ECE 587 – Hardware/Software Co-Design
Lecture 19
Neural Networks and Systolic Array

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

March 26, 2025

Outline

Reading Assignment

- This lecture: Neural Networks and Systolic Array
- Next lecture: Quantization

Outline

(Artificial) Neural Networks

- A model of computation inspired by biological neurons.
  - Still, we don't know how biological neural networks work.
  - Dated back to 1940's but with a few AI winters.
- Substantial progress since the last decade.
  - Availability of large amount of data.
  - Availability of GPUs for general-purpose computing.
- Most neural networks are DFGs.
  - Feedforward, uni-directional, without cycles or loops.
  - A node compute its output as a simple function of its inputs, e.g. weighted summation, activation, and softmax.
  - Together, any vector-valued function can be approximated.
- Layers: tremendous number of nodes are organized into layers to facilitate reasoning and implementation.
  - Layers are ordered so that outputs from previous layers are used as inputs to next layers.

A Typical Layer

$$\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})$$

- $\boldsymbol{x}$: input vector of this layer
- $\boldsymbol{h}$: output vector of this layer
- $\boldsymbol{W}$, $\boldsymbol{b}$: weight matrix and bias vector
  - Could be fixed paramaters or inputs to the layer.
- g: activation function
  - A fixed nonlinear function applied element-wise to a vector.
- Learning by approximating known input/output relations.
  - Find a good number of layers and then $\boldsymbol{W}$ and $\boldsymbol{b}$ for each layer.
  - Challenge: generalization – the learned model should also perform nicely on unseen inputs.
  - Deep learning: models with more layers tend to generalize better as they require less dimension in $\boldsymbol{W}$ and $\boldsymbol{b}$.

Computation vs. Communication

$$\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})$$

- Use simple assumptions to predict bottlenecks for neural network hardware accelerators.
- Assume $\boldsymbol{x}$ to be a $N \times 1$ vector and $\boldsymbol{W}$ to be a $N \times N$ matrix.
- Computation
  - $N \times N$ multiply–accumulate operations (MACs).
  - N activations.
- Communication
  - Assume $\boldsymbol{W}$ cannot all fit into memory but $\boldsymbol{x}$ and $\boldsymbol{b}$ can.
  - $2N$ to load $\boldsymbol{x}$ and $\boldsymbol{b}$ into memory.
  - $N \times N$ to process $\boldsymbol{W}$ column by column.
  - N to output $\boldsymbol{h}$.
- Too few computation per communication $(1:1)$
  - Difficult to design efficient hardware accelerators.
- Can we increase the computation-to-communication ratio?

Batch Processing

$$\boldsymbol{H} = g(\boldsymbol{W}^\top \boldsymbol{X} + \boldsymbol{B})$$

- There are cases where outputs are computed from multiple (M) inputs for the same neural network, e.g. for training.
  - $M << N$ so that $\boldsymbol{X}$ and $\boldsymbol{B}$ can fit into memory.
- $\boldsymbol{H}$, $\boldsymbol{X}$, $\boldsymbol{B}$ are $N \times M$ matrices while $\boldsymbol{W}$ is $N \times N$ matrix.
- Computation
  - $M \times N \times N$ multiply–accumulate operations (MACs).
  - $M \times N$ activations.
- Communication
  - $N + N \times M$ to load $\boldsymbol{X}$ and $\boldsymbol{B}$ into memory.
  - $N \times N$ to process $\boldsymbol{W}$ column by column.
  - $N \times M$ to output $\boldsymbol{H}$ row by row.
- Better computation-to-communication ratio $M : 1$.

