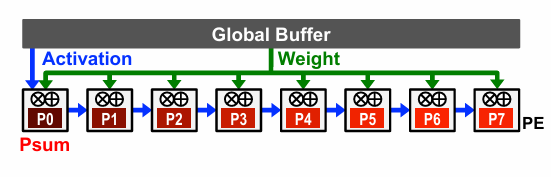
**LECTURE 12: DATAFLOW (CONTINUE)**

**Phân loại luồng dữ liệu: gồm 3 phần (Ouput Stationary, Weight Stationary và Input Stationary)**

***Output Stationary (OS)***

- Output Stationary (OS) là một chiến lược dataflow trong việc thực hiện phép tích chập (convolution) trên các kiến trúc phần cứng chuyên dụng cho deep learning

******

Sơ đồ minh họa một kiến trúc phần cứng sử dụng dataflow kiểu Output Stationary với các thành phần:

**1. Global Buffer**: Bộ đệm toàn cục phía trên cùng, lưu trữ cả activation (đầu vào) và weight (trọng số).

+ Mũi tên xanh dương: Luồng dữ liệu activation từ buffer xuống các PE

+ Mũi tên xanh lá: Luồng dữ liệu weight từ buffer xuống các PE

**2. PE (Processing Elements - P0 đến P7)**: 8 đơn vị xử lý song song, mỗi đơn vị chịu trách nhiệm tính toán một phần tử đầu ra cụ thể.

+ Mỗi PE có bộ tính toán tích tổng (phép nhân và cộng - ⊗⊕)

+ Các PE được sắp xếp thành một mảng ngang

**3.** **Psum (Partial Sum)**: Các tổng riêng phần được tính toán và tích lũy cục bộ trong mỗi PE

***Nguyên lý hoạt động***

1. Mỗi PE được gán trách nhiệm tính toán một phần tử đầu ra cụ thể

2. Các weights và activations được đưa đến các PE từ global buffer

3. Mỗi PE duy trì và tích lũy giá trị đầu ra của mình tại chỗ (giữ ổn định/stationary)

***Ưu điểm***

**1. Giảm thiểu tiêu thụ năng lượng đọc/ghi tổng riêng phần (partial sum):**

**+** Bằng cách "maximize local accumulation" (tối đa hóa tích lũy cục bộ)

+ Các tổng riêng phần được tính và tích lũy ngay trong PE, không cần di chuyển giữa các bộ nhớ

**2. Broadcast/Multicast và Tái sử dụng dữ liệu**:

+ **Filter weights** (trọng số) được broadcast (phát sóng)/multicast (phát đa hướng) đến nhiều PE

+ **Activations** (đầu vào) được tái sử dụng không gian ("reuse spatially") trên toàn mảng PE thông qua Global Buffer

Ưu điểm chính của Output Stationary là giảm thiểu chi phí năng lượng liên quan đến việc di chuyển các giá trị tổng riêng phần, thông qua việc tích lũy cục bộ và tái sử dụng dữ liệu đầu vào/trọng số trên nhiều PE.

***Tại sao gọi đây là Output Stationary?***

**+** Trong dataflow này, giá trị đầu ra (output) được giữ cố định tại một PE cụ thể

+ Mỗi PE chịu trách nhiệm tính toán hoàn chỉnh một phần tử đầu ra

+ Trong khi đầu vào và trọng số được đưa đến các PE, các partial sum không di chuyển mà được tích lũy tại chỗ

***Ứng dụng thực tế***

Output Stationary đặc biệt hiệu quả trong các trường hợp:

+ Các lớp tích chập có kích thước filter lớn

+ Hệ thống có giới hạn băng thông bộ nhớ

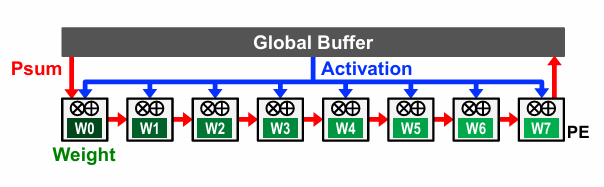
+ Kiến trúc hardware có nhiều PE song song

+ Khi chi phí di chuyển partial sum cao hơn so với weights/activations

Đây là một chiến lược dataflow quan trọng trong thiết kế các bộ tăng tốc tích chập hiệu quả về năng lượng cho các ứng dụng AI và deep learning.

