ECE 587 – Hardware/Software Co-Design
Lecture 22 Systolic Array

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

April 1, 2026

Outline

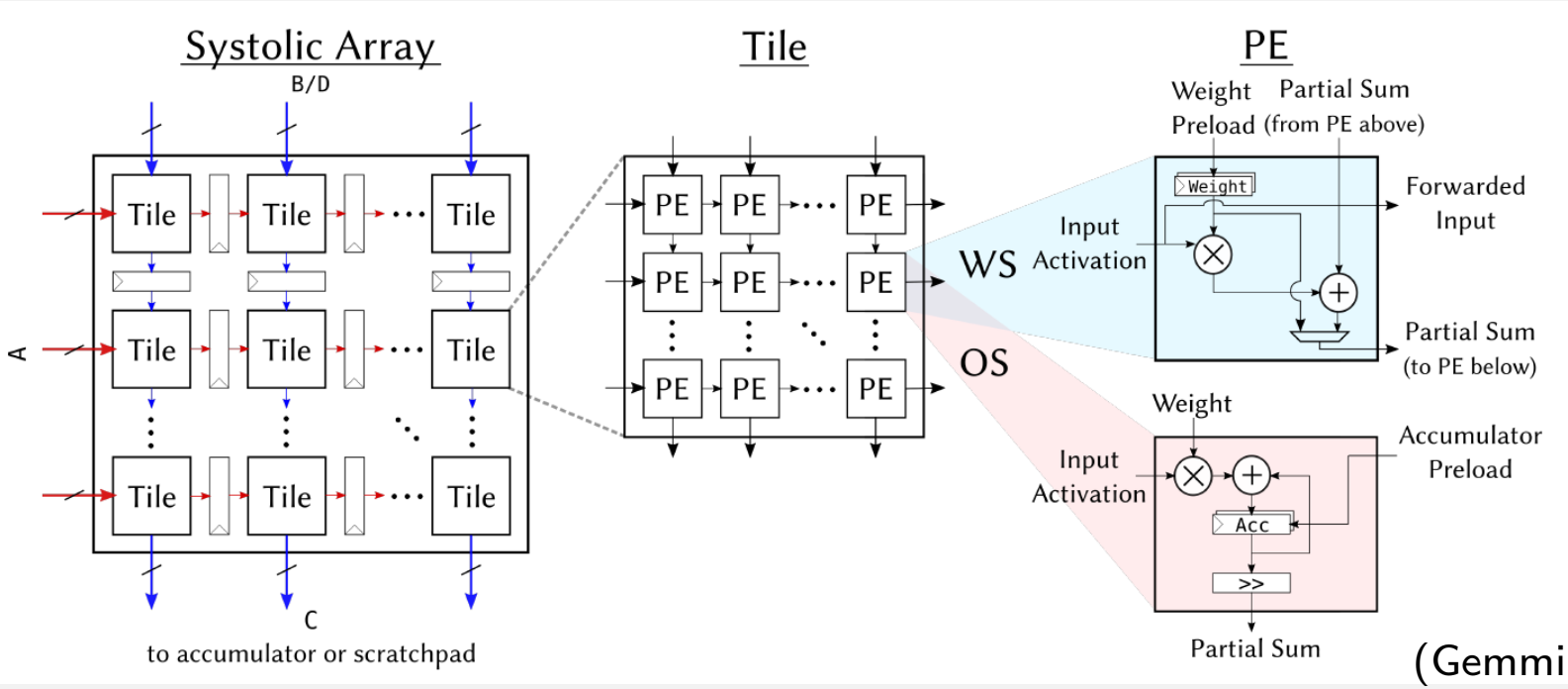
Reading Assignment

- This lecture: Systolic Array
- Next lecture: 6

Outline

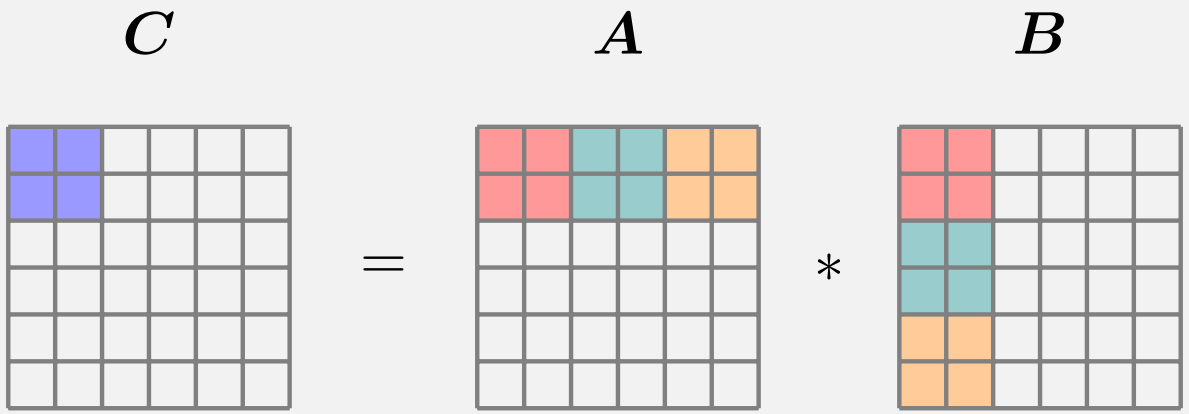
Systolic Array

- 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.
- Supports massively parallel datapath computing via scaling.
  - Communications between PEs should be local.
  - Avoid complex control logic when reusing data.



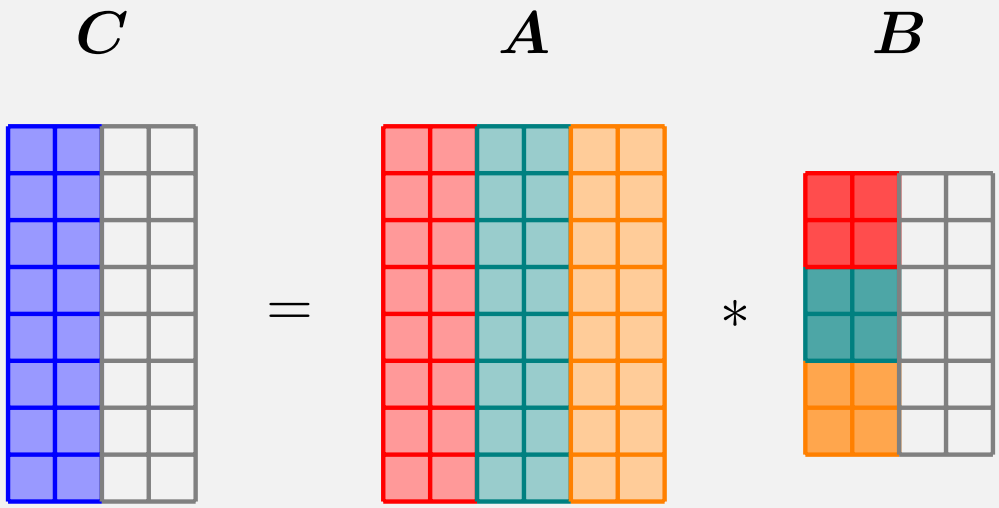
(Gemmini)

Block Matrix Multiplication Revisited



- For both of our tiled GEMM optimizations on CPU and CUDA, partial results for a tile of C are held in cache/registers while tiles from A and B are processed.
- This pattern is known as output-stationary since the output C is kept on chip.
- Is there a different way to organize block matrix multiplications?

Weight-Stationary Block Matrix Multiplication



- Weight-stationary: we may hold a tile of B on chip while reading columns of A to update columns of C.
- Depending on the shapes of A, B, and C and other factors, one may prefer weight-stationary over output-stationary or the other way around.
- Let's focus on weight-stationary for today as we are familiar with output-stationary from previous lectures.

