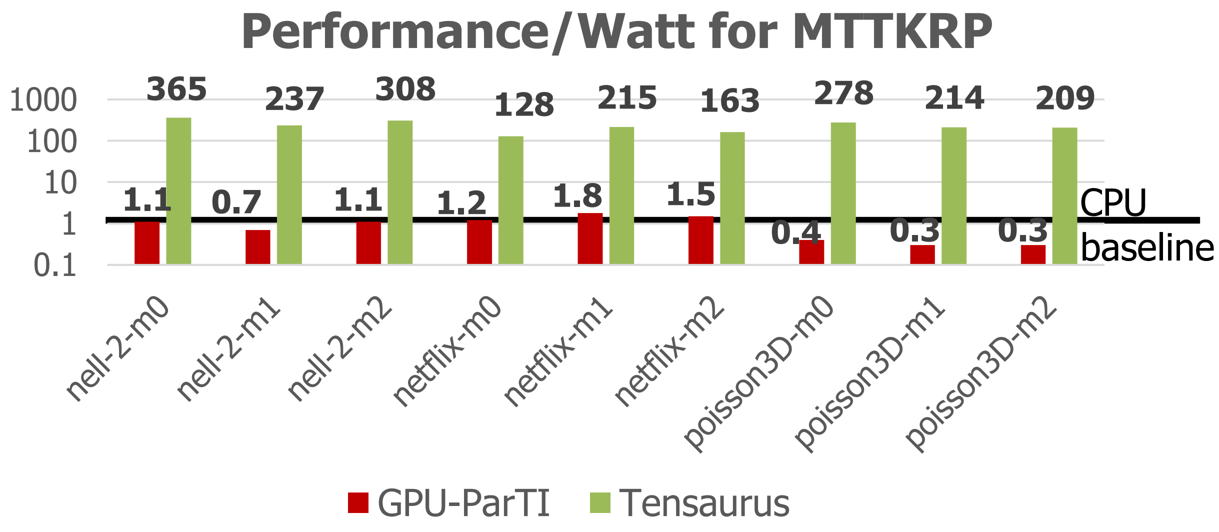


Overall Tensaurus is 349.2x, 1.8x and 1.9x faster and
1900x, 220x and 1.7x more energy-efficient than CPU, GPU and Cambricon-X

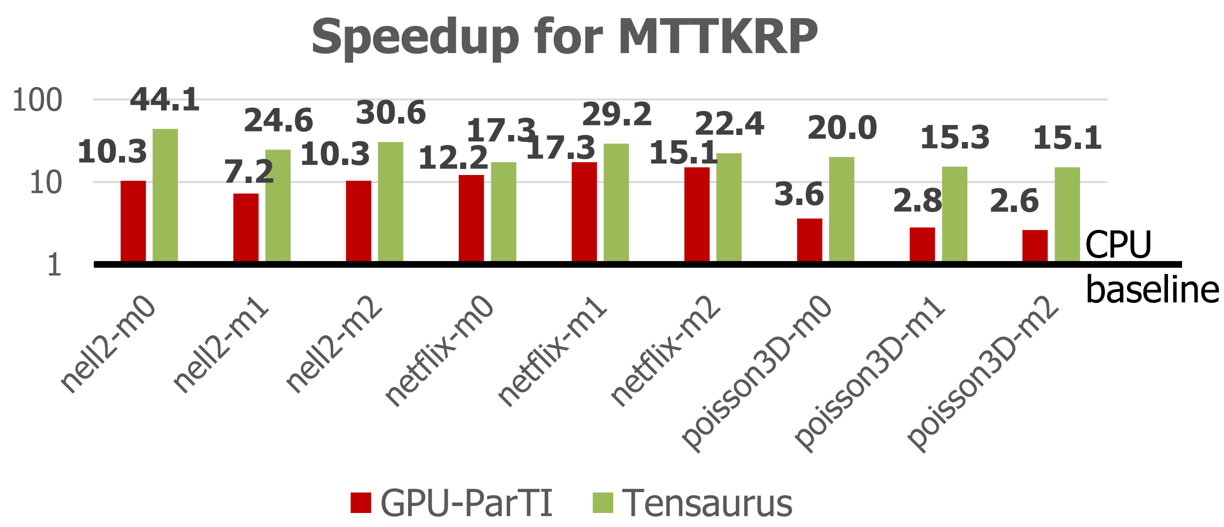
Results on Sparse Tensor Decompositions

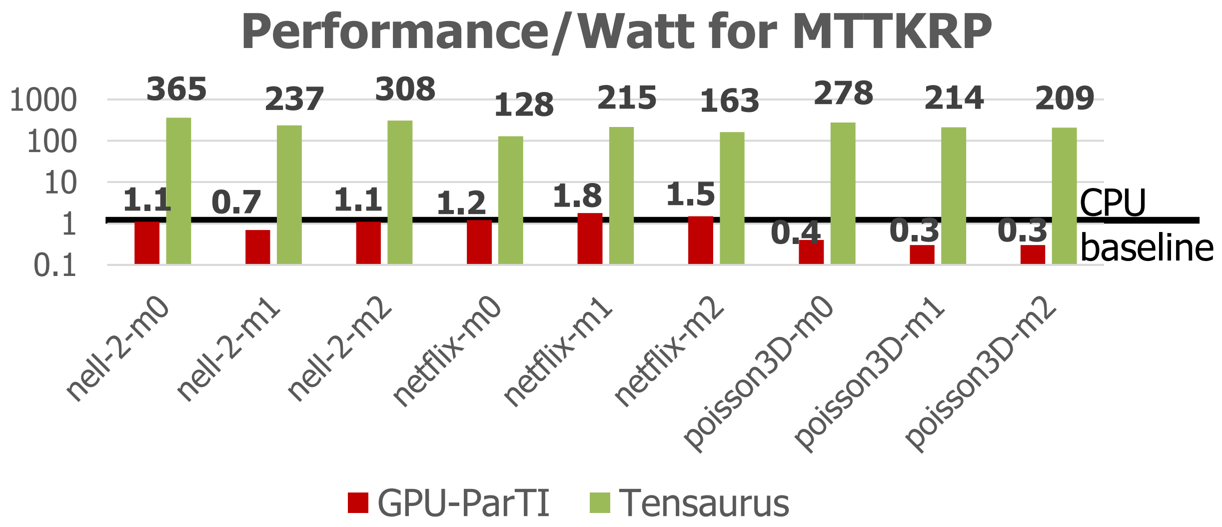
Results on Sparse Tensor Decompositions

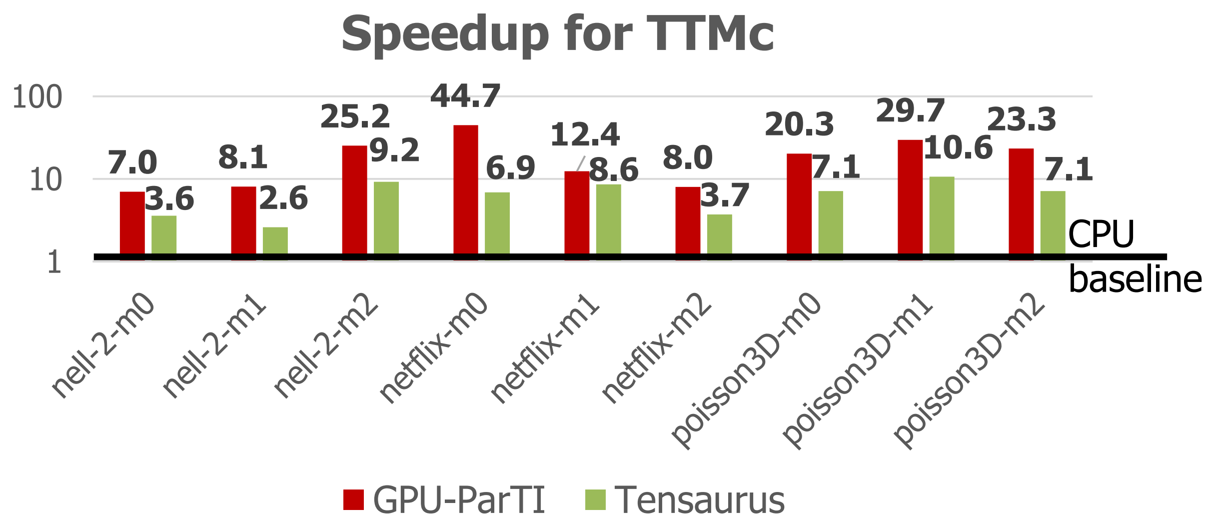


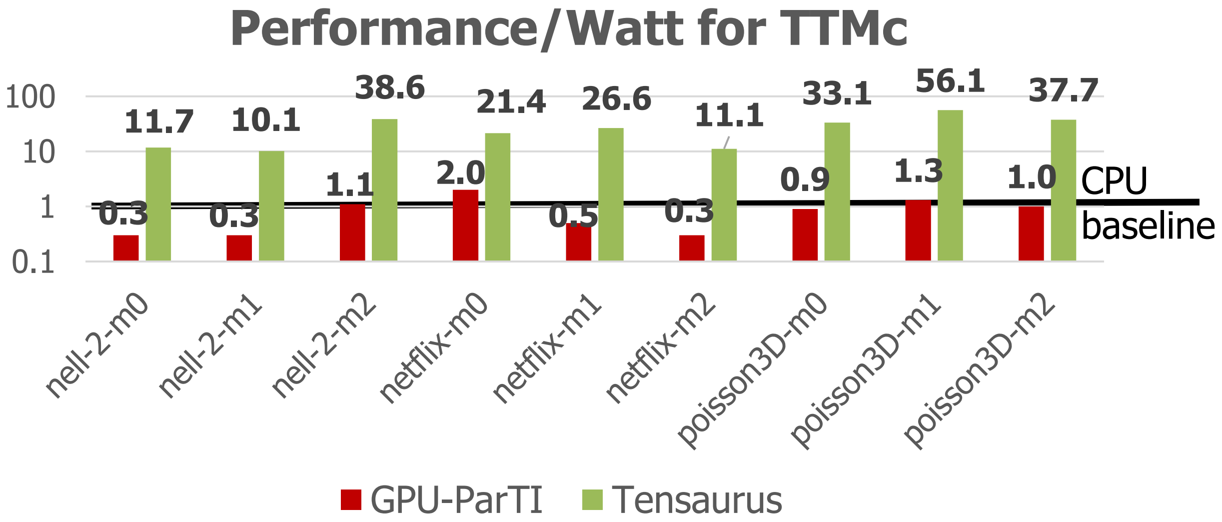


Results on Sparse Tensor Decompositions









Tensaurus is 22.9x & 3.1x faster, and 220x & 290x more energy-efficient than CPU & GPU for MTTKRP. For TTMc, it achieves 6x & 0.1x of the performance of CPU & GPU, and is 23x and 31x more energy-efficient than CPU & GPU

Chapter 5

MatRaptor : A Sparse-Sparse Matrix Multiplication Accelerator Based on Row-Wise Product

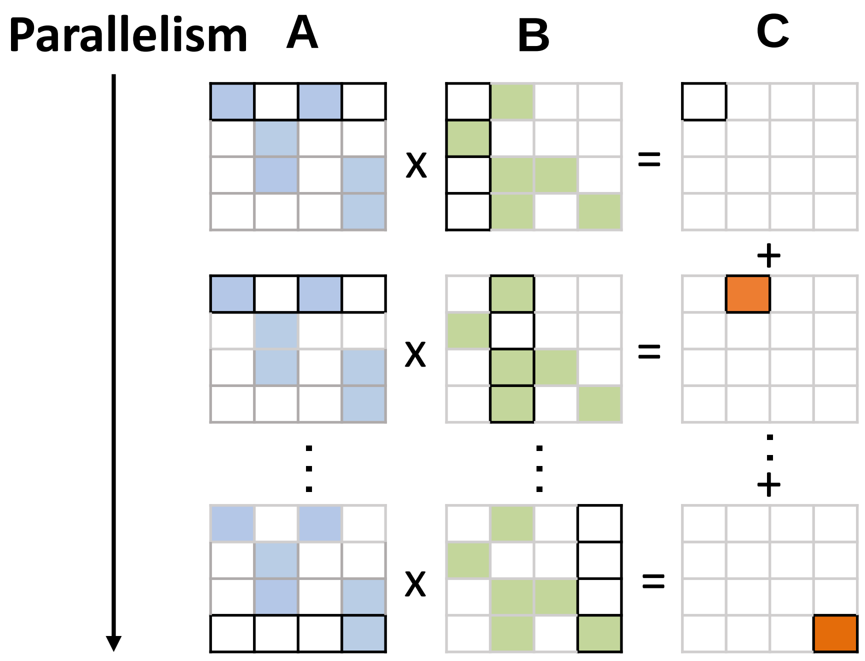
- Submitted to International Symposium on Computer Architecture (*ISCA'20*)

MatRaptor Contributions

- **MatRaptor:** An efficient hardware accelerator for sparse-sparse MM
- **Key Idea:** Co-design **algorithm** and **sparse format**
- **Applications of Sparse-Sparse MM:**
  - All Pair Shortest Path
  - Graph Searches & Contractions
  - Peer-pressure clustering, etc.