Convolutional Neural Networks

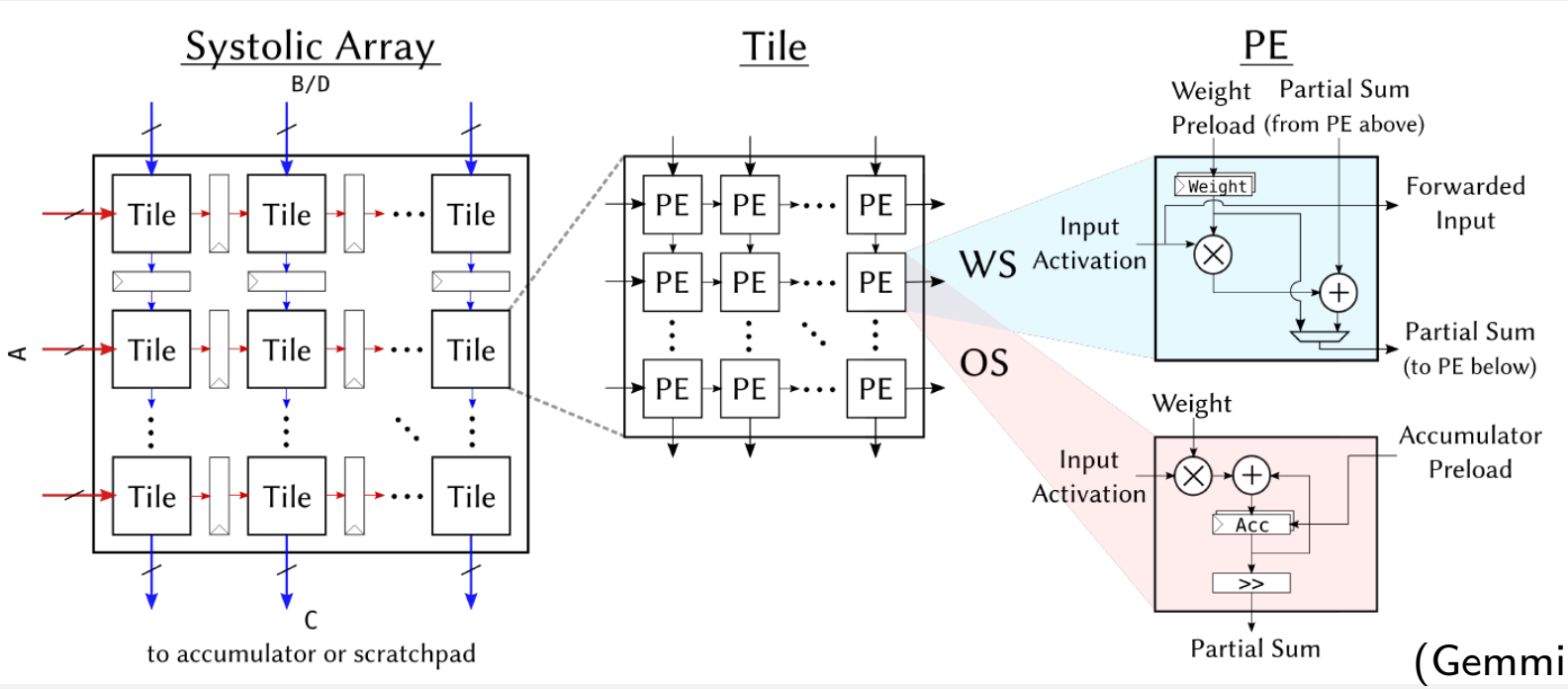
$$\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})$$

- $\boldsymbol{x}$ may have internal structures for particular applications.
  - E.g. 3-D (width/height/channels) for images.
- $\boldsymbol{W}$ maybe designed specifically for such structure.
  - With less freedom to improve generalization.
  - E.g. convolution where weighted summations are computed from a small moving window of $\boldsymbol{x}$ via identical kernels.
- Computation: reduced as many elements of $\boldsymbol{W}$ are 0.
  - kN assuming each element of $\boldsymbol{x}$ corresponds to k non-zeros in $\boldsymbol{W}$ with $k << N$.
- Communication: reduced as $\boldsymbol{W}$ can fit into memory.
  - $2N$ to input $\boldsymbol{x}$ and output $\boldsymbol{h}$
- Better computation-to-communication ratio $k:1$

Outline

Systolic Array

- An array of processing elements (PEs) with highly regular layout and local interconnects.
  - To match dataflow of a particular computational task.
  - Without the need to load/store intermediate results frequently.
  - E.g. for matrix-matrix multiplication.
- Support massively parallel datapath computing via scaling.
  - Communications between PEs should be local.
  - Avoid complex control logics when reusing data.



(Gemmini)

Matrix Multiplication $C = AB + D$

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ \dots & \dots \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \dots & \dots \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} + \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \\ \dots & \dots \end{pmatrix}$$

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} + d_{11}, & c_{12} &= a_{11}b_{12} + a_{12}b_{22} + d_{12}, \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} + d_{21}, & c_{22} &= a_{21}b_{12} + a_{22}b_{22} + d_{22}, \\ & \dots, & & \dots, \end{aligned}$$

