TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HCM
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

ĐỒ ÁN THIẾT KẾ
KỸ THUẬT MÁY TÍNH

Thiết kế SoC RISC-V tích hợp EdgeAI cho ứng dụng IoT

Học kỳ 251

GVHD: PGS. TS. Trần Ngọc Thịnh
ThS. Huỳnh Phúc Nghị

STT	Họ và tên	MSSV	Ghi chú
1	Lâm Nữ Uyển Nhi	2212429	
2	Vũ Đức Lâm	2211824	

TP. Hồ Chí Minh, Tháng 12/2025

Mục lục

Danh mục Ký hiệu và Chữ viết tắt **vi**

1 Giới thiệu đề tài **1**

1.1 Tổng quan về đề tài 1

1.2 Mục tiêu và Nhiệm vụ nghiên cứu 2

1.3 Phạm vi đề tài 2

1.4 Phân chia công việc 2

1.5 Cấu trúc báo cáo 2

2 Cơ sở lý thuyết **3**

2.1 Tổng quan về Mạng nơ-ron tích chập (CNN) 3

2.2 Kỹ thuật thiết kế bộ tăng tốc phần cứng (AI Accelerator) 3

2.3 Kiến trúc System-on-Chip 3

3 Phân tích yêu cầu và Kiến trúc tổng quan **4**

3.1 Phân tích yêu cầu thiết kế 4

3.2 Kiến trúc hệ thống tổng quan 4

3.3 Đặc tả giao diện kết nối 4

4 Thiết kế kiến trúc Accelerator và Chiến lược dòng dữ liệu **5**

4.1 Phân tích toán học của phép tính tích chập 5

4.1.1 Standard Convolution (Tích chập tiêu chuẩn) . . 6

4.1.1.1 Công thức toán học tổng quát 6

4.1.1.2 Cấu trúc vòng lặp (Loop Nest) 6
4.1.2 Depthwise Separable Convolution 7
4.1.2.1 Depthwise Convolution (DW) 8
4.1.2.2 Pointwise Convolution (PW) 9
4.1.3 Yêu cầu đối với Kiến trúc thống nhất (Unified Architecture) 9
4.1.4 Kỹ thuật Gập Batch Normalization (BN Folding) 9
4.1.4.1 Công thức biến đổi trọng số 10
4.2 Chiến lược phân mảnh và Dòng dữ liệu đề xuất 11
4.2.1 Định nghĩa khái niệm "Tile" (Mảnh dữ liệu) . . . 11
4.2.2 Phương pháp Phân mảnh không gian dữ liệu (Space Partitioning) 12
4.2.2.1 Công thức chia khối (Block Calculation) 12
4.2.3 Thuật toán Điều phối Pass (Pass Scheduling) . . 13
4.2.3.1 Thuật toán cho Standard Convolution . 13
4.2.3.2 Thuật toán cho Depthwise Convolution . 14
4.2.4 Phân tích vấn đề tại biên và Dữ liệu dôi ra 15
4.2.4.1 Cơ sở hình thành Dữ liệu dôi ra 16
4.2.5 Cơ chế Ping-Pong Buffer và Logic xử lý hàng hợp lệ 17
4.2.6 Thuật toán Điều phối và Xoay vòng bộ nhớ . . . 18
4.3 Thiết kế kiến trúc vi mô (Micro-architecture) 21
4.3.1 Sơ đồ khối tổng quát hệ thống 21
4.3.2 Tổ chức Mảng tính toán (Processing Hierarchy) . 23
4.3.2.1 Mảng xử lý (Process Array - PA) 23
4.3.2.2 Đơn vị xử lý (Process Unit - PU) 23
4.3.2.3 Phần tử xử lý (Process Element - PE) . 24
4.3.3 Đánh giá thời gian thực thi (Performance Estimation) 25
4.3.3.1 Thời gian xử lý một Pass cơ sở (T_{pass}) . 25