Inner & Outer Product Approaches

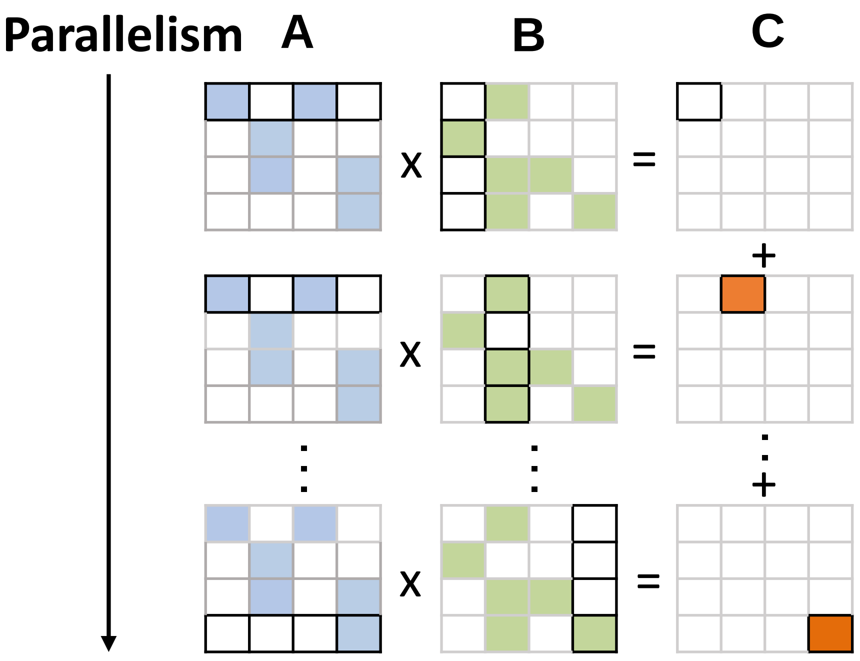
Inner & Outer Product Approaches



Inner Product

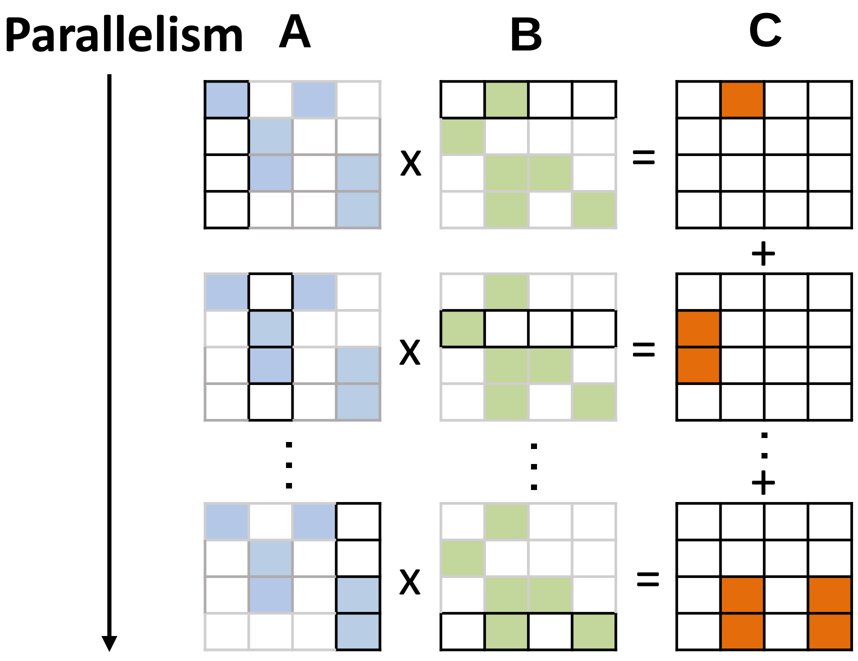
$$C[i,j] = \sum_{k} A[i,k] \, . \, B[k,j]$$

Inner & Outer Product Approaches



Inner Product

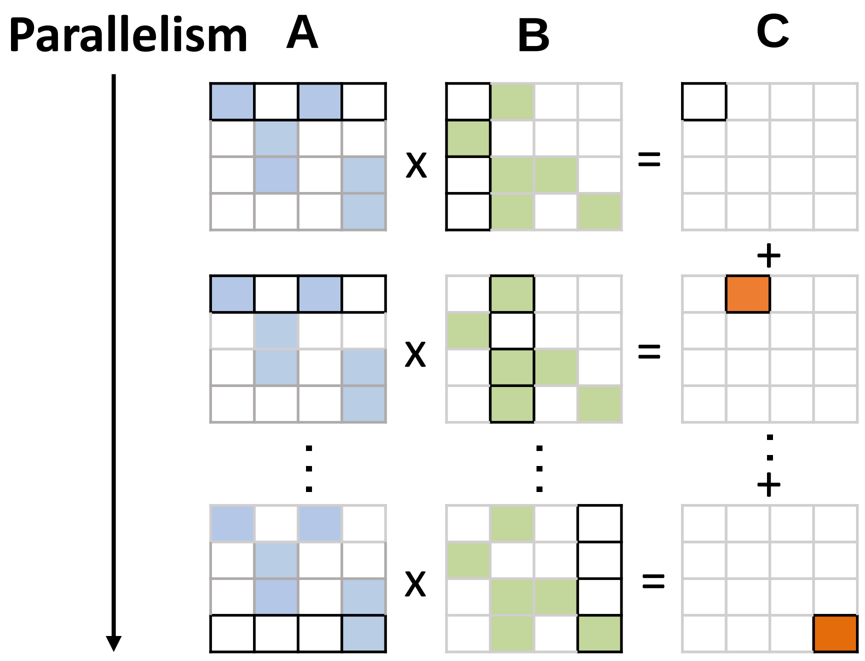
$$C[i,j] = \sum_{k} A[i,k] \,.\, B[k,j]$$



Outer Product

$$C[:,:] = \sum_{k} A[:,k] \,.\, B[k,:]$$

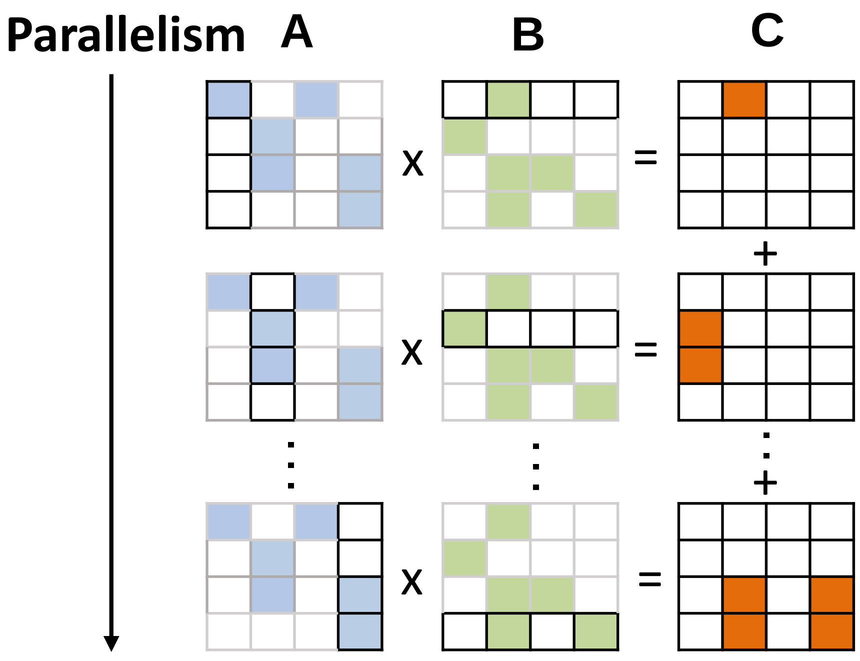
Inner & Outer Product Approaches



Inner Product

$$C[i,j] = \sum_k A[i,k]\,.\,B[k,j]$$

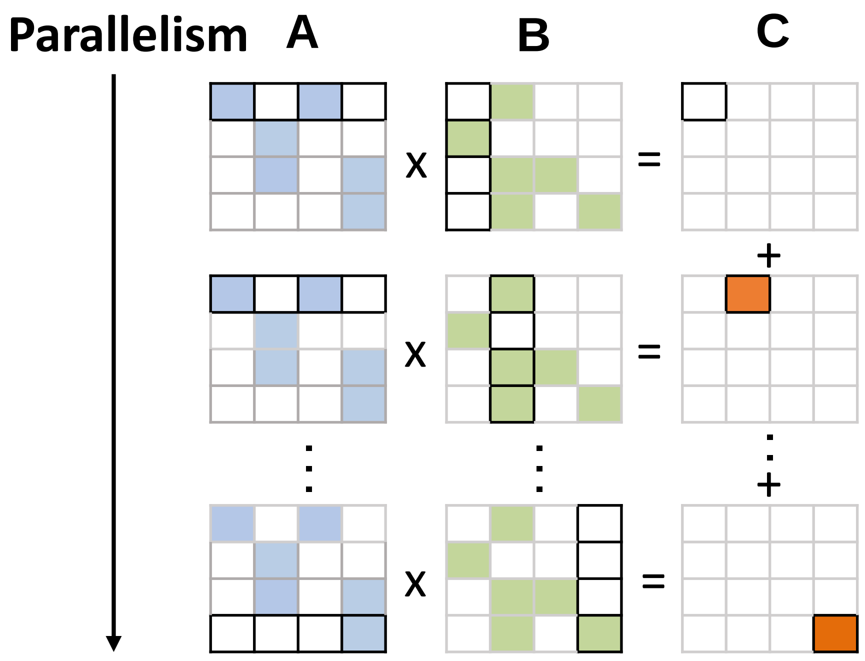
Inconsistent formats
Inefficient index matching
No synchronization
Low on-chip memory



Outer Product

$$C[:,:] = \sum_k A[:,k]\,.\,B[k,:]$$

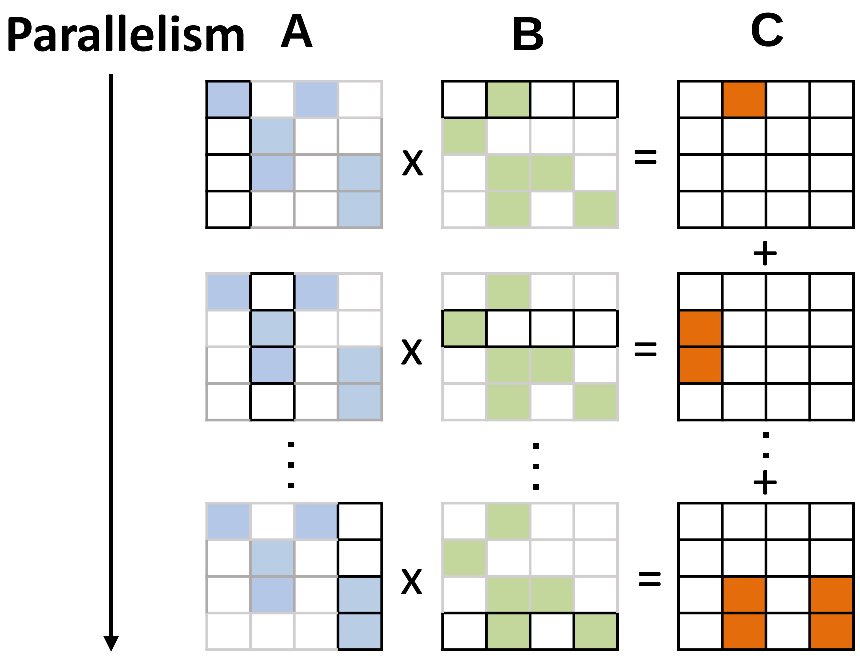
Inner & Outer Product Approaches



Inner Product

$$C[i,j] = \sum_k A[i,k]\,.\,B[k,j]$$

Inconsistent formats
Inefficient index matching
No synchronization
Low on-chip memory

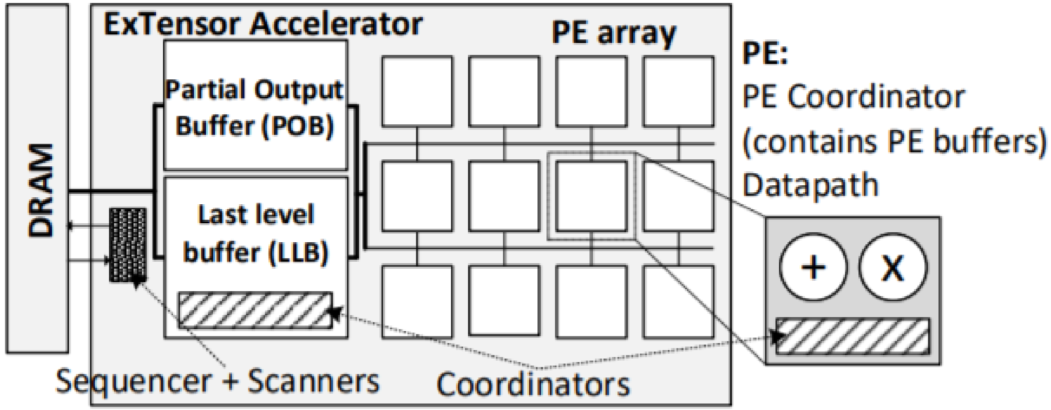


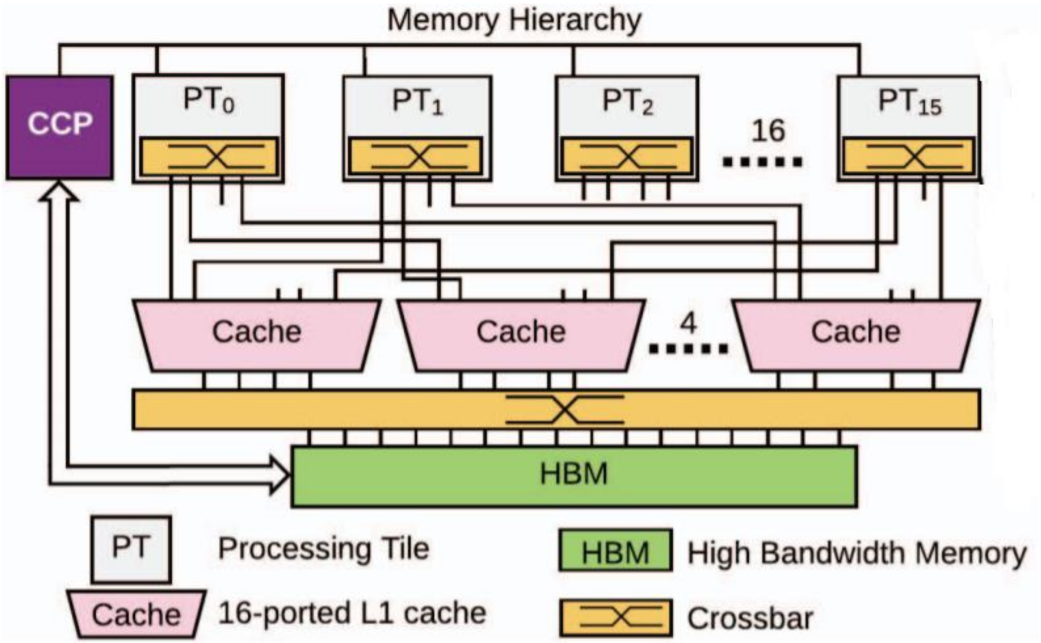
Outer Product

$$C[:,:] = \sum_k A[:,k]\,.\,B[k,:]$$

Inconsistent formats
Elimination of index matching
Synchronization
High on-chip memory

Inner & Outer Product Approaches





Inner Product – ExTensor[1]

Outer Product – OuterSPACE [2]

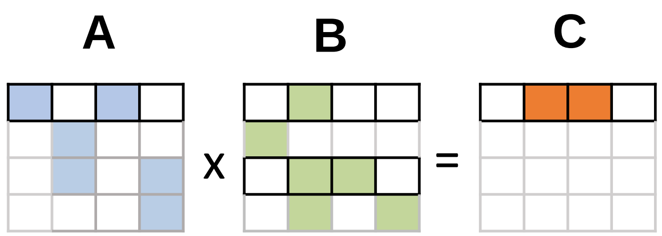
CSR & CSC formats
Skip mechanism

CR and CC formats
Locks
Caches & deals with evictions

[1] Hegde, Kartik, et al. "ExTensor: An Accelerator for Sparse Tensor Algebra.“, *Int'l Symp. on Microarchitecture (MICRO)*. 2019.
[2] Pal, Subhankar, et al. "Outerspace: An outer product based sparse matrix multiplication accelerator.“, *Int'l Symp. on High Performance Computer Architecture (HPCA)*. IEEE, 2018.