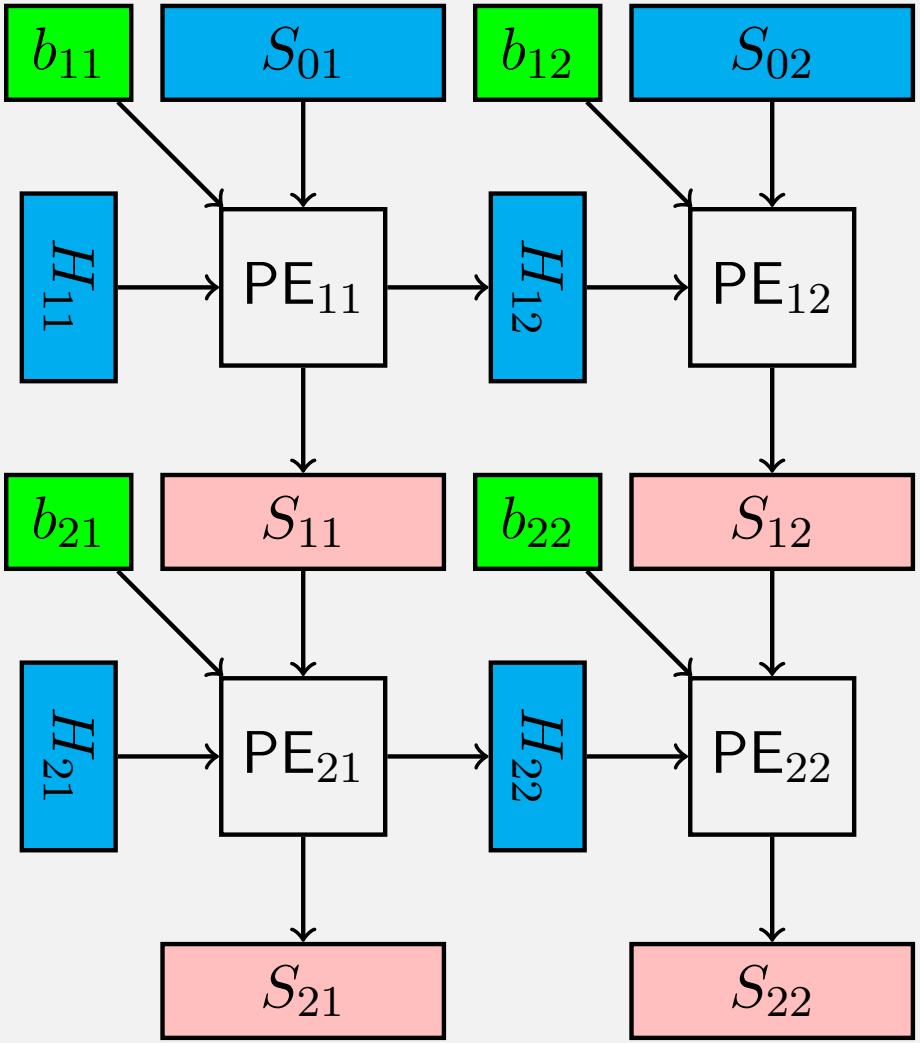
- Consider a simple case where size of B is small.
  - E.g. a 2×2 matrix.
- Weight-stationary dataflow
  - Preload B into the accelerator.
  - Process A, D, and C row by row.

Array Design for Weight-Stationary Dataflow

$$
\begin{aligned}
c_{11} &= a_{11}b_{11} + a_{12}b_{21} + d_{11}, & c_{12} &= a_{11}b_{12} + a_{12}b_{22} + d_{12}, \\
c_{21} &= a_{21}b_{11} + a_{22}b_{21} + d_{21}, & c_{22} &= a_{21}b_{12} + a_{22}b_{22} + d_{22}, \\
&\ldots, & &\ldots,
\end{aligned}
$$