4.3.3.2 Tổng thời gian thực thi (T_{total}) 26

5 Hiện thực nền tảng SoC **27**

5.1 Môi trường và Công cụ hiện thực 27

5.2 Cấu hình hệ thống xử lý (Processing System) 27

5.3 Thiết kế hệ thống kết nối (Interconnect Subsystem) . . . 27

5.4 Tích hợp và Kiểm thử nền tảng cơ sở 27

6 Đánh giá hiệu năng lý thuyết **28**

6.1 Phương pháp đánh giá: Mô hình Roofline 28

6.2 Ước lượng độ trễ và Tài nguyên 28

6.3 So sánh với các nghiên cứu liên quan 28

7 Kế hoạch phát triển **29**

7.1 Đánh giá mức độ hoàn thành Giai đoạn 1 29

7.2 Kế hoạch thực hiện Giai đoạn 2 29

7.3 Tiến độ dự kiến 29

Danh sách hình vẽ

Figure 4.1 Minh họa chiến lược phân chia Pass cho hai loại tích chập với $T_h = 11$. (a) Standard Convolution chia thành 2 phần theo chiều dọc và tích lũy theo chiều sâu. (b) Depthwise Convolution xử lý độc lập từng nhóm kênh và chia 2 phần theo chiều dọc. 15

Figure 4.2 Sơ đồ minh họa quá trình tính toán tích chập và sự hình thành dữ liệu dôi ra (Residual Data) trong một pass với tile đầu vào $T_h = 4$ và bộ lọc kích thước 3×3 trong trường hợp số kênh của ifmap feature là 1. 16

Figure 4.3 Sơ đồ luồng dữ liệu minh họa cơ chế Ping-Pong Buffer dùng để quản lý vùng dữ liệu dôi ra (Residual Data). Hệ thống luân phiên vai trò của Buffer A và B để đảm bảo tính liên tục của phép tính biên mà không cần nạp lại dữ liệu đầu vào. 20

Figure 4.4 Sơ đồ khối tổng quát kiến trúc Beta Accelerator 21

Figure 4.5 Kiến trúc bên trong khối Process Array (PA) 23

Figure 4.6 Kiến trúc khối Process Unit (PU) với các PE hoạt động song song 24

Figure 4.7 Cấu trúc bên trong một Process Element (PE) 25

Danh sách bảng biểu

Danh mục Ký hiệu và Chữ viết tắt

Ký hiệu	**Ý nghĩa**
N	Kích thước lô (Batch size)
C	Số lượng kênh đầu vào (Input Channels)
M	Số lượng kênh đầu ra (Output Channels/Filters)
H_{in}, W_{in}	Chiều cao và chiều rộng của đặc trưng đầu vào (Input Feature Map)
H_{out}, W_{out}	Chiều cao và chiều rộng của đặc trưng đầu ra (Output Feature Map)
R, S	Chiều cao và chiều rộng của bộ lọc (Kernel Height, Kernel Width)
U	Bước trượt (Stride)
P	Kích thước vùng đệm (Padding)
I	Tensor dữ liệu đầu vào
O	Tensor dữ liệu đầu ra
W	Tensor trọng số (Weights)
B	Vector hệ số chệch (Bias)
O_{dw}	Đầu ra của lớp Depthwise Convolution
O_{pw}	Đầu ra của lớp Pointwise Convolution

Ký hiệu	**Ý nghĩa**
μ	Giá trị trung bình (Mean) trong Batch Normalization
σ	Phương sai (Variance) trong Batch Normalization
γ	Tham số tỉ lệ (Scale factor)
β	Tham số dịch chuyển (Shift factor)
ϵ	Hằng số Epsilon

Viết tắt	**Ý nghĩa**
AI	Trí tuệ nhân tạo (Artificial Intelligence)
SoC	Hệ thống trên chip (System-on-Chip)
FPGA	Mảng cổng lập trình được dạng trường (Field-Programmable Gate Array)
CNN	Mạng nơ-ron tích chập (Convolutional Neural Network)
DNN	Mạng nơ-ron sâu (Deep Neural Network)
RTL	Mức chuyển giao thanh ghi (Register Transfer Level)
IP	Sở hữu trí tuệ (Intellectual Property - Khối thiết kế phần cứng)
PE	Phần tử xử lý (Processing Element)
MAC	Phép tính Nhân-Cộng tích lũy (Multiply-Accumulate)
DMA	Truy cập bộ nhớ trực tiếp (Direct Memory Access)
AXI-Lite	Giao diện mở rộng nâng cao rút gọn (Advanced eXtensible Interface Lite)
AXI-Stream	Giao diện luồng dữ liệu mở rộng nâng cao (Advanced eXtensible Interface Stream)
BRAM	Block RAM (Bộ nhớ nội trên FPGA)
DSP	Digital Signal Processing (Khối xử lý tín hiệu số trên FPGA)
LUT	Bảng tra (Look-Up Table)
FF	Flip-Flop
OSPI	Giao diện ngoại vi nối tiếp 8 kênh (Octal Serial Peripheral Interface)