Row-wise Product Approach for Sparse MM computation

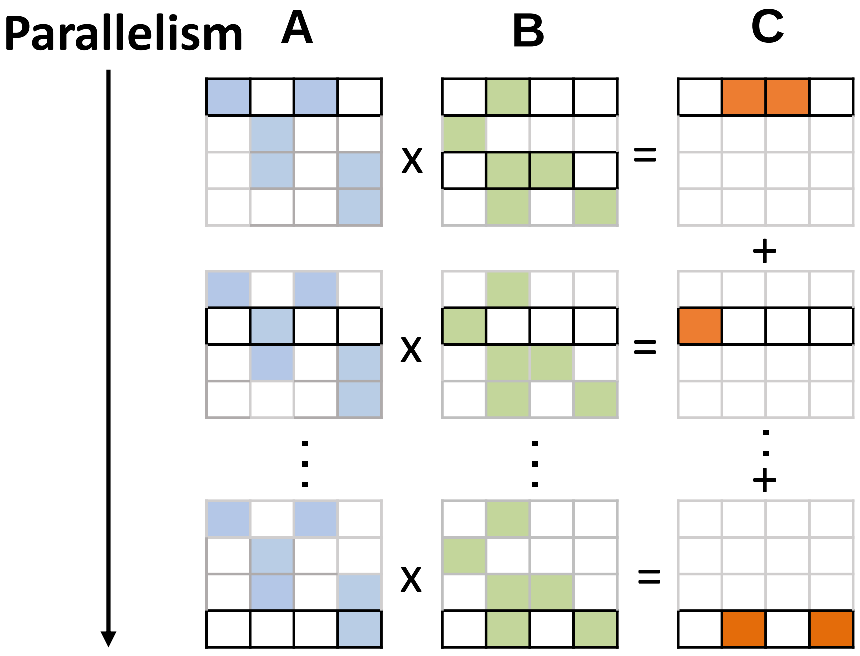
Row-wise Product Approach for Sparse MM computation



Row-wise Product

$$C[i,:] = \sum_{k} A[i,k] \, . \, B[k,:]$$

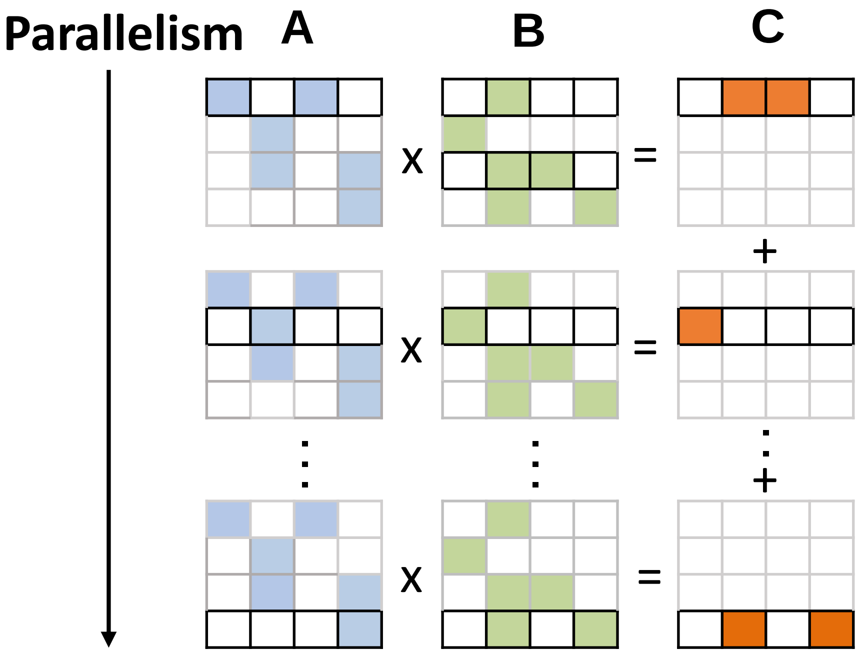
Row-wise Product Approach for Sparse MM computation



Row-wise Product

$$C[i,:] = \sum_{k} A[i,k] \,.\, B[k,:]$$

Row-wise Product Approach for Sparse MM computation

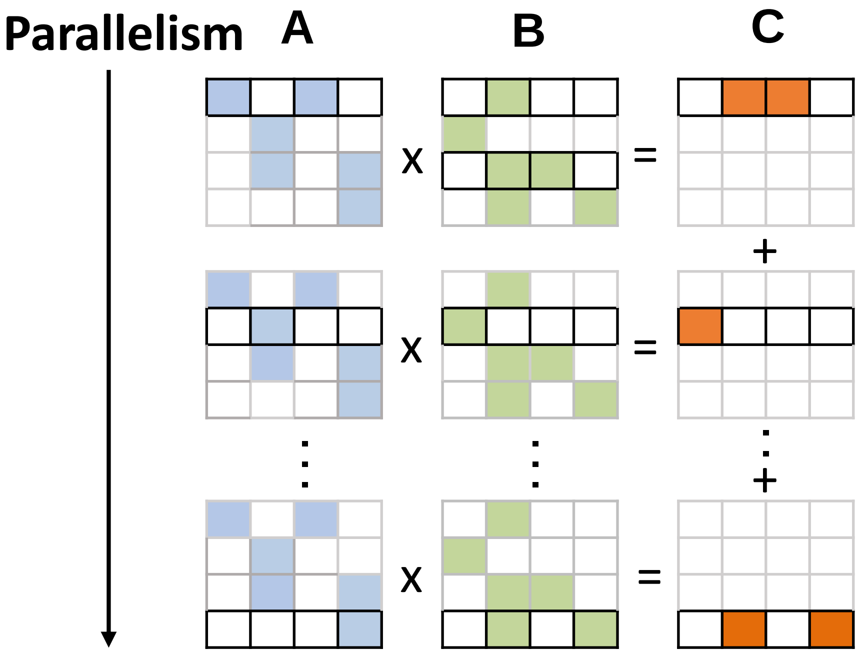


Row-wise Product

$$C[i,:] = \sum_{k} A[i,k] \,.\, B[k,:]$$

Benefits over other approaches due to:

Row-wise Product Approach for Sparse MM computation



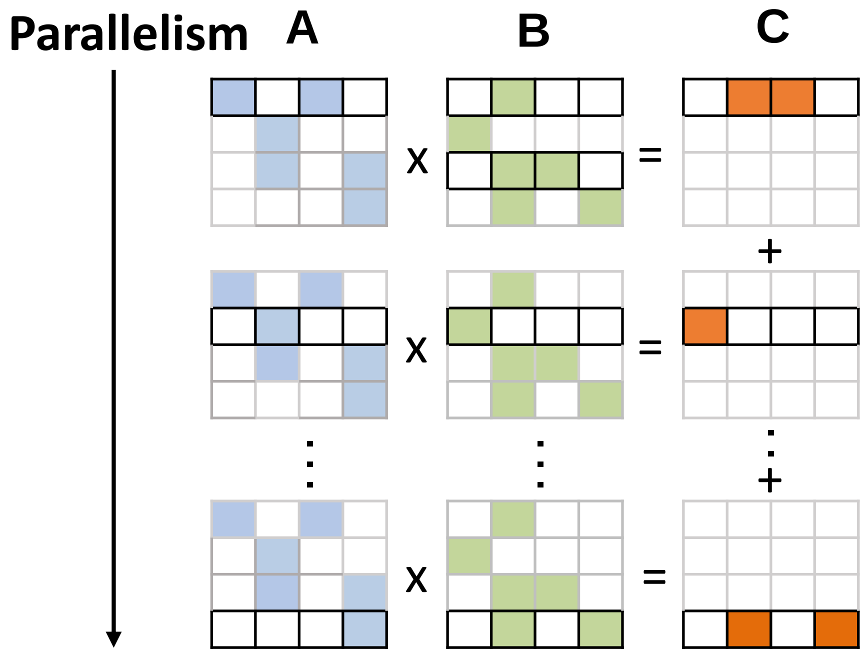
Row-wise Product

$$C[i,:] = \sum_{k} A[i,k] \, . \, B[k,:]$$

Benefits over other approaches due to:

- **Consistent formatting**
  - Row-major format for both inputs and output

Row-wise Product Approach for Sparse MM computation



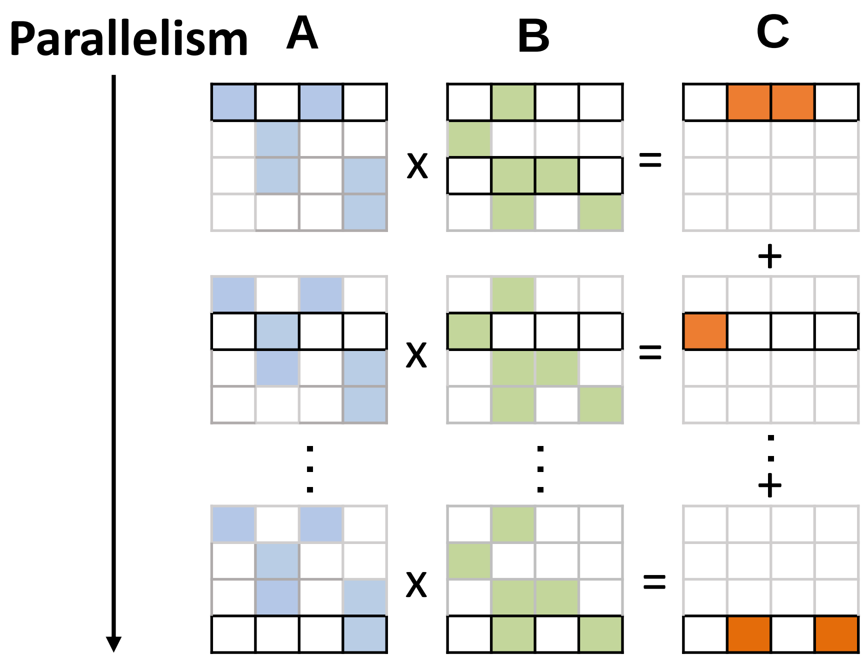
Row-wise Product

$$C[i,:] = \sum_{k} A[i,k] \,.\, B[k,:]$$

Benefits over other approaches due to:

- **Consistent formatting**
  - Row-major format for both inputs and output
- **Elimination of index matching**
  - No index matching comparisons as in inner-product

Row-wise Product Approach for Sparse MM computation



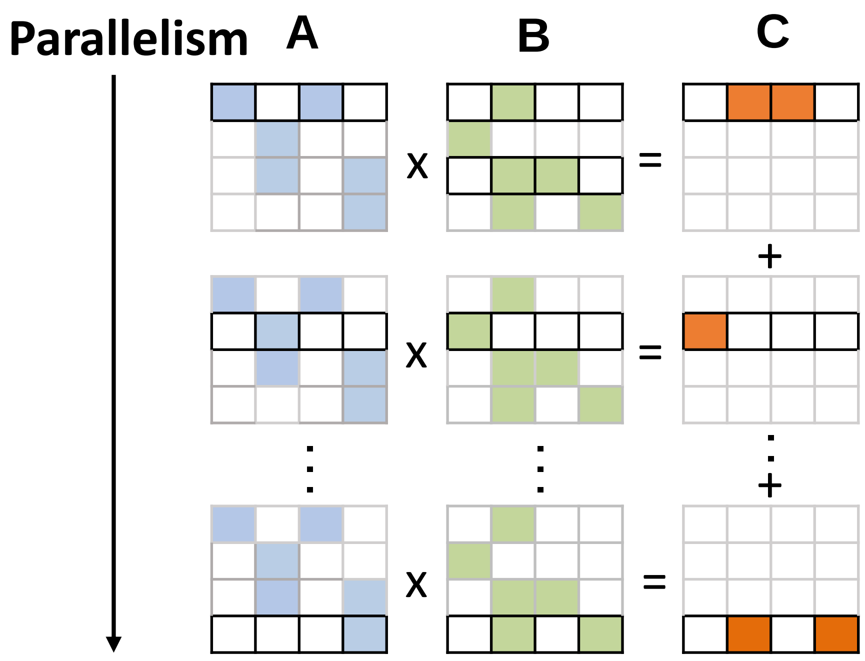
Row-wise Product

$$C[i,:] = \sum_{k} A[i,k] \, . \, B[k,:]$$

Benefits over other approaches due to:

- **Consistent formatting**
  - Row-major format for both inputs and output
- **Elimination of index matching**
  - No index matching comparisons as in inner-product
- **Elimination of synchronization**
  - No RAW dependencies → No synchronization