Weight-Stationary Matrix Multiplication

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

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

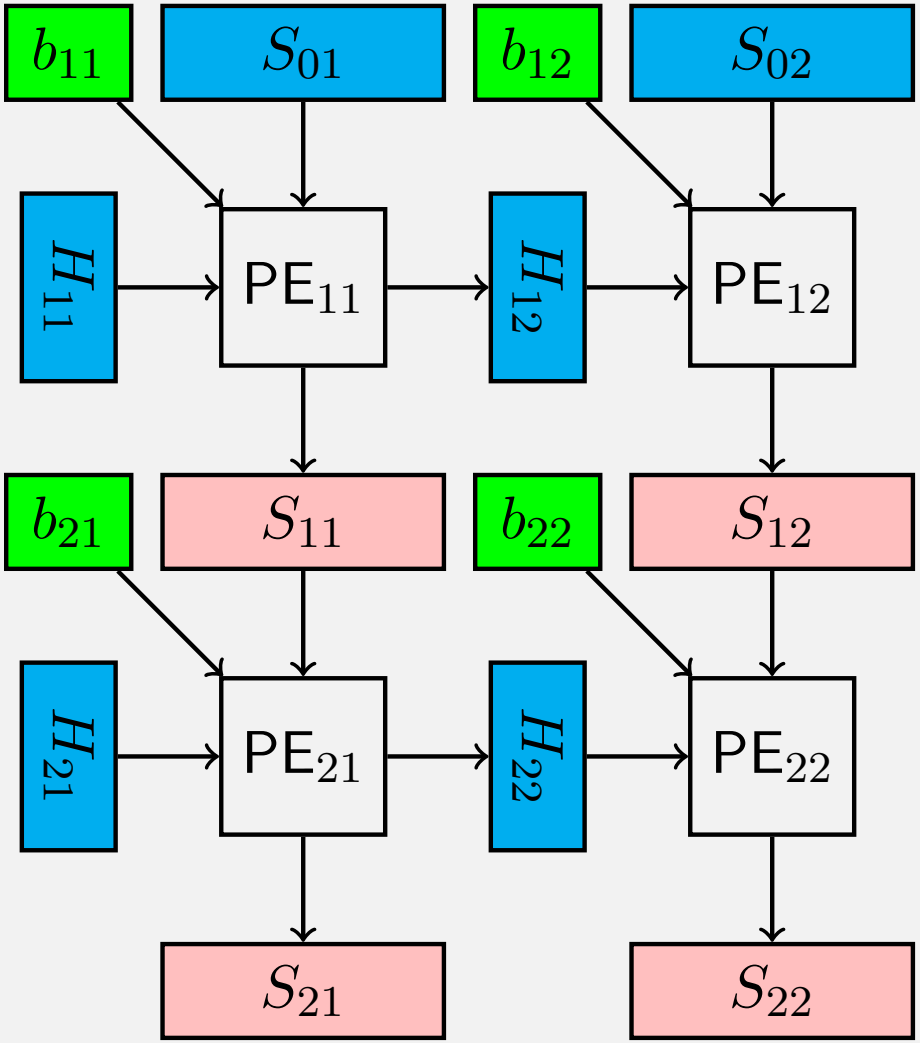
- Consider multiplication with a tile of $\boldsymbol{B}$.
  - E.g. a 2×2 tile.
  - Now we need to use 2 columns of $\boldsymbol{A}$ to update 2 corresponding columns of the partial sums stored in $\boldsymbol{C}$.
- Weight-stationary dataflow
  - Preload the $\boldsymbol{B}$ tile into the accelerator.
  - Stream the columns of $\boldsymbol{A}$ row by row.
  - Stream the columns of $\boldsymbol{C}$ and update them row by row.