Ví dụ: ShiDianNao, KU Leuven, …

**Weight Stationary (WS)**



1. **Global Buffer**: Bộ nhớ đệm chính phía trên cùng

+ Lưu trữ dữ liệu activation và partial sums

+ Mũi tên xanh dương: Luồng dữ liệu activation đến các PE

+ Mũi tên đỏ: Luồng dữ liệu partial sum (Psum) quay trở lại global buffer

2. **PE Array (W0-W7)**: 8 phần tử xử lý

+ Mỗi PE chứa một phần tử trọng số cụ thể (W0, W1, W2, v.v.)

+ Mỗi PE có bộ tính toán tích-tổng (⊗⊕) để thực hiện nhân và cộng

3. **Weight**: Các trọng số được gán cố định cho mỗi PE (màu xanh lá)

***Nguyên lý hoạt động***

+ Mỗi PE giữ cố định một trọng số filter cụ thể (do đó có tên "Weight Stationary")

+ Các giá trị activation (input) được truyền qua mảng PE

+ Partial sums được tích lũy và di chuyển qua các PE, sau đó quay trở lại global buffer

***Ưu điểm:***

**1. Giảm thiểu năng lượng đọc trọng số:**

+ "Minimize weight read energy consumption": Giảm năng lượng cần thiết để đọc trọng số

+ "Maximize convolutional and filter reuse of weights": Tối đa hóa việc tái sử dụng trọng số trong các phép tính chập

**2. Phân phối dữ liệu:**

+ "Broadcast activations": Các giá trị activation được phát sóng đến nhiều PE cùng lúc

+ "Accumulate psums spatially": Các partial sum được tích lũy không gian (trên toàn mảng PE)

***Ứng dụng thực tế***

+ Phù hợp với các mạng có filter nhỏ được sử dụng nhiều lần

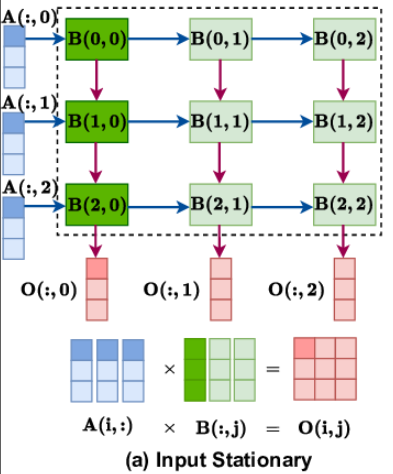
+ Hiệu quả trong các bộ tăng tốc có bộ nhớ on-chip giới hạn

+ Thường được sử dụng trong các kiến trúc TPU (Tensor Processing Unit) và FPGA

Weight Stationary là một chiến lược quan trọng trong thiết kế các bộ tăng tốc neural network hiệu quả về năng lượng, đặc biệt trong các ứng dụng có giới hạn về băng thông bộ nhớ dành cho việc truy cập trọng số.

***Input Stationary (IS)***

Input Stationary là chiến lược dataflow trong kiến trúc tăng tốc mạng nơ-ron tích chập, trong đó dữ liệu đầu vào (input activations) được giữ cố định và tái sử dụng, trong khi weights và partial sums có thể di chuyển.