Row-wise Product Approach for Sparse MM computation



Row-wise Product

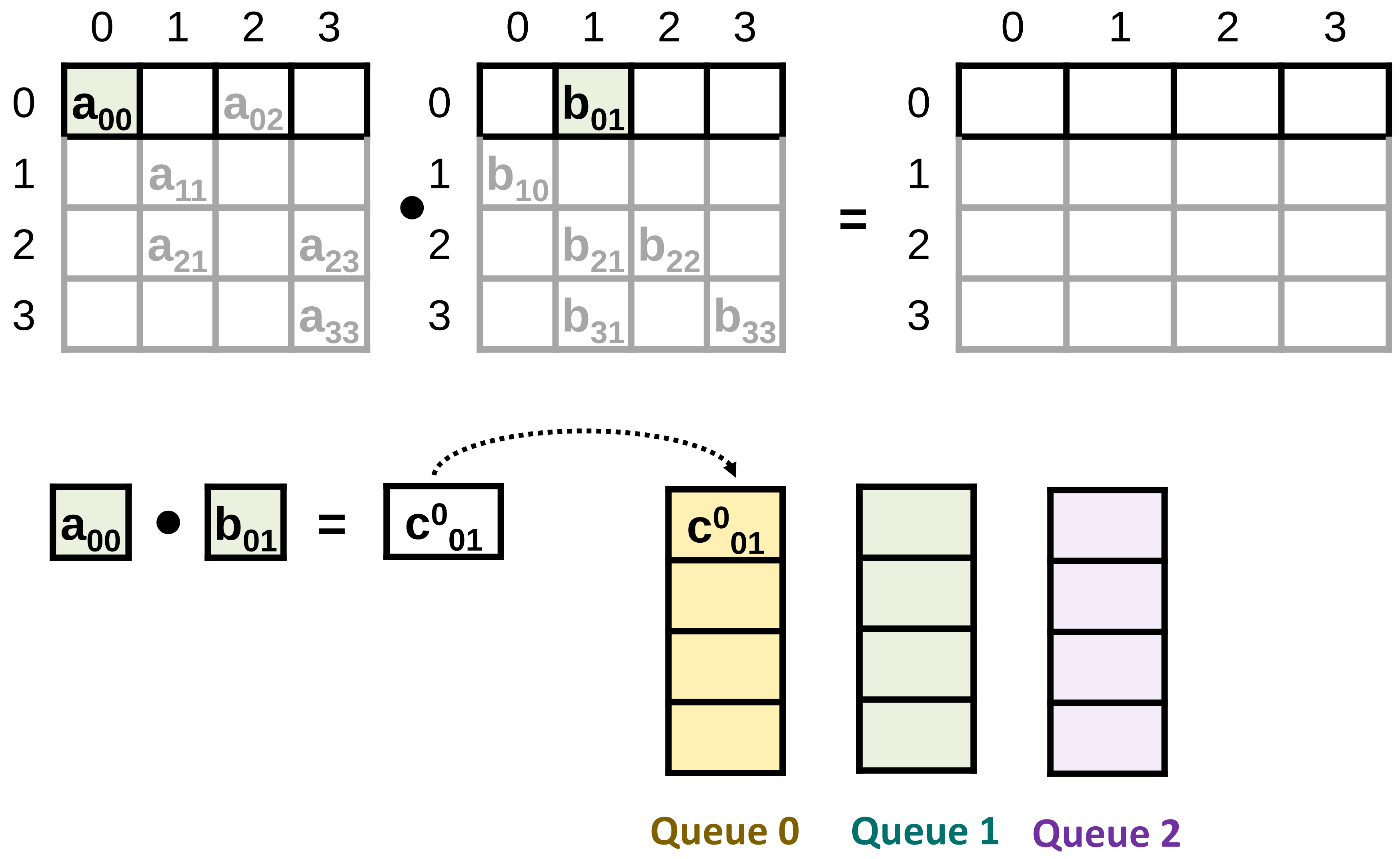
$$C[i,:] = \sum_k A[i,k]\,.\,B[k,:]$$

Benefits over other approaches due to:

- **Consistent formatting**
  - Row-major format for both inputs and output
- **Elimination of index matching**
  - No index matching comparisons as in inner-product
- **Elimination of synchronization**
  - No RAW dependencies → No synchronization
- **Low on-chip memory requirements**
  - Only stores a single row of output matrix on chip

Row-wise Product Implementation

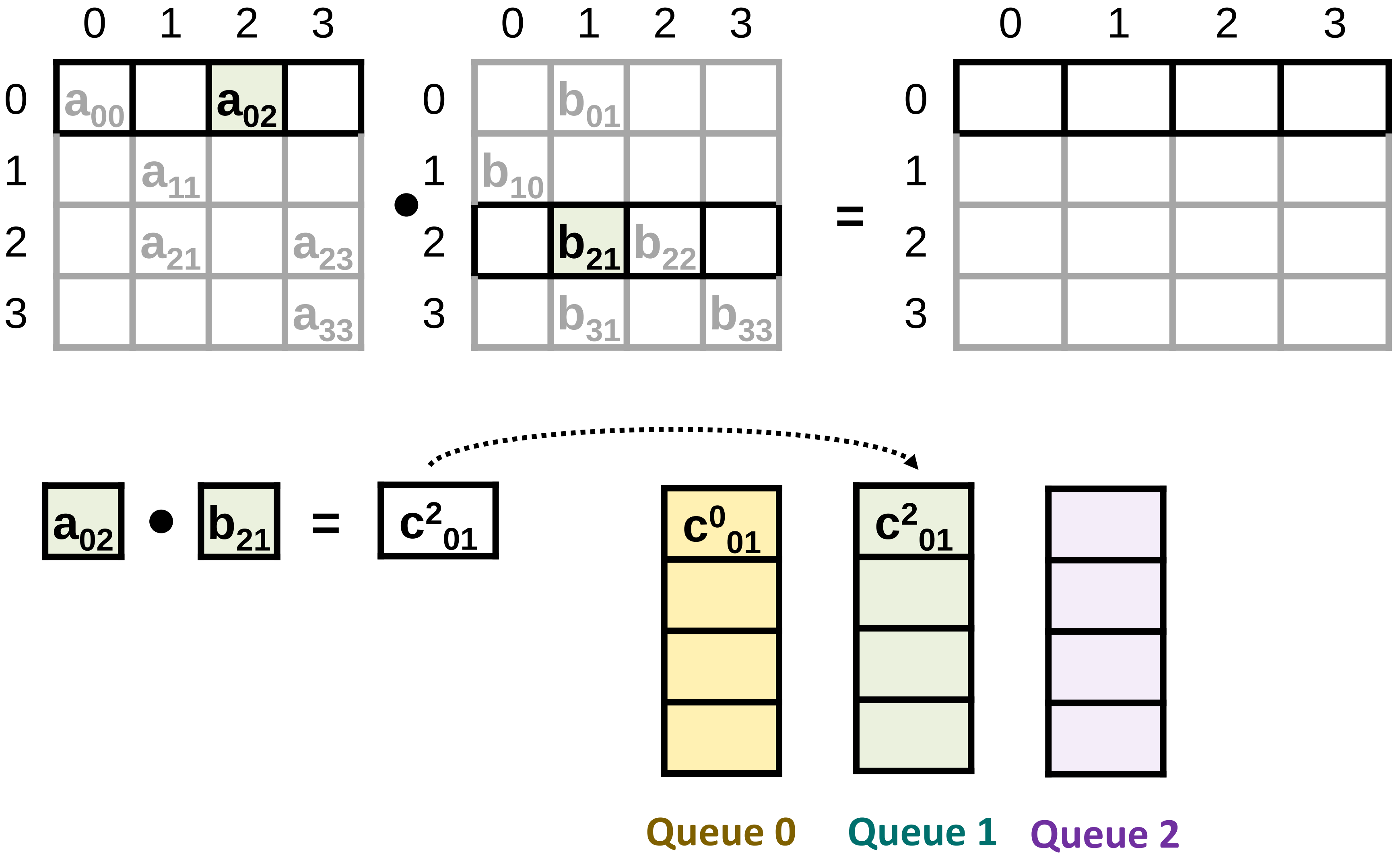
$$C[i,:] = \sum_k A[i,k]\,.\,B[k,:]$$



Notation: $a_{ik}\,.\,b_{kj} = c^k_{ij}$

Row-wise Product Implementation

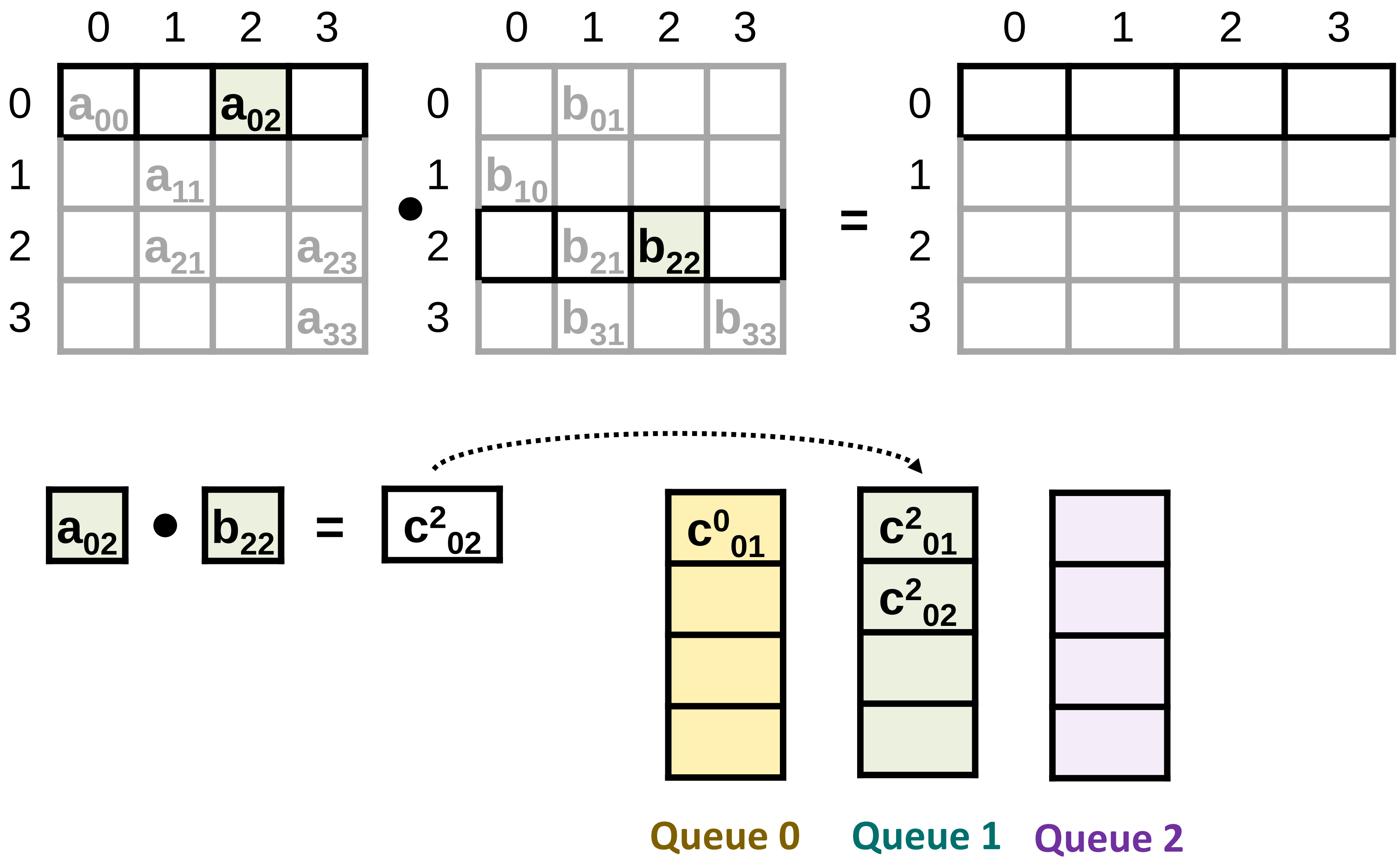
$$C[i,:] = \sum_k A[i,k] \, . \, B[k,:]$$