Weight-Stationary Accelerator Design

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

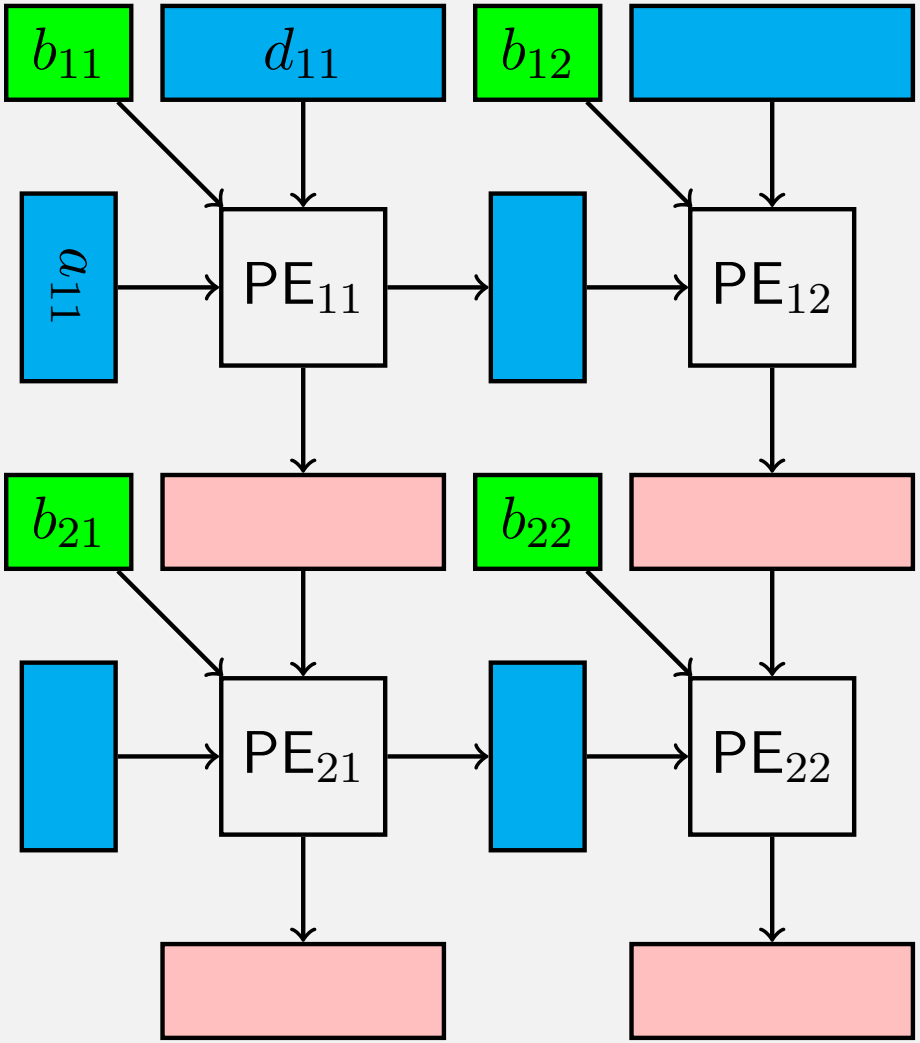
- Use an array of multiply-accumulate (MAC) PEs.
  - Each PE completes one MAC operation per clock cycle.
- Match the shape of the array to that of $\boldsymbol{B}$ tile.
  - Each PE performs multiplications with one fixed element of $\boldsymbol{B}$.
- How to connect PEs? How to schedule operations?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

Example: 2×2 Systolic Array



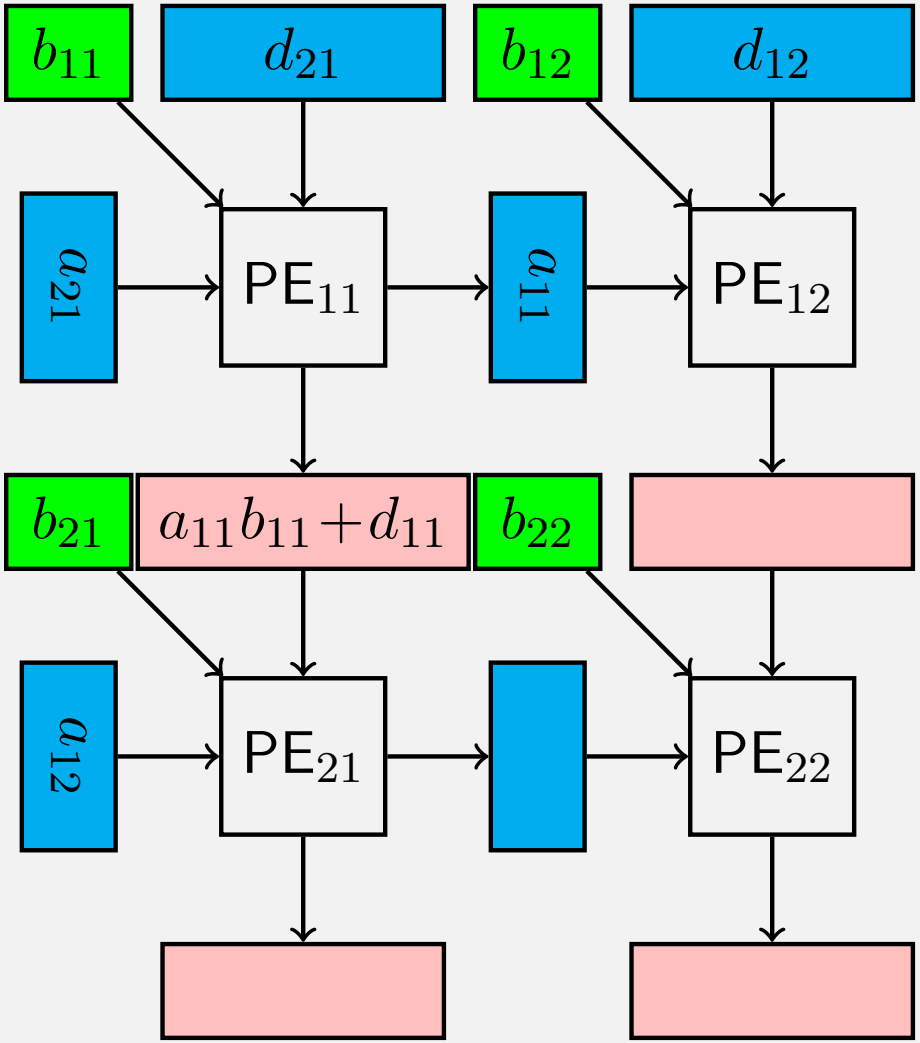
- Registers are colored.
- Preload $\boldsymbol{B}$ to green registers.
- For each cycle (next state),
  - Load a row of $\boldsymbol{A}$ into H_{11}, H_{21}
  - Load a row of $\boldsymbol{D}$ into S_{01}, S_{02}
  - Store S_{21}, S_{22} into a row of $\boldsymbol{C}$
  - $H_{12} \leftarrow H_{11}$,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