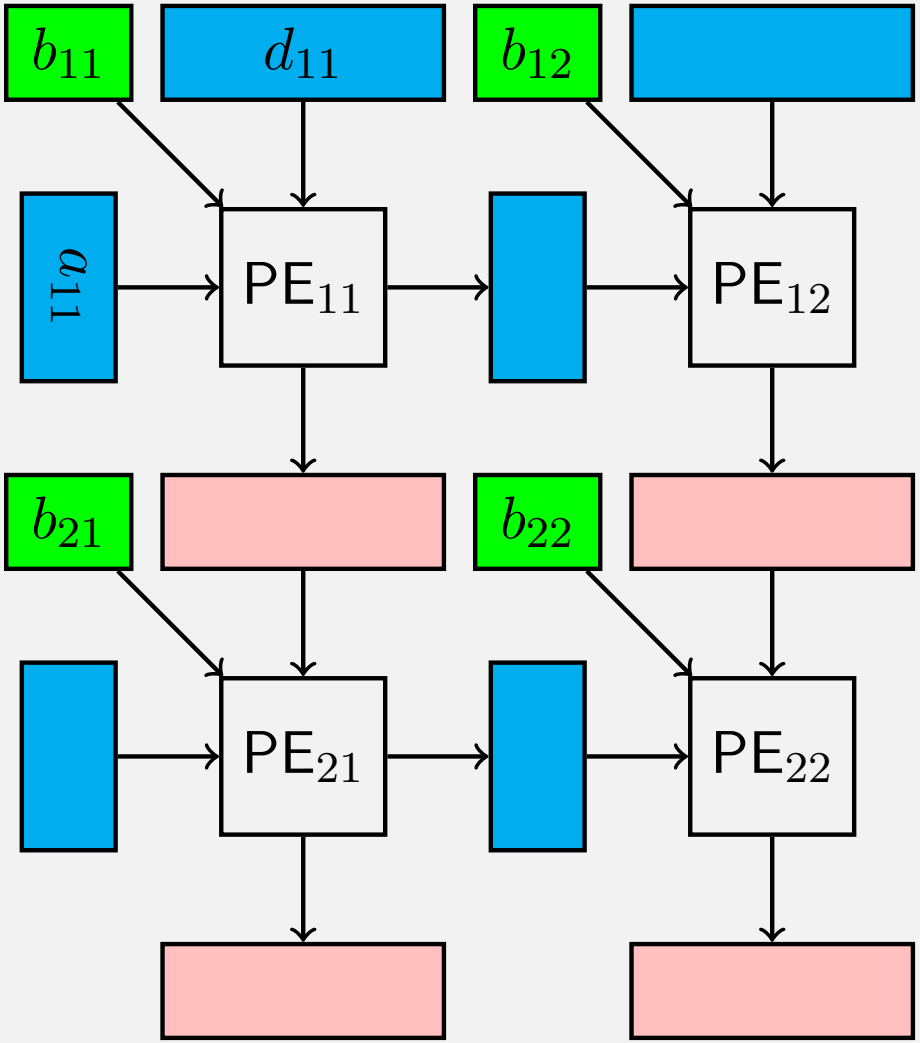
- Use an array of multiply-accumulate (MAC) PEs.
  - Each PE completes one MAC operation per clock cycle.
- Match the shape of the array to that of $\boldsymbol{B}$.
  - Each PE performs multiplications with one fixed element of $\boldsymbol{B}$.
- How to connect PEs? How to schedule and bind operations?
  - So that the design can be scaled to larger arrays.
  - Need to avoid global interconnects and complex control logics.

Example: 2×2 Systolic Array



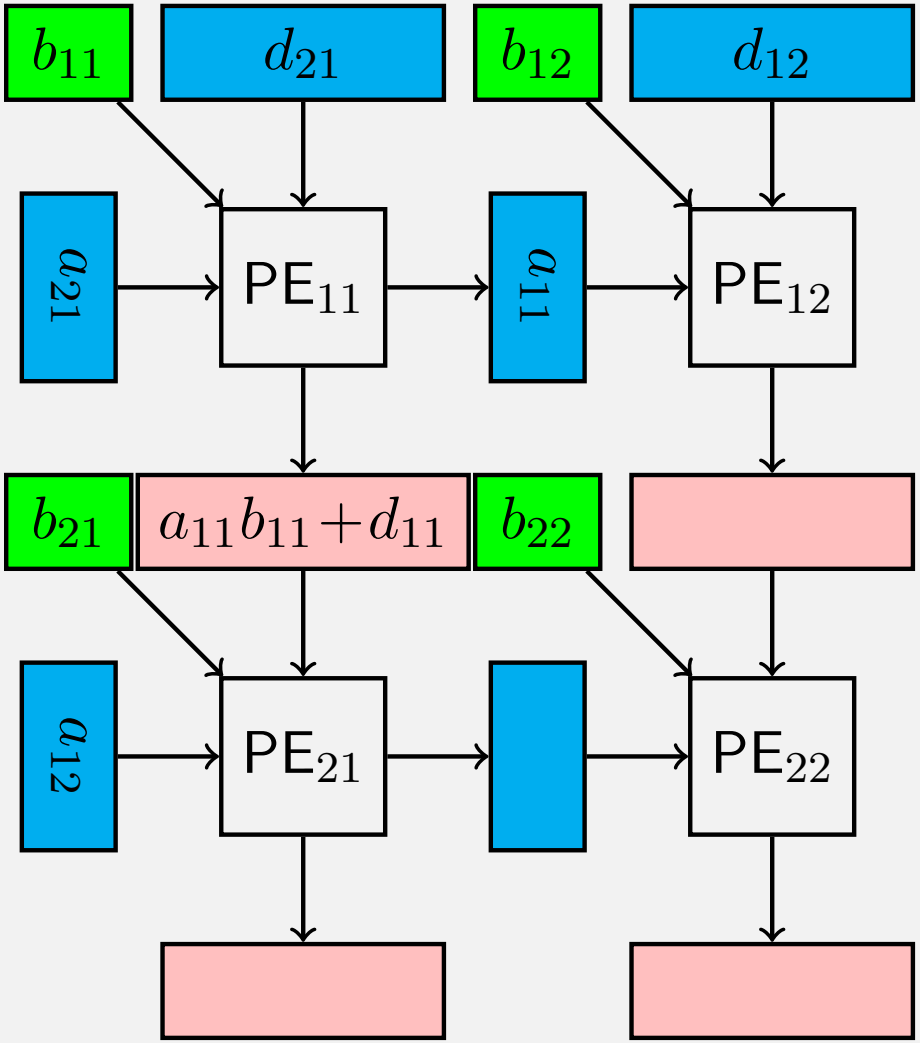
- Registers are colored.
- Preload $\boldsymbol{B}$ to green registers.
- For each cycle (next state),
  - Load a row of $\boldsymbol{A}$ into H_{11}, H_{21}
  - Load a row of $\boldsymbol{D}$ into S_{01}, S_{02}
  - Store S_{21}, S_{22} into a row of $\boldsymbol{C}$
  - $H_{12} \leftarrow H_{11}$, $H_{22} \leftarrow H_{21}$
  - $S_{11} \leftarrow b_{11} * H_{11} + S_{01}$
  - $S_{12} \leftarrow b_{12} * H_{12} + S_{02}$
  - $S_{21} \leftarrow b_{21} * H_{21} + S_{11}$
  - $S_{22} \leftarrow b_{22} * H_{22} + S_{12}$
- Local interconnects.
- Simple control logics.