Notation: $a_{ik} \, . \, b_{kj} = c^k_{ij}$

Row-wise Product Implementation

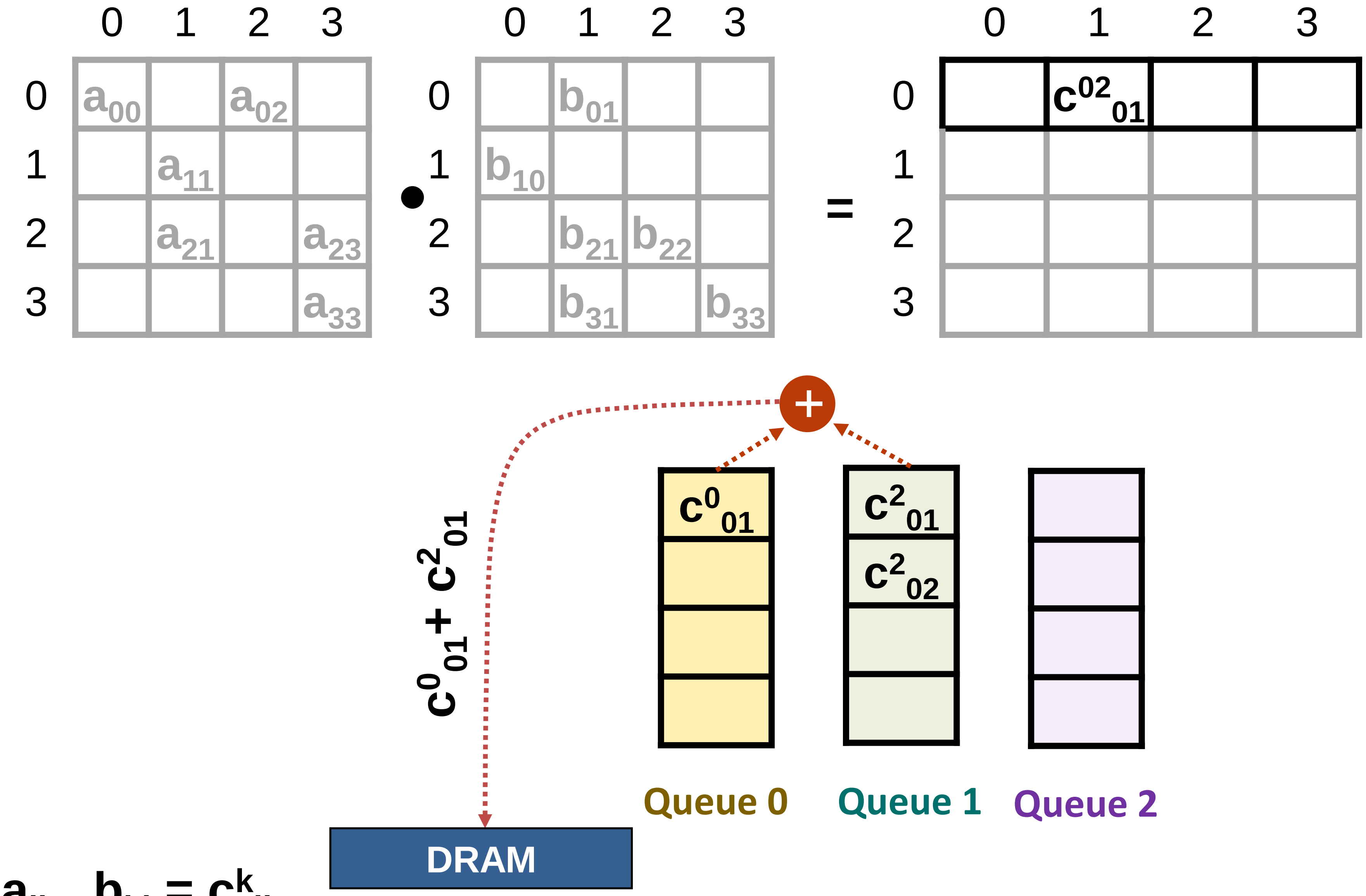
$$C[i,:] = \sum_k A[i,k] \,.\, B[k,:]$$



Notation: $a_{ik} \,.\, b_{kj} = c^k_{ij}$

Row-wise Product Implementation

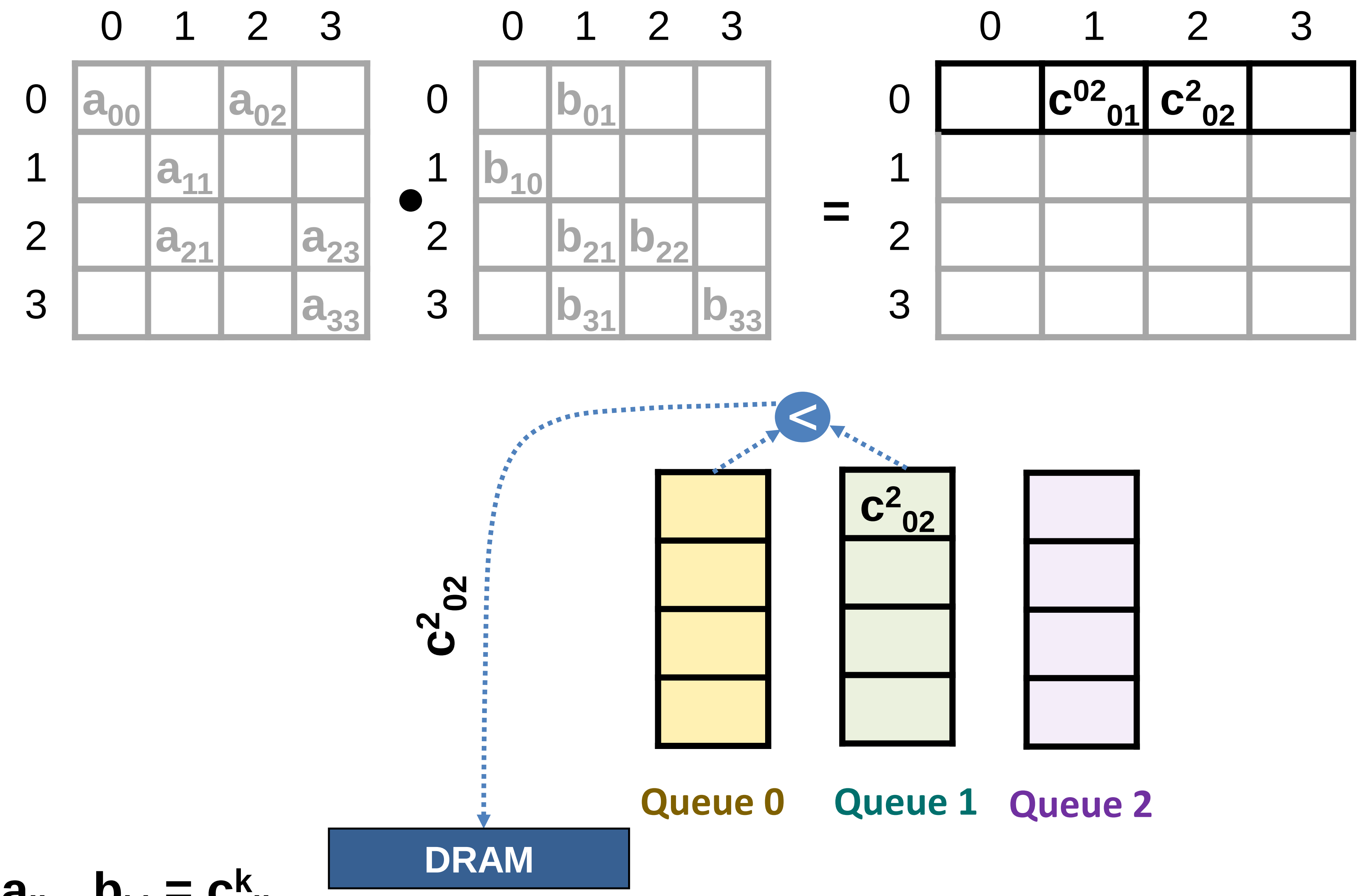
$$C[i,:] = \sum_k A[i,k] \, . \, B[k,:]$$



Notation: $a_{ik} \, . \, b_{kj} = c^k_{ij}$

Row-wise Product Implementation

$$C[i,:] = \sum_k A[i,k] \, . \, B[k,:]$$



Notation: $a_{ik} \, . \, b_{kj} = c^{k}_{ij}$

Requirements for Sparse Format for Sparse-Sparse MM

- **Output format needs to be consistent with input format**
  - Many applications use a chain of sparse-sparse MMs
  - CISR requires global scheduling for load balancing → cannot be used for output format

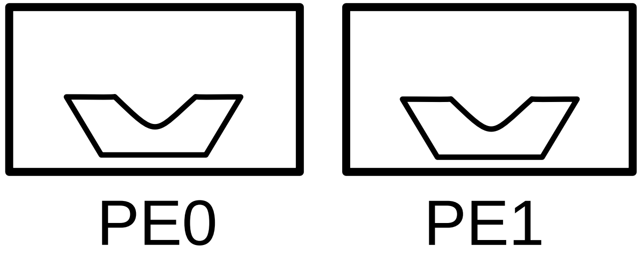
- **The same format should result in streaming memory accesses for both A & B**
  - CISR is efficient for the case when a row of sparse matrix is accessed by only one PE
  - In sparse-sparse MM, B matrix is accessed based on indices of non-zeros in A matrix

CISR for Sparse-Sparse MM

	0	1	2	3
0	a_{00}		a_{02}	
1		a_{11}		
2		a_{21}		a_{23}
3				a_{33}

●

	0	1	2	3
0		b_{01}		
1	b_{10}			
2		b_{21}	b_{22}	
3		b_{31}		b_{33}



CISR

PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

	0	1	0	1	0	1
Row Len.	1	1	2	2		
Col. Id	1	0	1	1	2	
Values	b_{01}	b_{10}	b_{21}	b_{31}	b_{22}	b_{33}

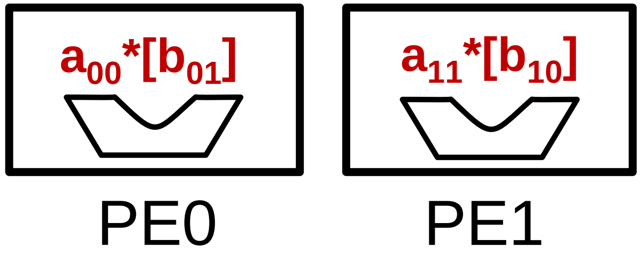
CISR for Sparse-Sparse MM

	0	1	2	3
0	a_{00}		a_{02}	
1		a_{11}		
2		a_{21}		a_{23}
3				a_{33}

●

	0	1	2	3
0		b_{01}		
1	b_{10}			
2		b_{21}	b_{22}	
3		b_{31}		b_{33}

— Cycle 0

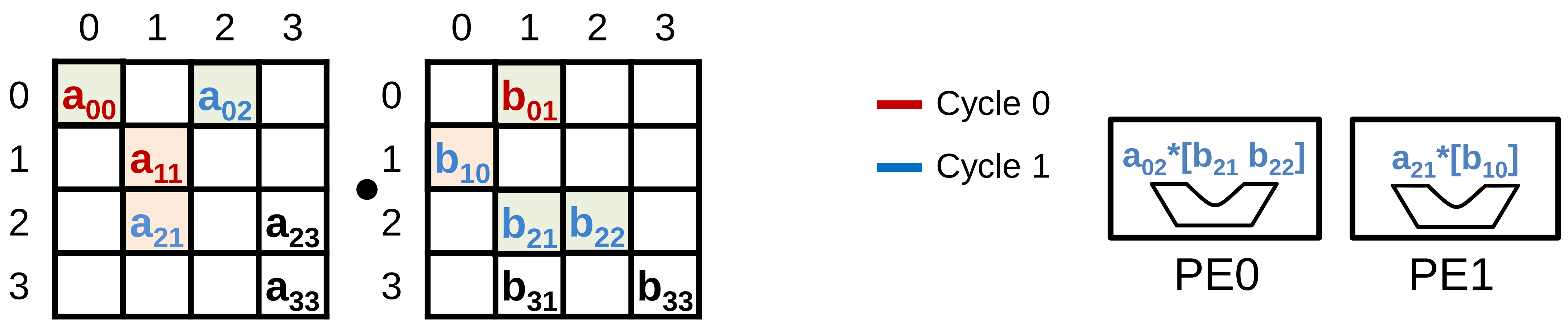


CISR

PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

	0	1	0	1	0	1
Row Len.	1	1	2	2		
Col. Id	1	0	1	1	2	
Values	b_{01}	b_{10}	b_{21}	b_{31}	b_{22}	b_{33}

CISR for Sparse-Sparse MM

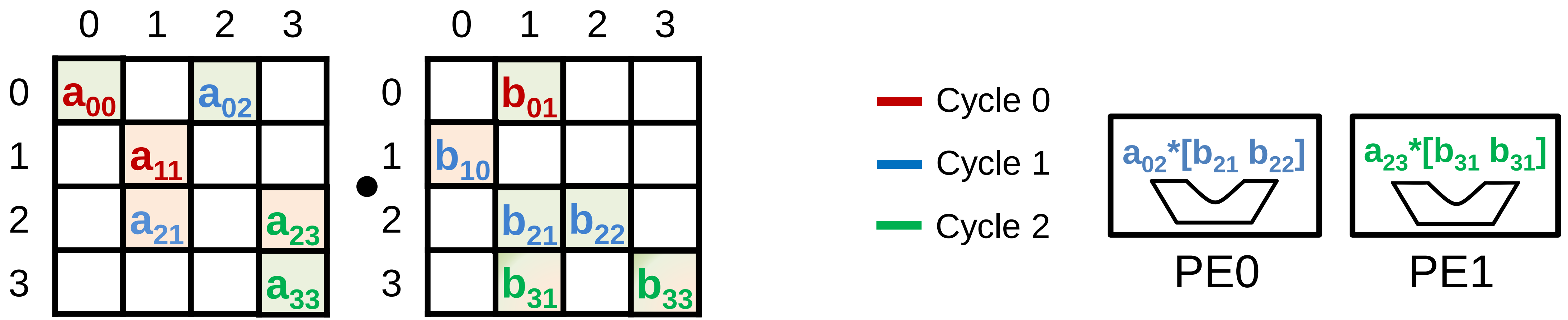


CISR

PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

	0	1	0	1	0	1
Row Len.	1	1	2	2		
Col. Id	1	0	1	1	2	
Values	b_{01}	b_{10}	b_{21}	b_{31}	b_{22}	b_{33}

CISR for Sparse-Sparse MM



CISR

PE	0	1	0	1	0	1
Row Len.	**2**	**1**	**1**	**2**		
Col. Id	**0**	**1**	**2**	**1**	**3**	**3**
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

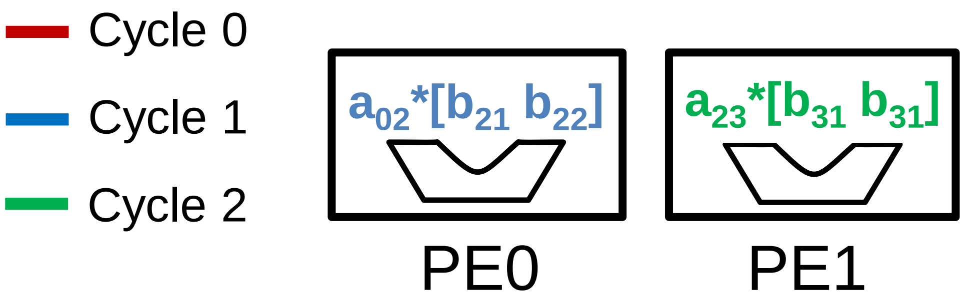
	0	1	0	1	0	1
Row Len.	**1**	**1**	**2**	**2**		
Col. Id	**1**	**0**	**1**	**1**	**2**	
Values	b_{01}	b_{10}	b_{21}	b_{31}	b_{22}	b_{33}

CISR for Sparse-Sparse MM

	0	1	2	3
0	a_{00}		a_{02}	
1		a_{11}		
2		a_{21}		a_{23}
3				a_{33}

●

	0	1	2	3
0		b_{01}		
1	b_{10}			
2		b_{21}	b_{22}	
3		b_{31}		b_{33}



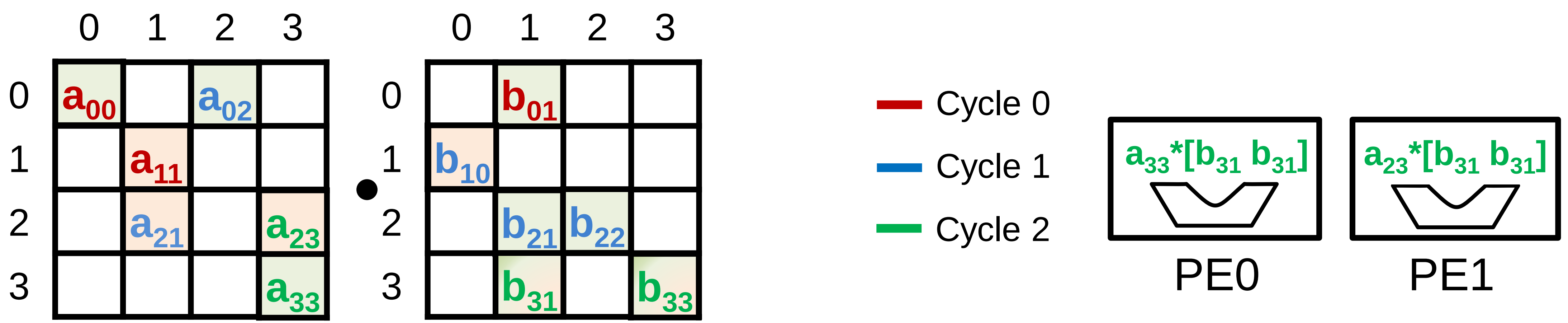
CISR

PE	0	1	0	1	0	1
Row Len.	2	1	1	2		
Col. Id	0	1	2	1	3	3
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

	0	1	0	1	0	1
Row Len.	1	1	2	2		
Col. Id	1	0	1	1	2	
Values	b_{01}	b_{10}	b_{21}	b_{31}	b_{22}	b_{33}