**Đặc điểm chính:**

+ Giữ đầu vào cố định thay vì đầu ra hoặc trọng số

+ Được sử dụng chủ yếu trong các CNN thưa (sparse CNN). Ví dụ: Parashar et al., SCNN, ISCA 2017

**Vì sao hiệu quả với CNN thưa?**

+ CNN thưa là khi nhiều trọng số có giá trị bằng 0

+ Lợi thế là đầu vào thường lớn hơn trọng số và cần nhiều bộ nhớ hơn -> Giảm số lần đọc từ bộ nhớ lớn hơn

+ Không được phân tích nhiều cho các mạng dày đặc (dense networks)

***Nguyên lý hoạt động***

+ Các đơn vị xử lý (PE) sẽ nạp và giữ các giá trị đầu vào cố định

+ Các trọng số được đưa vào các PE để tính toán

+ Partial sums được tạo ra và di chuyển qua các PE

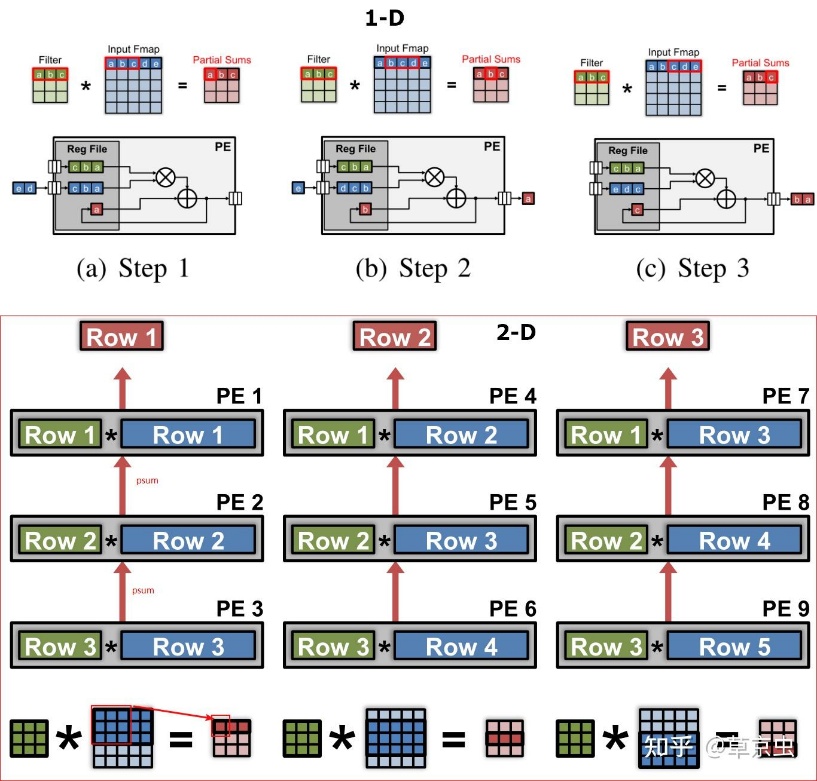
***Ưu điểm***

+ Giảm đáng kể năng lượng truy cập bộ nhớ khi đầu vào lớn hơn nhiều so với trọng số

+ Đặc biệt hiệu quả khi bộ nhớ đầu vào lớn hơn (off-chip) và tốn nhiều năng lượng truy cập hơn

**Luồng dữ liệu tiết kiệm năng lượng: Row Stationary (RS)**

Row Stationary (RS) là một chiến lược dataflow tiên tiến được thiết kế để tối ưu hóa hiệu suất năng lượng tổng thể trong các bộ tăng tốc neural network



***Đặc điểm chính:***

1. **Tối đa hóa việc tái sử dụng và tích lũy tại Register File (RF)**:

+ RS ưu tiên giữ dữ liệu trong các register file - bộ nhớ cấp thấp nhất với chi phí năng lượng truy cập thấp nhất

+ Thiết kế dữ liệu được lưu trữ và tái sử dụng nhiều lần tại register file

2. **Tối ưu hóa hiệu quả năng lượng tổng thể**:

+ Thay vì chỉ tối ưu cho một loại dữ liệu cụ thể (như Weight Stationary hoặc Output Stationary), RS tối ưu hóa hiệu quả năng lượng tổng thể

+ Xem xét tổng chi phí năng lượng cho tất cả các loại dữ liệu (weights, inputs, partial sums)