Example: Cycle 1



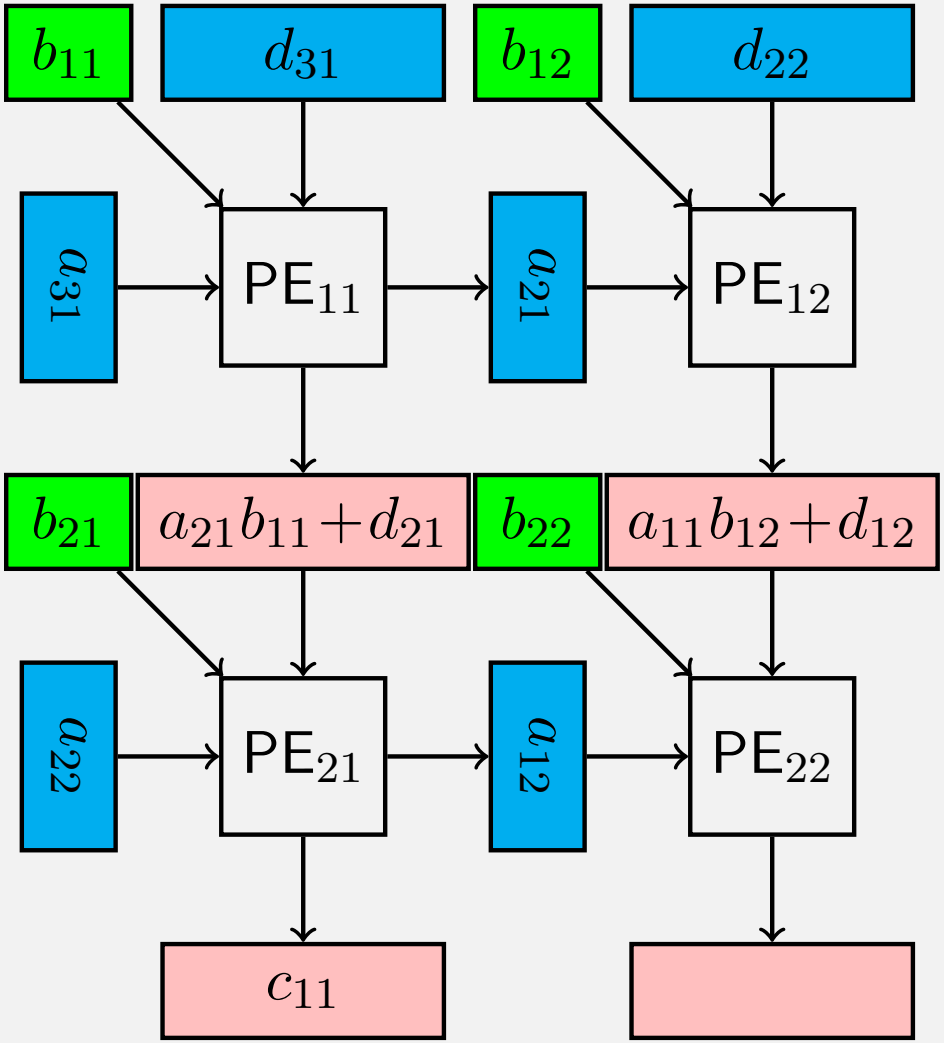
- Actually, rows of $\boldsymbol{A}$, $\boldsymbol{D}$, and $\boldsymbol{C}$ are staggered for load/store.
  - a_{11} and d_{11} become available at the beginning of the first cycle.