Streaming and vectorized accesses

CISR for Sparse-Sparse MM



CISR

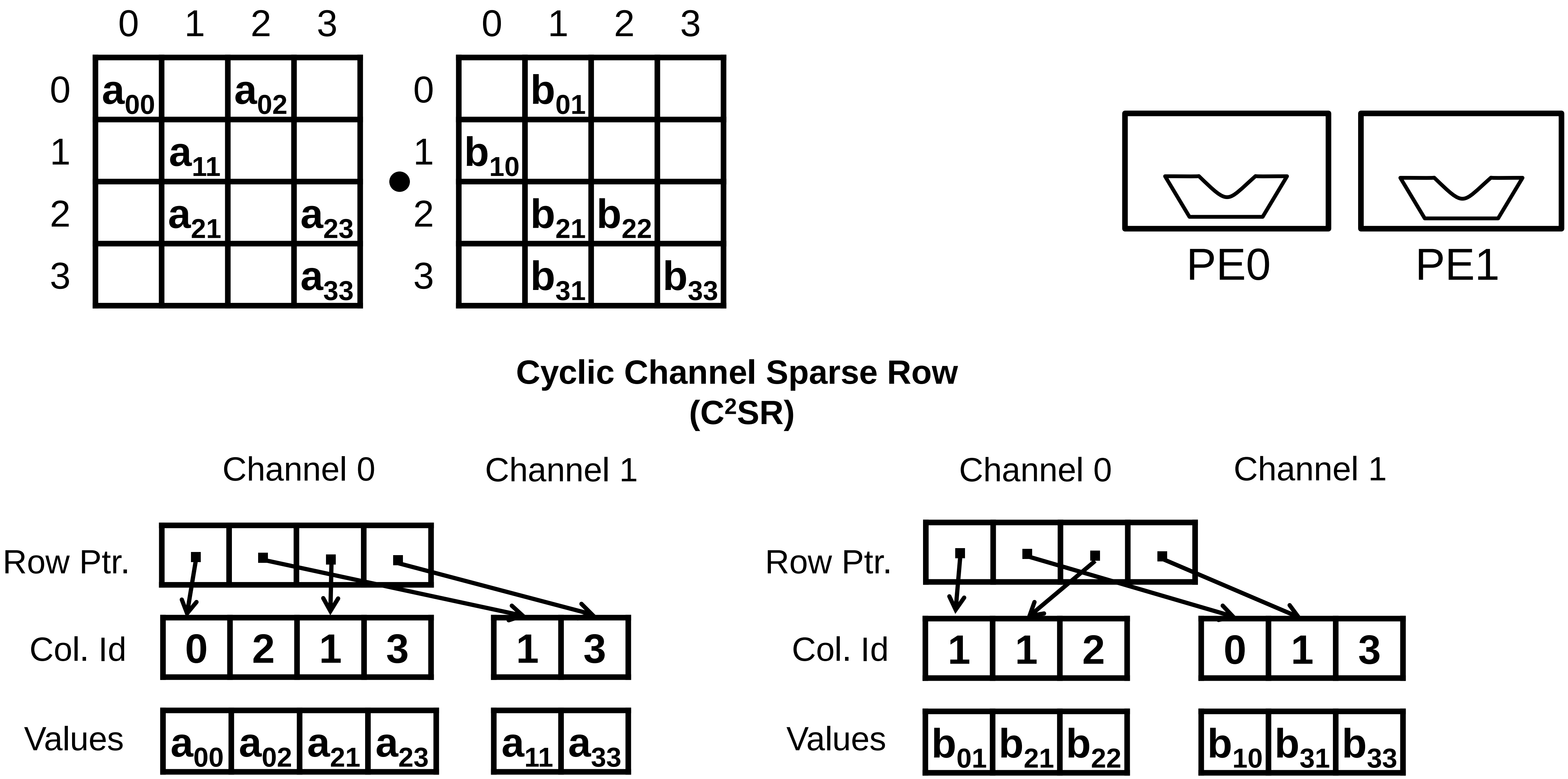
PE	0	1	0	1	0	1
Row Len.	**2**	**1**	**1**	**2**		
Col. Id	**0**	**1**	**2**	**1**	**3**	**3**
Values	a_{00}	a_{11}	a_{02}	a_{21}	a_{33}	a_{23}

Streaming and vectorized accesses

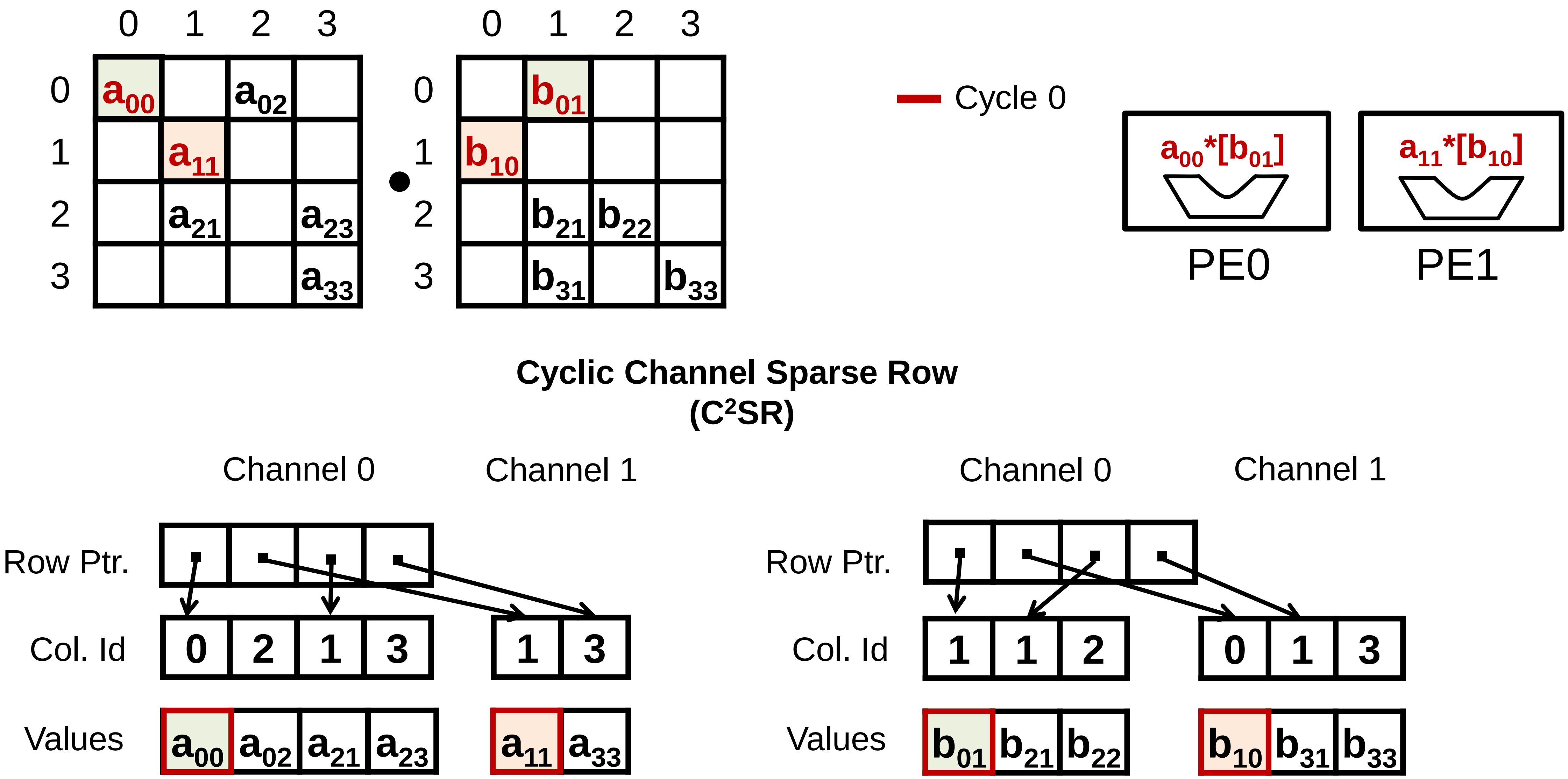
	0	1	0	1	0	1
Row Len.	**1**	**1**	**2**	**2**		
Col. Id	**1**	**0**	**1**	**1**	**2**	
Values	b_{01}	b_{10}	b_{21}	b_{31}	b_{22}	b_{33}