***Nguyên lý hoạt động***

Row Stationary hoạt động bằng cách tổ chức dữ liệu theo các "hàng" (rows):

+ Giữ một hàng của filter weights cố định trong register file

+ Giữ một hàng của input feature map (ifmap) cố định trong register file

+ Tính toán các partial sums dọc theo một hàng của output feature map (ofmap)

**So sánh với các dataflow khác**

+ **Weight Stationary**: Tối ưu cho việc tái sử dụng weight, nhưng không tối ưu cho input hoặc partial sum

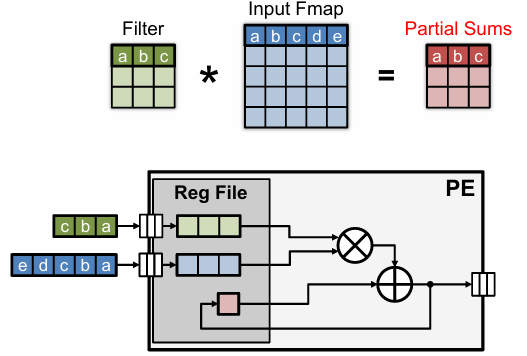
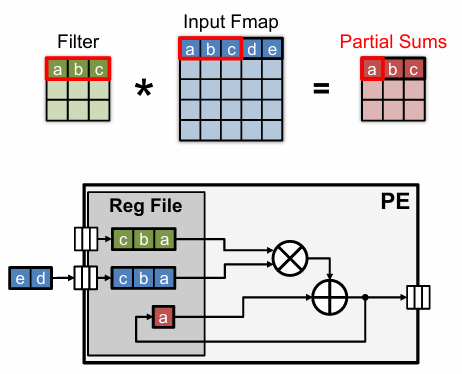
+ **Output Stationary**: Tối ưu cho việc tích lũy partial sum, nhưng có thể yêu cầu nhiều truy cập weight/input

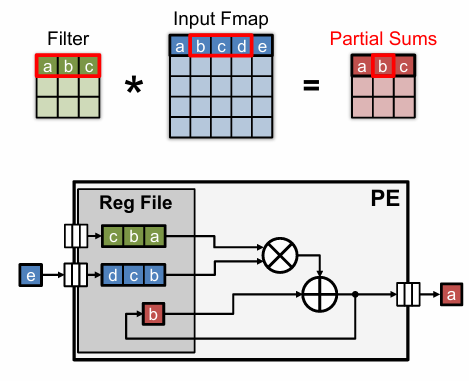
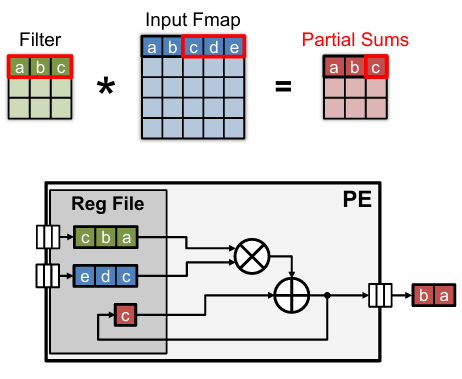
+ **Input Stationary**: Tối ưu cho việc tái sử dụng input, nhưng có thể không hiệu quả cho weight hoặc partial sum

+ **Row Stationary**: Cân bằng việc tái sử dụng tất cả các loại dữ liệu để tối ưu hóa hiệu suất năng lượng tổng thể

Row Stationary là một bước tiến so với các phương pháp dataflow truyền thống, nhằm giải quyết các hạn chế của từng phương pháp riêng lẻ và tối ưu hóa việc sử dụng năng lượng trên toàn bộ hệ thống tính toán.