Example: Cycle 2



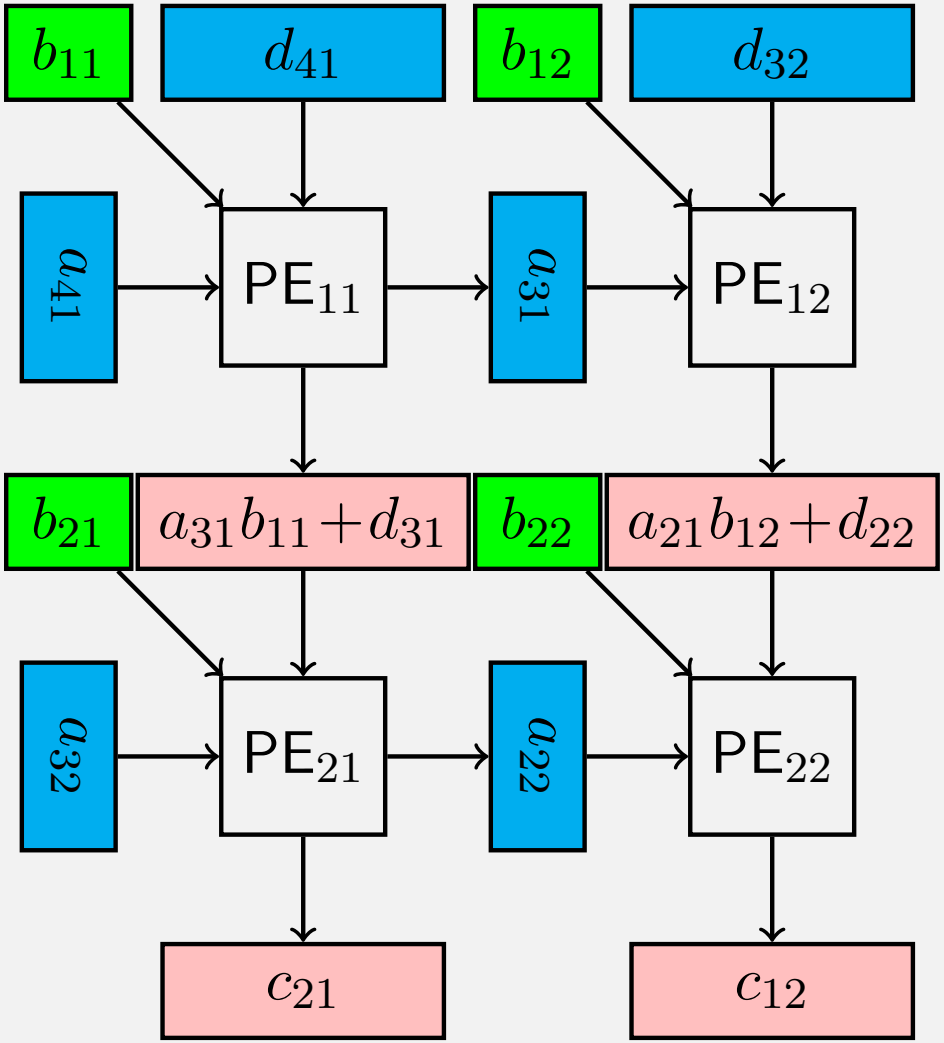
- $c_{11} = a_{11}b_{11} + a_{12}b_{21} + d_{11}$ will be stored next cycle.

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 tiles?
  - PEs could be kept busy by preloading next $\boldsymbol{B}$ tile while computing with the current $\boldsymbol{B}$ tile.
- Double buffer: provide storage for two $\boldsymbol{B}$ tiles
  - Compute with one $\boldsymbol{B}$ tile while preload the next set.
  - Preload $\boldsymbol{B}$ tile by shifting rows in a staggered manner to match the loading pattern of $\boldsymbol{A}$ tiles.
  - Tag elements of $\boldsymbol{B}$ to indicate which set to use and to stop shifting once elements reach their designated PEs.
- How to design an accelerator for output-stationary matrix multiplication?