Non-streaming & non-vectorized accesses

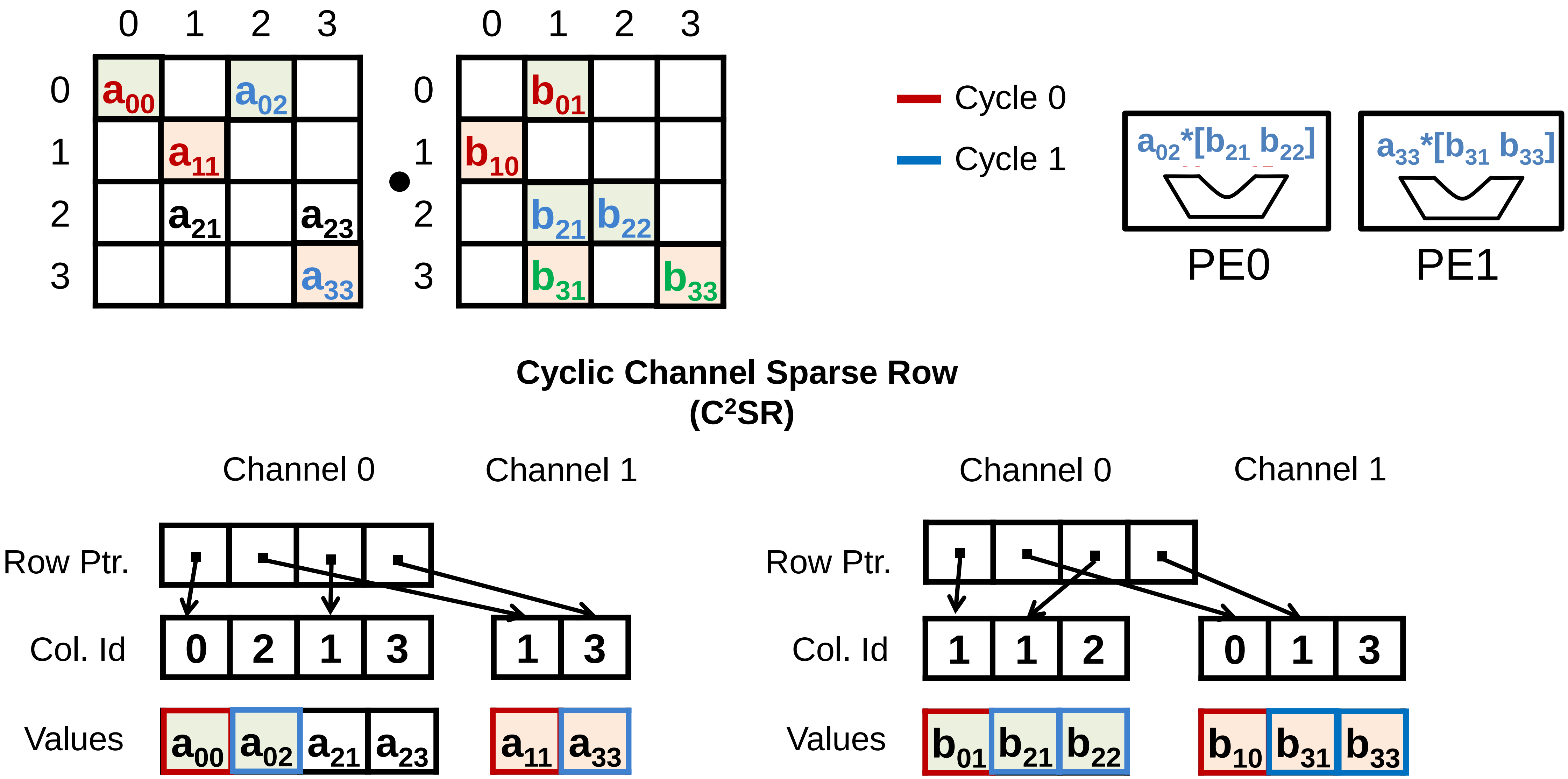
C^2SR for Sparse-Sparse MM



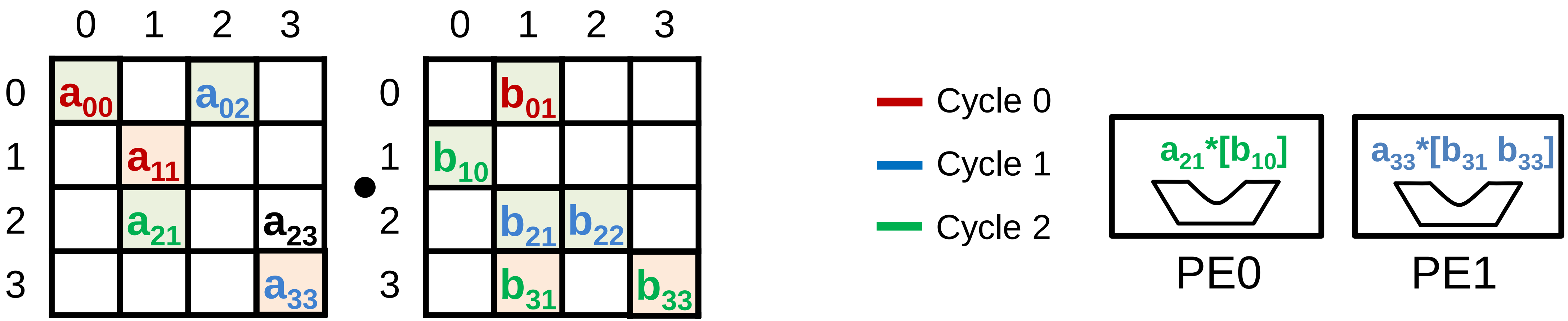
C²SR for Sparse-Sparse MM



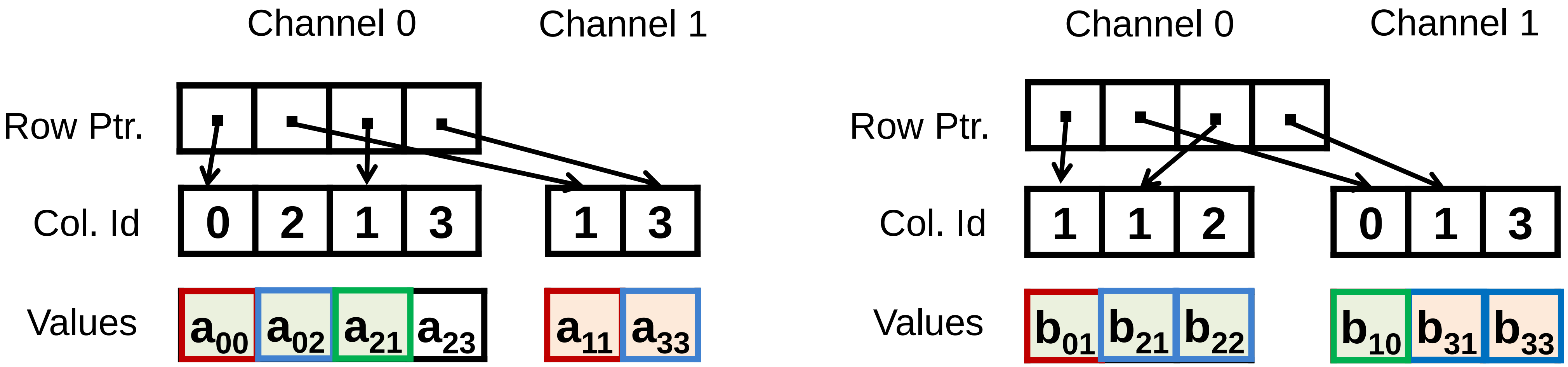
C^2SR for Sparse-Sparse MM



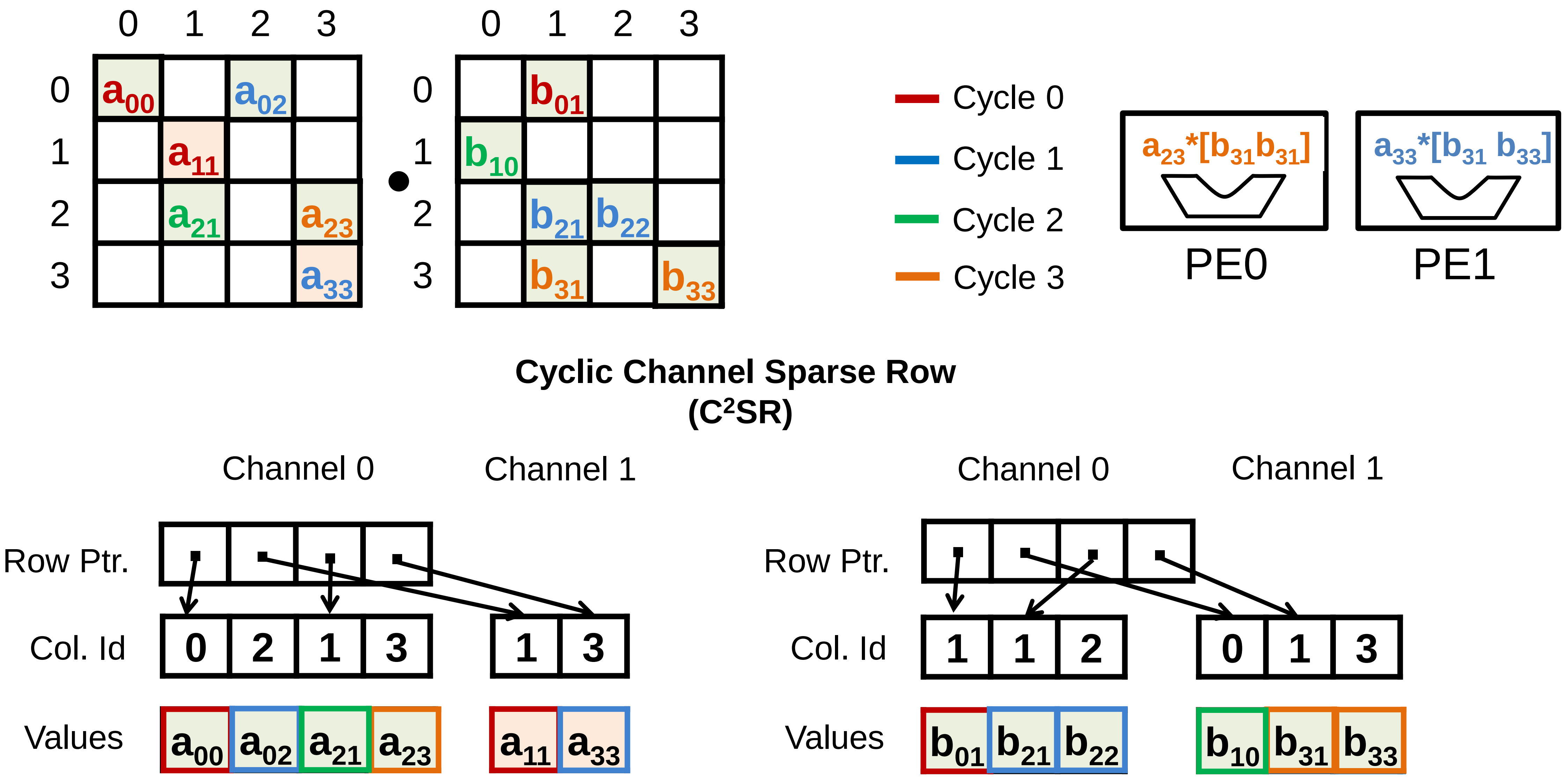
C^2SR for Sparse-Sparse MM



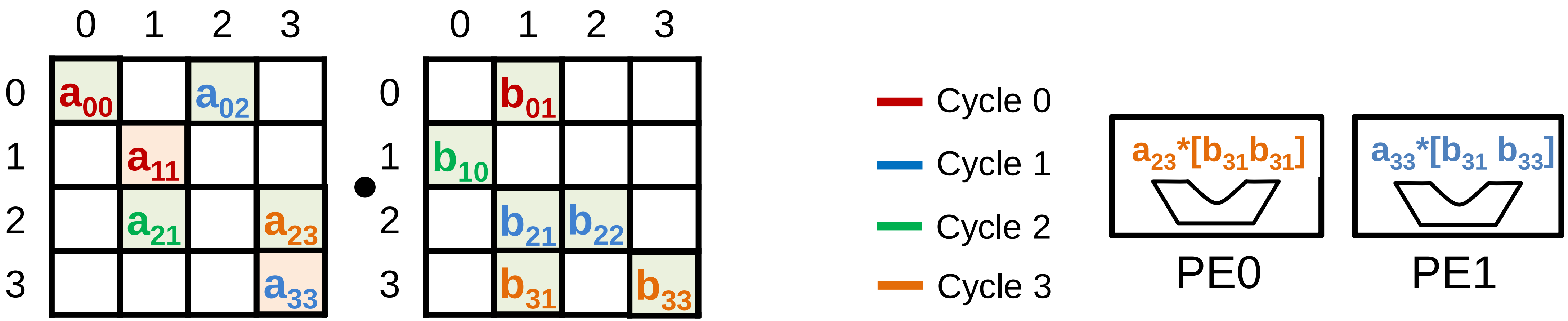
Cyclic Channel Sparse Row
(C^2SR)



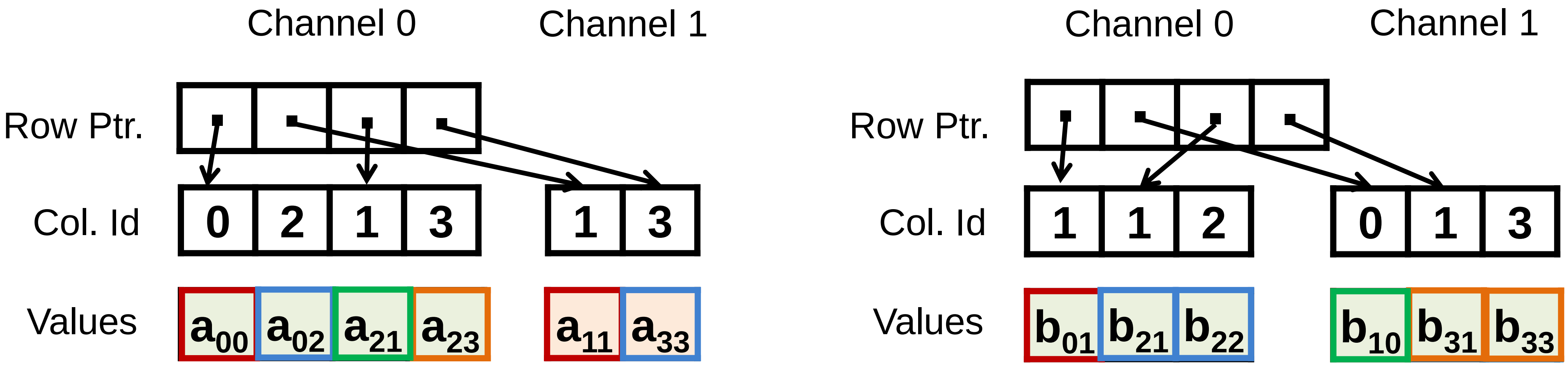
C^2SR for Sparse-Sparse MM



C^{2}SR for Sparse-Sparse MM

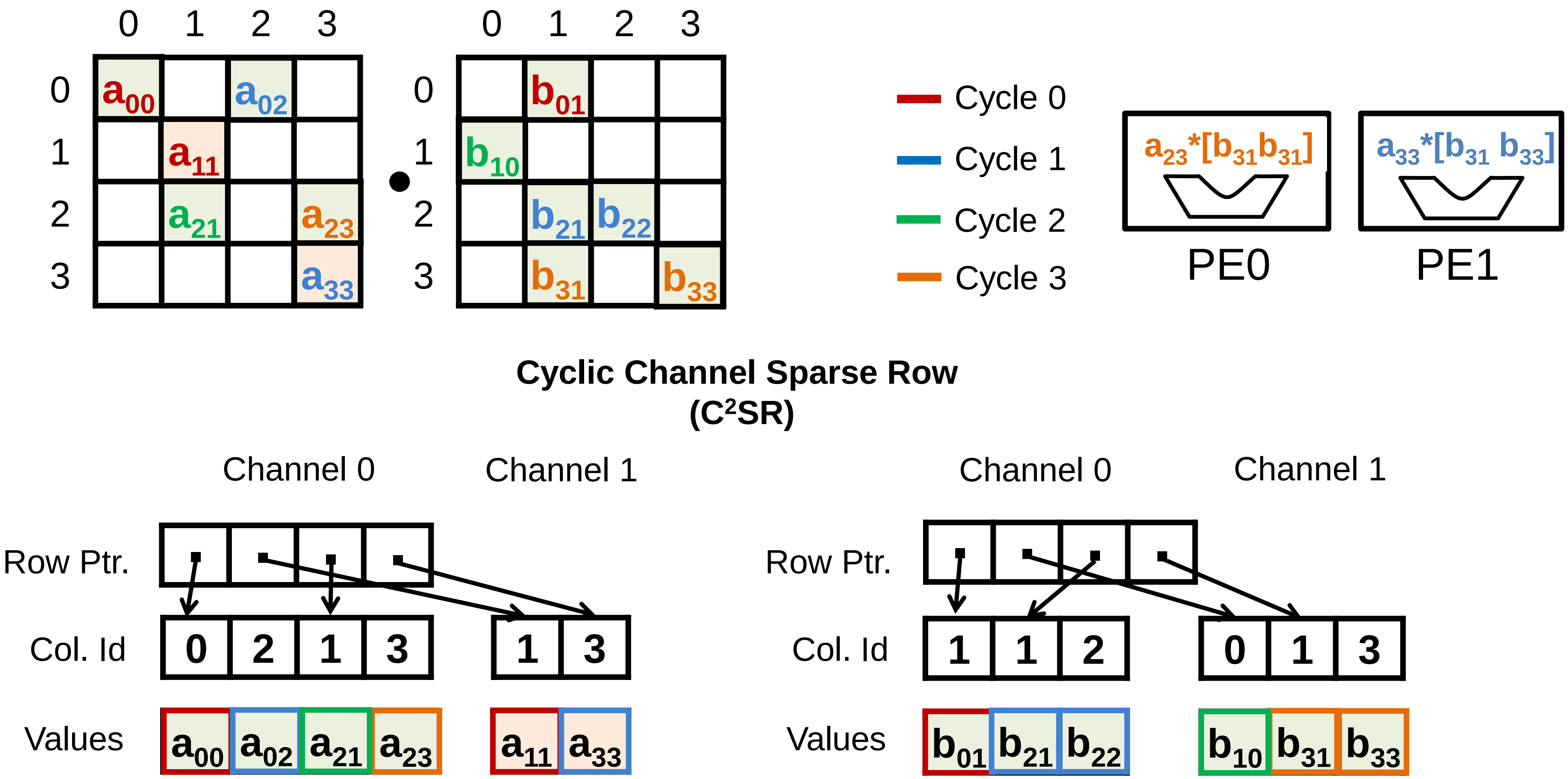


Cyclic Channel Sparse Row (C^{2}SR)