Quá trình tính toán tích chập 1D theo phương pháp Row Stationary:

Các slide minh họa quá trình tính toán tích chập 1D theo phương pháp Row Stationary trong một đơn vị xử lý (PE). Tôi sẽ giải thích từng bước một:

## Cấu trúc chung trong các slide:

* **Phần trên**: Minh họa tổng quan về phép tích chập 1D giữa Filter và Input Fmap (Feature map)
* **Phần dưới**: Chi tiết hoạt động bên trong PE với Register File và quá trình tính toán

## Quá trình tính toán qua các bước (từ Image 1 đến Image 4):

### Image 1: Khởi tạo

* **Filter**: Hàng [a,b,c] được nạp vào register file
* **Input**: Chuỗi đầu vào [e,d,c,b,a] được chuẩn bị
* **Register File**: Lưu trữ filter [c,b,a] và đầu vào ban đầu
* **Thao tác**: Chuẩn bị cho phép tính tích chập

### Image 2: Bước tính toán đầu tiên

* **Register File**:
  + Lưu trữ filter [c,b,a]
  + Lưu trữ window đầu vào [c,b,a]
  + Lưu trữ partial sum [a]
* **Phép tính**:
  + a × a + b × b + c × c = partial sum đầu tiên (a)
  + Dữ liệu [e,d] đang chờ được đưa vào

### Image 3: Bước tính toán thứ hai

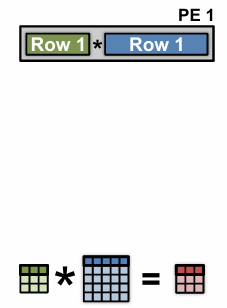
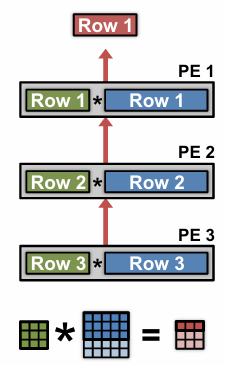
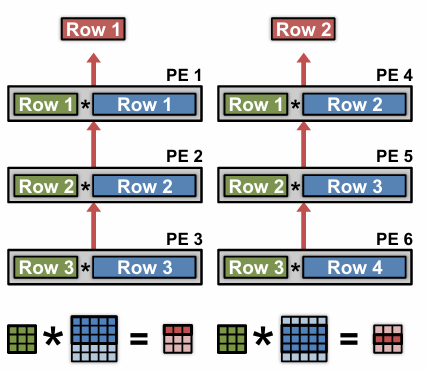
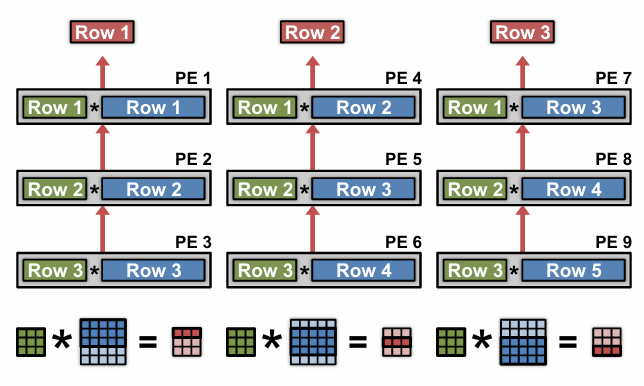
* **Register File**:
  + Filter vẫn giữ nguyên [c,b,a]
  + Window đầu vào được cập nhật thành [d,c,b]
  + Partial sum [b]
* **Phép tính**:
  + a × b + b × c + c × d = partial sum thứ hai (b)
  + Partial sum [a] đã được xuất ra

### Image 4: Bước tính toán thứ ba

* **Register File**:
  + Filter vẫn giữ nguyên [c,b,a] (row stationary)
  + Window đầu vào đã cập nhật thành [e,d,c]
  + Partial sum [c]
* **Phép tính**:
  + a × c + b × d + c × e = partial sum thứ ba (c)
  + Partial sums [b,a] đã được xuất ra

Đây là minh họa cụ thể về cách Row Stationary tối ưu hóa các hoạt động tính toán trong PE bằng cách giữ hàng filter cố định và xử lý hiệu quả dữ liệu đầu vào theo từng cửa sổ trượt, tạo ra partial sums tương ứng cho quá trình tích chập 1D.