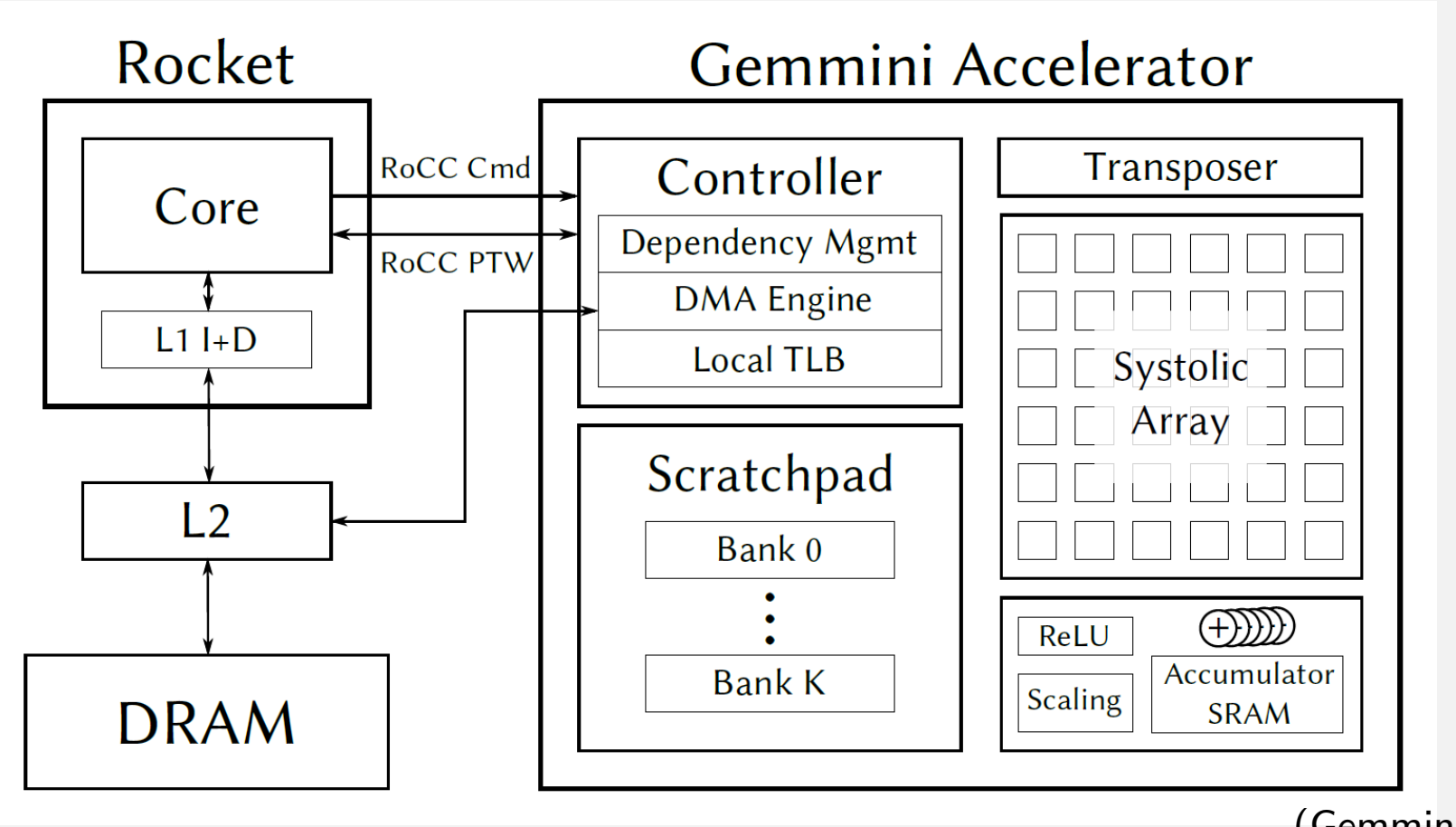
Outline

Systolic Array

Gemmini

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.
- Features a systolic array for efficient matrix multiplication.
  - Support weight-stationary and other dataflows.
  - Additional 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.
  - They are 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 pipelines
- 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: configure options for multiplication, e.g. dataflow and activation function.
  - gemmini_preload: move B from Gemmini scratchpad into the systolic array, configure 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 GEMM Project 4 to learn more on high-level Gemmini functions that are easier to use.