Example: Cycle 1



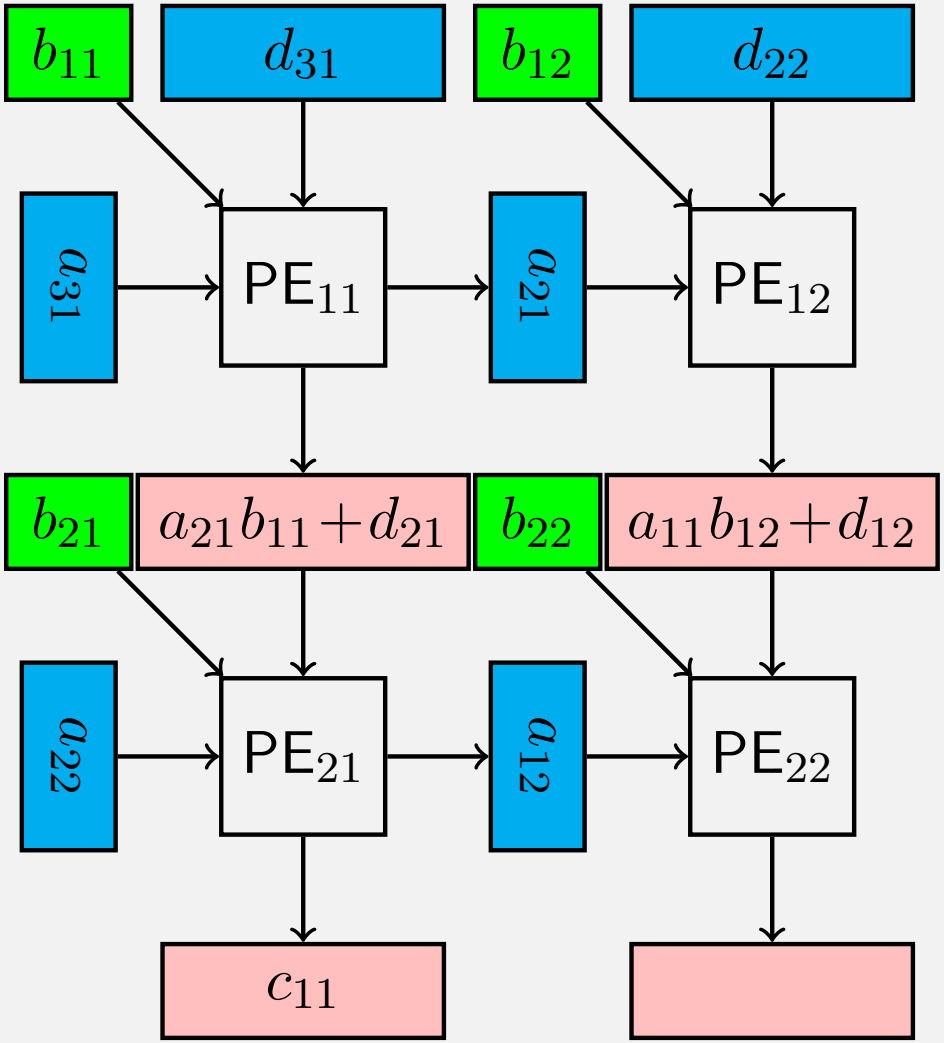
- Actually, rows of $\boldsymbol{A}$, $\boldsymbol{D}$, and $\boldsymbol{C}$ are staggered for load/store.
  - a_{11} and d_{11} become available at the beginning of the first cycle.