Quá trình tính toán tích chập 2D trong mảng PE:

**Hình 1: Tích chập cơ bản trên một PE**

Hình 1 cho thấy một đơn vị xử lý đơn lẻ (PE1) đang thực hiện phép nhân giữa Row 1 của kernel (màu xanh lá) với Row 1 của dữ liệu đầu vào (màu xanh dương). Đây là phép toán cơ bản nhất trong tích chập.

**Hình 2: Tích chập theo cột dọc**

Hình 2 mở rộng thành 3 đơn vị xử lý (PE1, PE2, PE3) hoạt động song song:

* PE1: Row 1 (kernel) \* Row 1 (input)
* PE2: Row 2 (kernel) \* Row 2 (input)
* PE3: Row 3 (kernel) \* Row 3 (input)

Sau đó, kết quả từ các PE này được tổng hợp để tạo thành một phần tử đầu ra ở Row 1 của ma trận kết quả. Mô hình này tận dụng tính song song để tính toán nhiều phần tử cùng lúc.

**Hình 3: Tích chập theo hai cột**

Hình 3 mở rộng thêm với 6 đơn vị xử lý (PE1-PE6), được tổ chức thành hai cột:

* Cột 1 (PE1, PE2, PE3): Xử lý tích chập với Row 1 của input
* Cột 2 (PE4, PE5, PE6): Xử lý tích chập với Row 2 của input

Mỗi cột PE tính toán một phần tử đầu ra của ma trận kết quả.

**Hình 4: Tích chập đầy đủ**

Hình 4 mở rộng thành 9 đơn vị xử lý (PE1-PE9), được tổ chức thành ba cột:

* Cột 1 (PE1, PE2, PE3): Xử lý tích chập với Row 1 của input
* Cột 2 (PE4, PE5, PE6): Xử lý tích chập với Row 2 của input
* Cột 3 (PE7, PE8, PE9): Xử lý tích chập với Row 3 của input

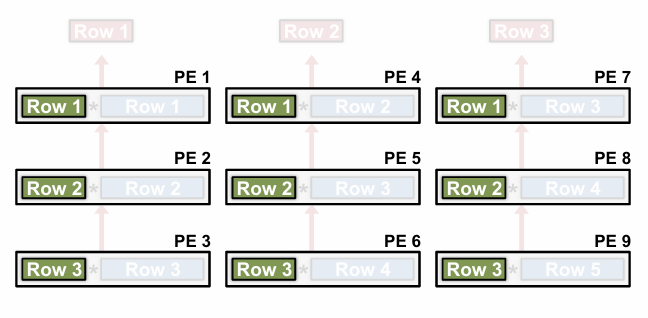
**Ý nghĩa trong Dataflow của Machine Learning**

Cách tổ chức này minh họa một kiến trúc dataflow quan trọng trong xử lý học máy:

1. **Song song hóa**: Tính toán được phân tán trên nhiều đơn vị xử lý, giúp tăng tốc độ xử lý đáng kể.
2. **Tái sử dụng dữ liệu**: Mỗi hàng dữ liệu được sử dụng nhiều lần với các hàng kernel khác nhau, tối ưu hóa việc truy cập bộ nhớ.
3. **Mô hình dòng chảy dữ liệu (Dataflow)**:
4. **Tăng dần quy mô**: Các hình ảnh cho thấy khả năng mở rộng từ 1 PE lên 3, 6, và 9 PE, minh họa tính linh hoạt trong thiết kế hệ thống.

Kiến trúc này đặc biệt hiệu quả cho các ứng dụng học sâu như CNN, nơi cần thực hiện hàng triệu phép tích chập. Bằng cách tổ chức tính toán theo hình thức mảng PE, các bộ tăng tốc AI hiện đại có thể đạt được hiệu suất cao hơn nhiều so với các kiến trúc truyền thống.