Streaming and vectorized accesses

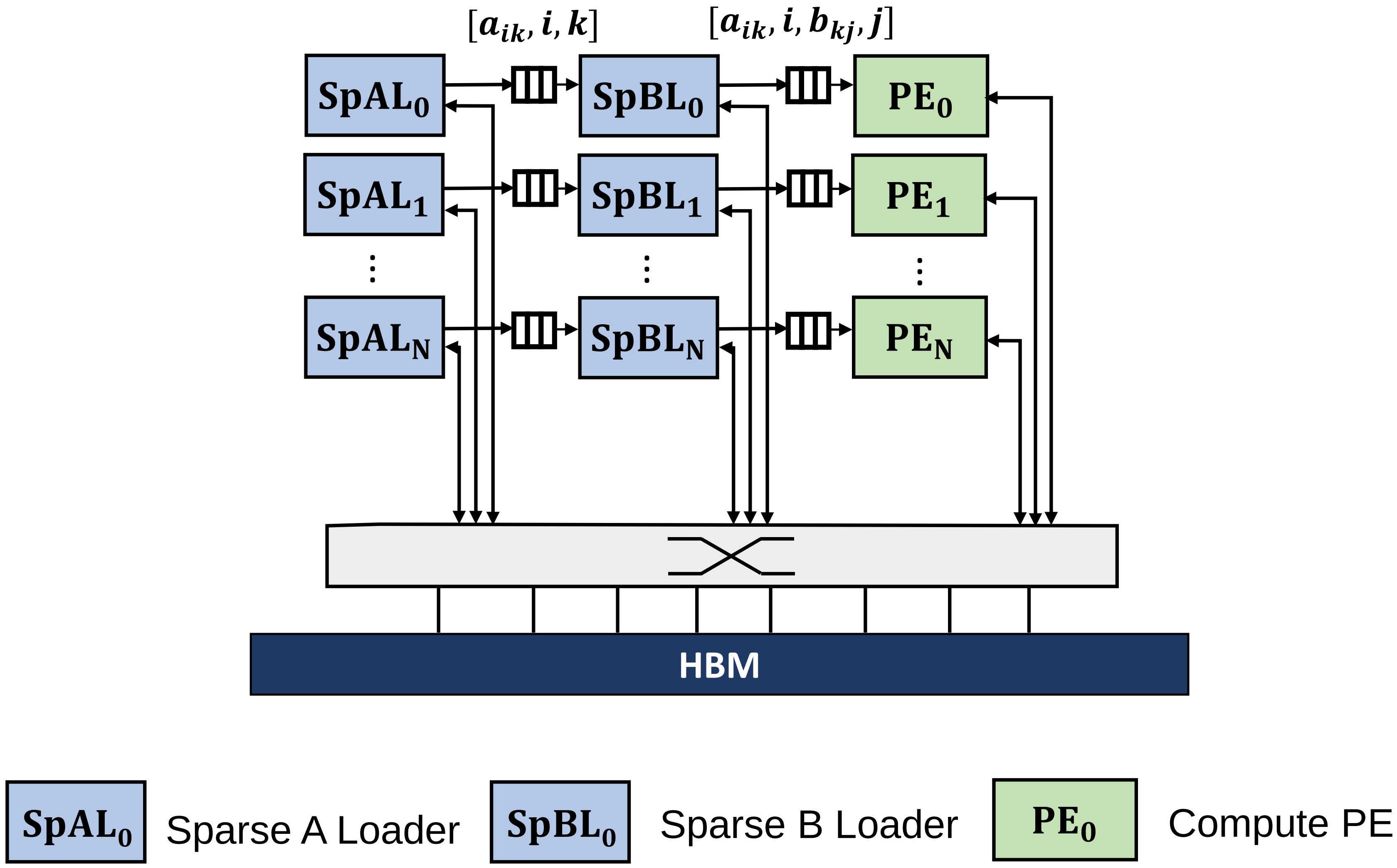
C²SR for Sparse-Sparse MM



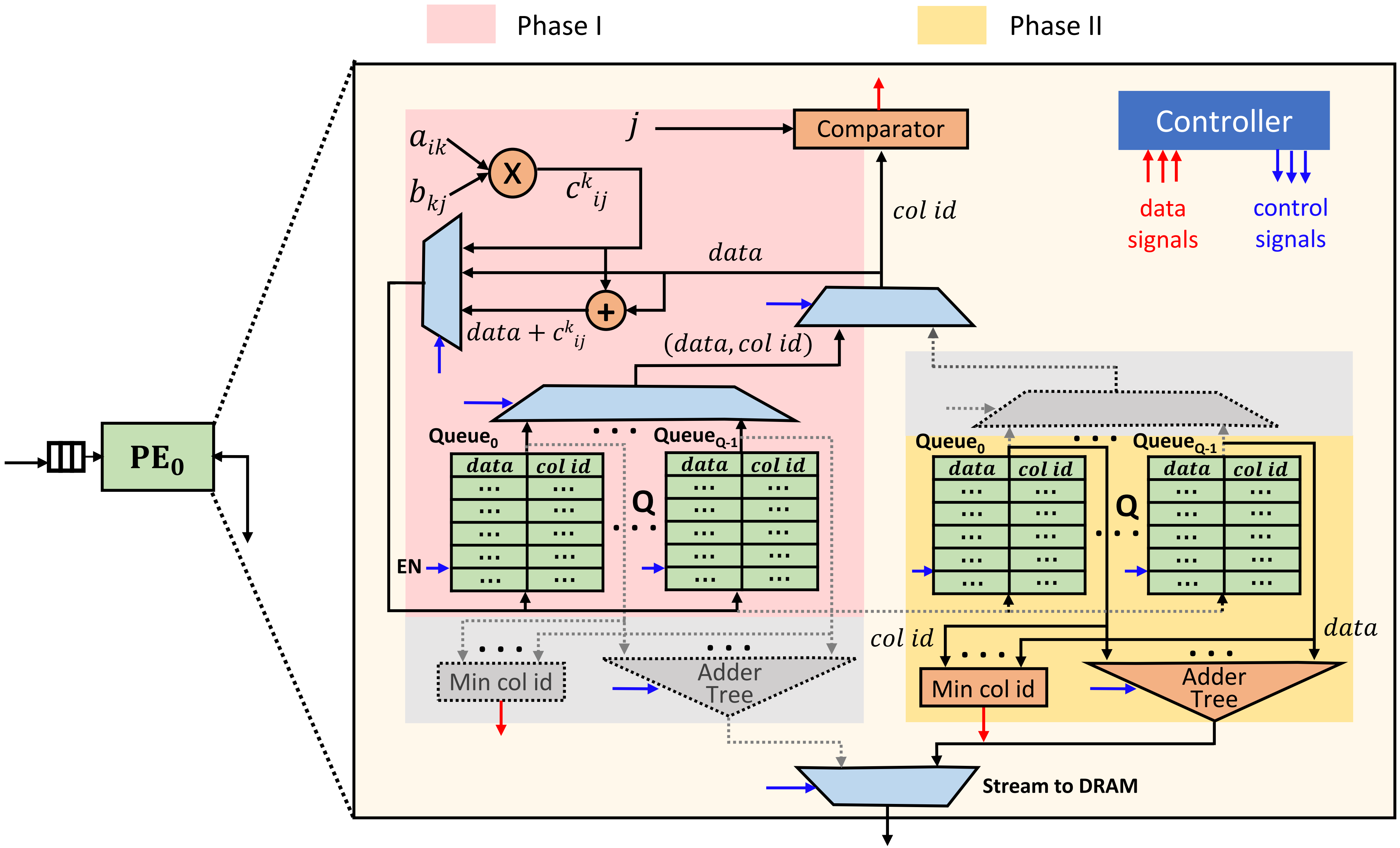
Streaming and vectorized accesses

Streaming and vectorized accesses within each channel

MatRaptor Micro-architecture



PE Micro-architecture



Evaluation Methodology

- **Cycle-level simulation in gem5**
  - 8 PE array, Memory Width 128 bytes
  - 10 8 KB Queues (RAMs) per PE
  - HBM: 8 128-bit physical channels (128 GB/s peak bandwidth)

- **RTL Modeling of a PE using PyMTL**

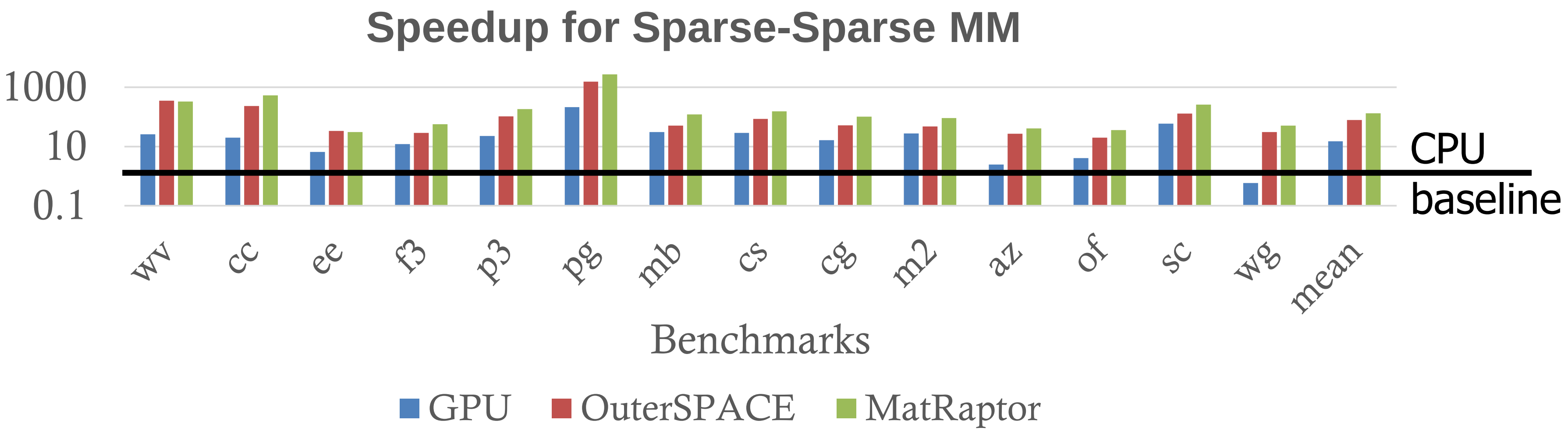
- **Baselines**
  - CPU: Intel(R) Xeon(R) CPU E7-8867
    - Intel MKL
  - GPU: Titan XP
    - CuSparse
  - Accelerator:
    - OuterSPACE

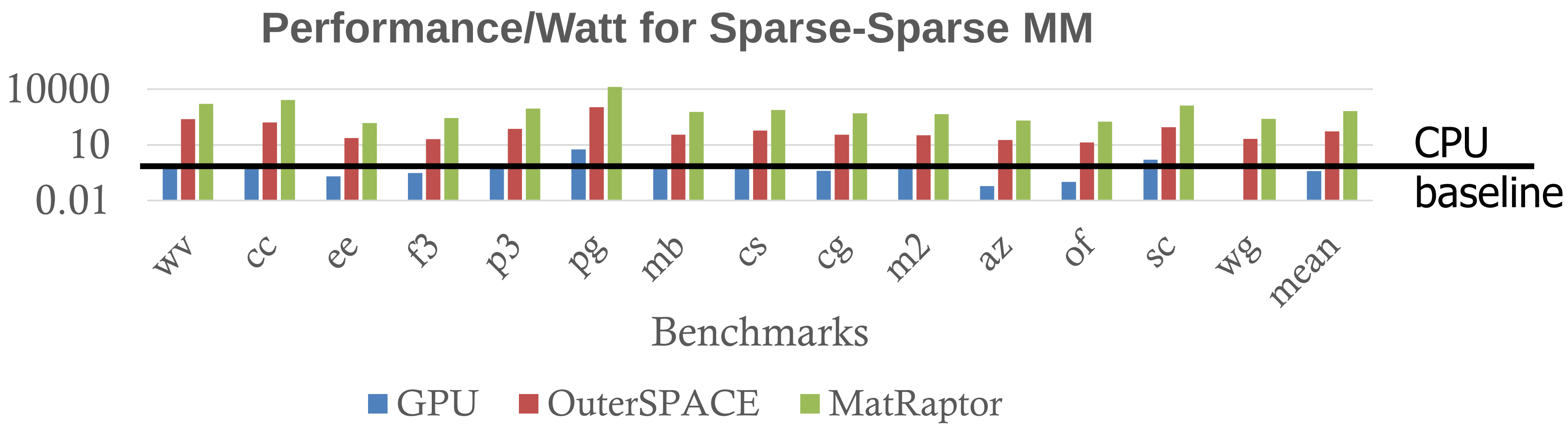
Area and Power Breakdown

Component	Area (mm^2)	%	Power (mW)	%
PE	1.981	88.44 %	1050.57	78.46 %
– Logic	0.080	3.61 %	43.08	3.22 %
– Sorting Queues	1.901	84.83 %	1007.49	75.24 %
SpAL	0.129	5.78 %	144.15	10.77 %
SpBL	0.129	5.78 %	144.15	10.77 %
Total	2.241	100 %	1338.89	100 %

- **Datasets**
  - Florida Sparse Matrices

Results on Sparse-Sparse MM





MatRaptor is 158.7x, 6.9x and 1.7x faster, and 780x, 1300x and 12x more energy-efficient than CPU, GPU and OuterSPACE

Dissertation Summary

Dense

Productivity

T2S-Tensor: Solves productivity issue by providing a language that decouples algorithm and hardware customization

Dissertation Summary

Dense

Productivity

T2S-Tensor: Solves productivity issue by providing a language that decouples algorithm and hardware customization

Sparse

Flexibility, Efficiency

Tensaurus: A hardware accelerator that is both flexible (runs multiple dense & sparse-dense tensor kernels) and efficient (outperforms state-of-the-art accelerator)

MatRaptor: A hardware accelerator for spars-sparse MM that is more efficient than the existing state-of-the-art accelerator

Publications

- **Nitish Srivastava**, Jie Liu, Hanchen Jin, David Albonesi and Zhiru Zhang, “MatRaptor: A Sparse-Sparse Matrix Multiplication Accelerator Based on Row-Wise Product Approach”, in MICRO’20
- **Nitish Srivastava**, Hanchen Jin, Shaden Smith, Hongbo Rong, David Albonesi and Zhiru Zhang, “Tensaurus: A versatile accelerator for Mixed Sparse-Dense Tensor Computations”, in HPCA’20
- **Nitish Srivastava**, Hongbo Rong, *et al.*, “T2S-Tensor: Productively Generating High-Performance Spatial Hardware for Dense Tensor Computations”, in FCCM’19
- **Nitish Srivastava** and Rajit Manohar, “Operation Dependent Frequency Scaling Using Desynchronization”, in TVLSI’19
- Yuan Zhou, Udit Gupta, Steve Dai, Ritchie Zhao, **Nitish Srivastava**, *et al.*, “Rosetta: A Realistic HLS Benchmark Suite for Software Programmable FPGAs”, in FPGA’18
- **Nitish Srivastava**, Steve Dai, Rajit Manohar and Zhiru Zhang, “Accelerating Face Detection on Programmable SoC Using C-Based Synthesis”, in FPGA’17

Acknowledgements

- **Special Committee**
  - Prof. Zhiru Zhang
  - Prof. David Albonesi
  - Prof. Christopher Batten
  - Prof. Rajit Manohar

- **Members (past and present) of CSL**
  - Hanchen Jin, Sean Li, Jie Liu, Yuan Zhou, Neeraj Kulkarni (Cornell)
  - Dr. Shreesha Srinath (SciFive)
  - Dr. Steve Dai (NVIDIA)
  - Dr. Gai Liu (Xilinx)
  - Dr. Skand Hurkat, Dr. Ritchie Zhao (Microsoft)
  - Members of Zhang and Albonesi group

- **Outside Collaborators**
  - Hongbo Rong, Christopher Hughes, Pradeep Dubey, Paul Petersen, Tim Mattson, Geoff Lowney (Intel)
  - Prithayan Barua, Vivek Sarkar (Georgia Tech)
  - Guanyu Feng, Huanqi Cao, Wenguang Chen (Tsinghua University)

- **Funding Sources**
  - NSF, DARPA, JUMP CRISP

Thank you! Questions?

Design and Generation of Efficient Hardware Accelerators for Tensor Computations

Nitish Srivastava

January 14, 2020

Electrical and Computer Engineering, Cornell University