Example: Cycle 2



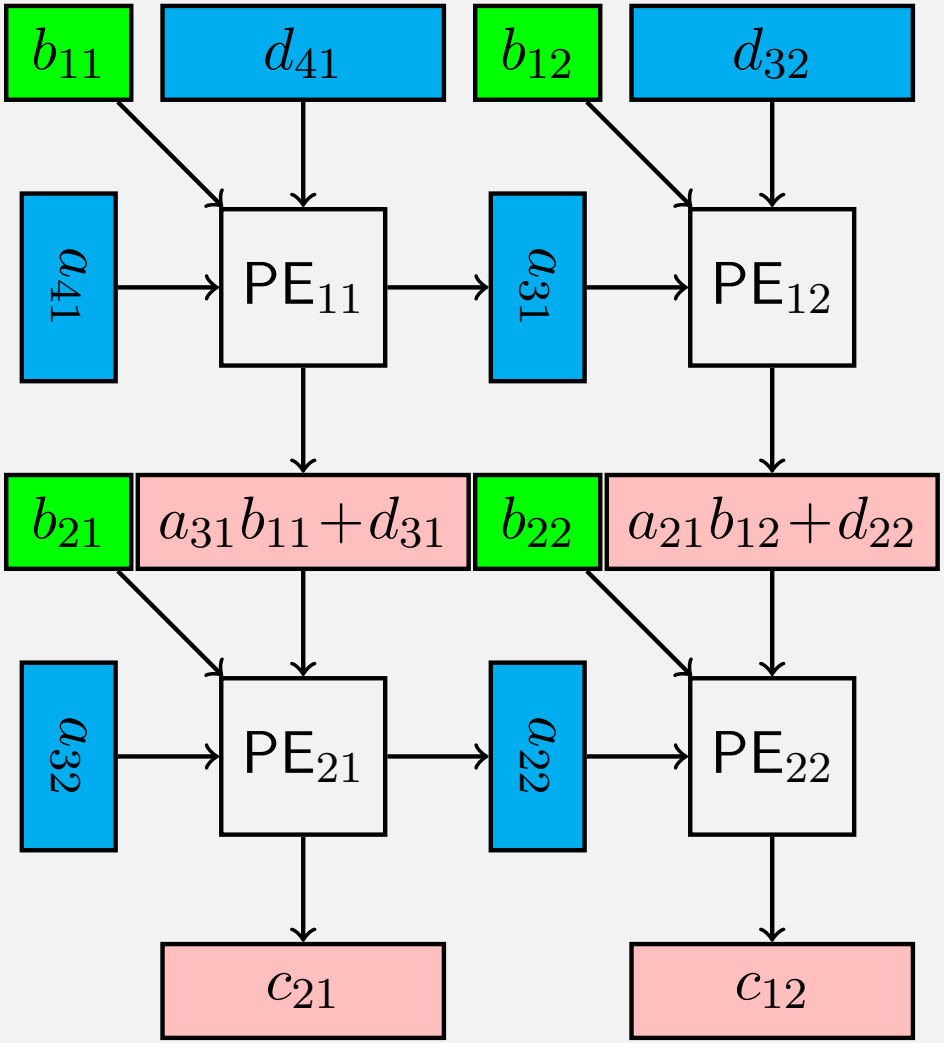
- $c_{11} = a_{11}b_{11} + a_{12}b_{21} + d_{11}$ will be stored next cycle.

Example: Cycle 3



- $c_{21} = a_{21}b_{11} + a_{22}b_{21} + d_{21}$ will be available next cycle.
- $c_{12} = a_{11}b_{12} + a_{12}b_{22} + d_{12}$ will be available next cycle.