Tái sử dụng dữ liệu trong tính toán tích chập



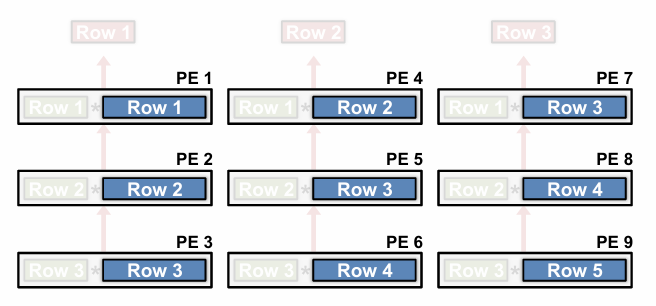
Minh họa việc tái sử dụng hàng bộ lọc (Filter rows) theo chiều ngang giữa các PE:

+ Mỗi hàng của bộ lọc (Row 1, Row 2, Row 3 màu xanh lá) được sử dụng đồng thời trên nhiều PE trong cùng một hàng

+ Ví dụ: Row 1 của bộ lọc được sử dụng đồng thời trên PE1, PE4, và PE7

+ Tương tự, Row 2 được sử dụng trên PE2, PE5, và PE8

Lợi ích: Giảm đáng kể việc truy cập bộ nhớ, vì cùng một dữ liệu bộ lọc được đọc một lần và phân phối cho nhiều đơn vị xử lý.



Cách tái sử dụng hàng đặc trưng đầu vào (Fmap rows - Feature map rows) theo đường chéo:

+ Hàng dữ liệu đầu vào (màu xanh dương) được sắp xếp theo mô hình đường chéo

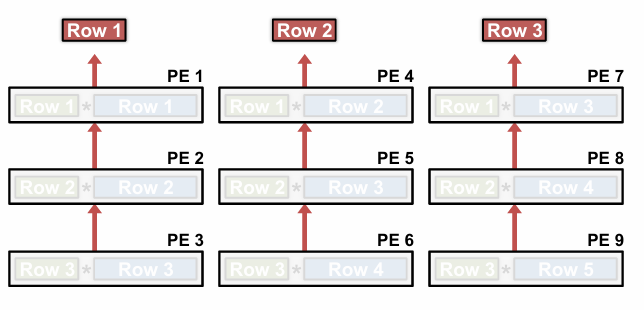
+ Ví dụ:

Row 1 → PE1, Row 2 → PE4, Row 3 → PE7

Row 2 → PE2, Row 3 → PE5, Row 4 → PE8

Row 3 → PE3, Row 4 → PE6, Row 5 → PE9

Lợi ích: Mô hình này tận dụng tính địa phương trong tích chập - các phần tử đầu ra liền kề cần truy cập các hàng đầu vào chồng lấn. Mô hình đường chéo này đảm bảo rằng dữ liệu đầu vào di chuyển hiệu quả qua mảng PE.



Minh họa việc tích lũy các tổng một phần (partial sums) theo chiều dọc:

+ Mỗi cột PE (PE1-PE3, PE4-PE6, PE7-PE9) đóng góp vào tính toán một phần tử đầu ra

+ Các kết quả trung gian (tổng một phần) từ mỗi PE trong cột được tích lũy theo chiều dọc

+ Mỗi cột PE cuối cùng tạo ra một phần tử trong hàng đầu ra (Row 1, Row 2, Row 3)

Lợi ích: Thiết kế này tối ưu hóa lưu lượng dữ liệu, cho phép các tổng một phần "chảy" tự nhiên theo chiều dọc thay vì phải di chuyển dữ liệu một cách không hiệu quả qua mảng.

**Tổng hợp: Tối ưu hóa dataflow trong tính toán tích chập**

Khi kết hợp cả ba chiến lược, bạn có một mô hình dataflow có hiệu quả cao:

1. **Tái sử dụng theo chiều ngang**: Giảm lượng truy cập bộ lọc
2. **Tái sử dụng theo đường chéo**: Tối ưu hóa việc di chuyển dữ liệu đầu vào
3. **Tích lũy theo chiều dọc**: Giảm thiểu di chuyển dữ liệu trung gian