Example: Cycle 4

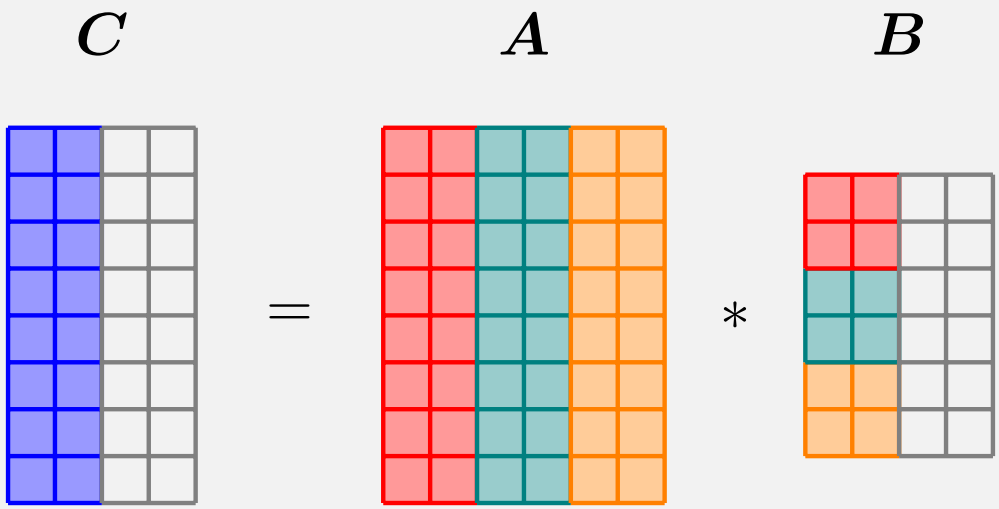


- $c_{31} = a_{31}b_{11} + a_{32}b_{21} + d_{31}$ will be available next cycle.
- $c_{22} = a_{21}b_{12} + a_{22}b_{22} + d_{22}$ will be available next cycle.
- Eventually, 2 cycles after loading all rows of $\boldsymbol{A}$ and $\boldsymbol{D}$, all rows of $\boldsymbol{C}$ are computed and stored.

Additional Details

- It seems 3 PEs are idle for Cycle 1.
  - Keep PEs always busy to improve utilization.
- What if we need to compute another set of $\boldsymbol{C} = \boldsymbol{AB} + \boldsymbol{D}$?
  - PEs could be kept busy by preloading this set of $\boldsymbol{B}$ while computing with the previous set.
- Double buffer: provide two sets of storage for $\boldsymbol{B}$
  - Compute with one set of $\boldsymbol{B}$ while preload the next set.
  - Preload $\boldsymbol{B}$ by shifting rows in a staggered manner to match the loading pattern of $\boldsymbol{A}$.
  - Tag elements of $\boldsymbol{B}$ to indicate which set to use and to stop shifting once elements reach their desiginated PEs.
- What about multiplying matrices with arbitrary sizes?
  - Use block matrix multiplication to decompose the computation into smaller matrix multiplications that fit into the array.

Block Matrix Multiplication

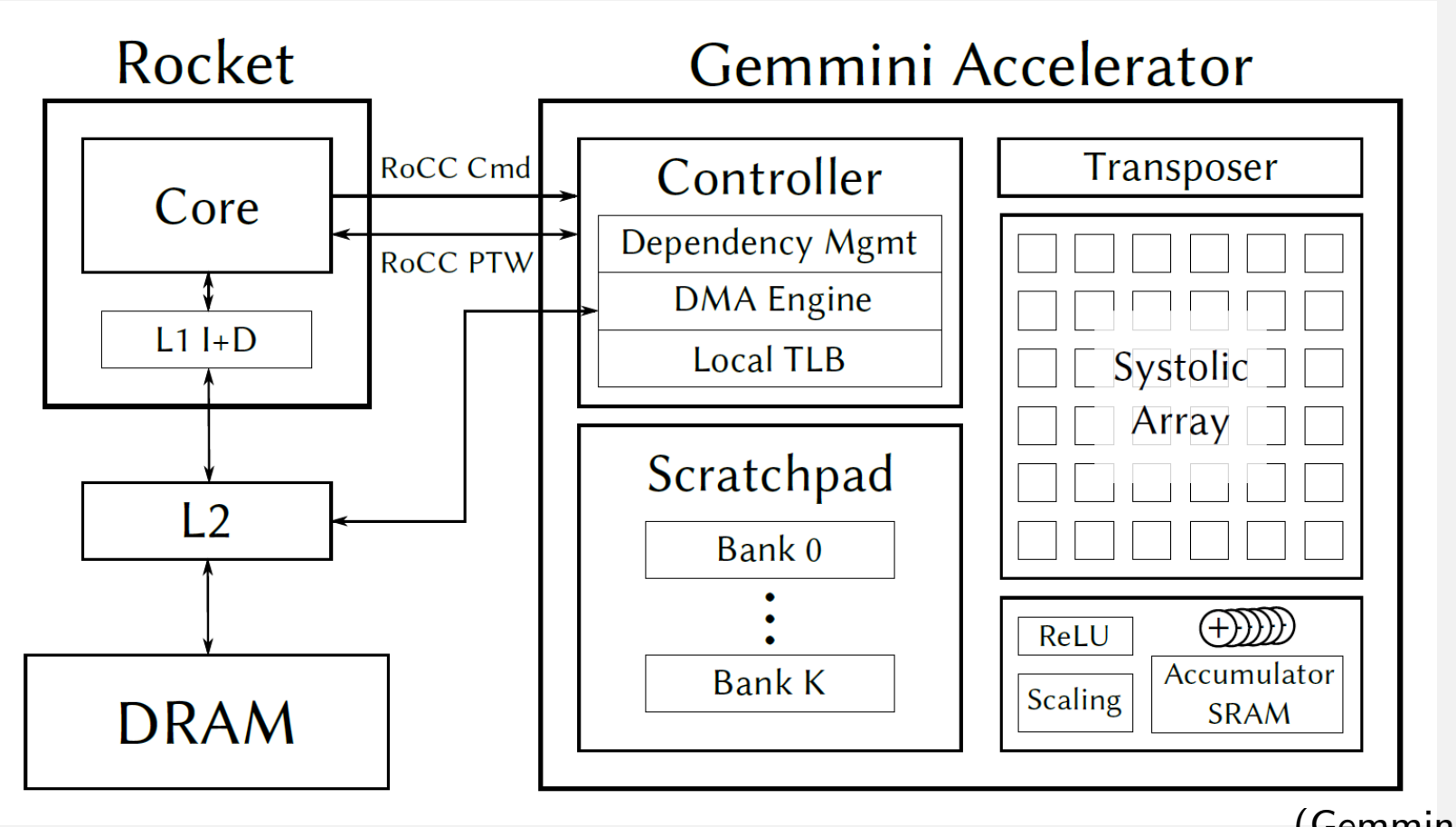


- $\boldsymbol{C}[1:8, 1:2]$ can be computed in 3 array multiplications.
  1. $\boldsymbol{C}[1:8, 1:2] = \boldsymbol{A}[1:8, 1:2]\boldsymbol{B}[1:2, 1:2]$
  2. $\boldsymbol{C}[1:8, 1:2] = \boldsymbol{A}[1:8, 3:4]\boldsymbol{B}[3:4, 1:2] + \boldsymbol{C}[1:8, 1:2]$
  3. $\boldsymbol{C}[1:8, 1:2] = \boldsymbol{A}[1:8, 5:6]\boldsymbol{B}[5:6, 1:2] + \boldsymbol{C}[1:8, 1:2]$
- Overall, $\boldsymbol{C}$ can be computed in 6 array multiplications.
  - Need controller to coordinate data movement.

Outline

Gemmini

- A full-system, full-stack DNN hardware exploration and evaluation platform.
  - Part of Chipyard and written in Chisel.



(Gemmini)

Gemmini Features

- A RoCC (RISC-V Custom Coprocessor) accelerator.
  - Connect to a Rocket or BOOM tile through the RoCC port and interfaces.
  - Execute custom RISC-V instructions sent by RISC-V processor.
- Feature a systolic array for efficient matrix multiplication.
  - Support weight-stationary and other dataflows.
  - Addtional accumulator consisting of SRAM storage and adders for weight-stationary dataflow.
- Memory architecture
  - Explicitly managed internal scratchpad.
  - Interface with processor memory via the System Bus, linking directly to the L2 cache.
- Hardware support for activation functions (like ReLU), quantization, and matrix transpose.

Scratchpad and Accumulator SRAM

- The systolic array has a dimension of DIMxDIM.
- Both scratchpad and accumulator SRAM consist of rows.
  - Addressed in the same shared private memory space by rows.
  - Each row contains DIM elements.
  - The elements are of inputType for scratchpad.
  - The elements are of accType for accumulator.
- accType usually has more bits than inputType so that accumulator can add with higher precision.
- Activations and quantizations may be applied when moving data from accumulator SRAM to scratchpad.

Gemmini Instructions

- Provide low-level control over three internal pipeplines
- Load pipeline and store pipeline
  - gemmini_config_ld and gemmini_config_st: let Gemmini know the matrix row size in the main memory.
  - gemmini_mvin and gemmini_mvout: load a DIMxDIM block from L2 to Gemmini, or store a DIMxDIM block to L2.
- Execute pipeline
  - gemmini_config_ex: configurate options for multiplication, e.g. dataflow and activation function.
  - gemmini_preload: move B from Gemmini scratchpad into the systolic array, configurate where C should be stored.
  - gemmini_compute_preloaded: move A into the systolic array, multiply B and write result to C as set by gemmini_preload.
- Work on the project to learn more on high-level Gemmini functions that are easier to use.