Ký hiệu	**Ý nghĩa**
SPI	Giao diện ngoại vi nối tiếp (Serial Peripheral Interface)
UART	Bộ truyền nhận dữ liệu nối tiếp bất đồng bộ (Universal Asynchronous Receiver-Transmitter)
I2C	Giao thức giao tiếp giữa các vi mạch (Inter-Integrated Circuit)
DVP	Cổng dữ liệu hình ảnh kỹ thuật số (Digital Video Port)
GPIO	Cổng vào/ra đa dụng (General Purpose Input/Output)

Chương 1

Giới thiệu đề tài

Chương này trình bày tổng quan về bối cảnh nghiên cứu, xác định mục tiêu cụ thể, phạm vi thực hiện.

1.1 Tổng quan về đề tài

Tên đề tài: Thiết kế SoC RISC-V tích hợp EdgeAI cho ứng dụng IoT.

Đề tài tập trung vào việc thiết kế và phát triển một hệ thống SoC dựa trên vi xử lý RISC-V tích hợp phần tăng tốc EdgeAI, nhằm xử lý các tác vụ trí tuệ nhân tạo tại biên. Hệ thống sẽ được triển khai thử nghiệm trên nền tảng FPGA, với kiến trúc hướng tới khả năng chuyển đổi sang thiết kế ASIC. Đề tài cũng bao gồm thử nghiệm hiệu suất hệ thống với một tập dữ liệu cố định và triển khai một số ứng dụng IoT làm case study nhằm đánh giá tính khả thi và hiệu quả của hệ thống trong môi trường thực tế.

Hệ thống sẽ bao gồm: Vi xử lý RISC-V, CNN accelerator, Bus giao tiếp nội bộ (AXI-Lite, AXI-Stream), bộ điều khiển DMA và giao tiếp I/O với ngoại vi (OSPI, SPI, UART, I2C, DVP, GPIO,...).

1.2 Mục tiêu và Nhiệm vụ nghiên cứu

Mục tiêu chính của đề tài là nghiên cứu, thiết kế và hiện thực một hệ thống trên chip (SoC) hoàn chỉnh tích hợp lõi vi xử lý RISC-V và bộ tăng tốc phần cứng (Hardware Accelerator) cho các tác vụ trí tuệ nhân tạo tại biên (EdgeAI). Cụ thể, đề tài hướng tới các mục tiêu sau:

- **Về kiến trúc hệ thống:** Xây dựng kiến trúc SoC tối ưu năng lượng, sử dụng chuẩn giao tiếp AXI để kết nối giữa vi xử lý trung tâm và khối tăng tốc tính toán.
- **Về xử lý AI:** Thiết kế khối Accelerator chuyên dụng hỗ trợ các phép toán trọng yếu của mạng nơ-ron tích chập (CNN) như Tích chập (Convolution), Pooling và hàm kích hoạt (Activation), nhằm giảm tải cho CPU và tăng tốc độ xử lý thực tế.
- **Về ứng dụng thực tế:** Tích hợp đầy đủ các giao tiếp ngoại vi (Camera DVP, UART, SPI) để xây dựng một ứng dụng IoT trọn vẹn (ví dụ: nhận diện vật thể hoặc phân loại ảnh) chạy trực tiếp trên nền tảng FPGA.
- **Về quy trình thiết kế:** Làm chủ quy trình thiết kế từ mức RTL (Verilog) đến mô phỏng (Simulation), tổng hợp (Synthesis) và kiểm tra trên phần cứng thực (FPGA Prototyping).

1.3 Phạm vi đề tài

1.4 Phân chia công việc

1.5 Cấu trúc báo cáo

Chương 2

Cơ sở lý thuyết

Chương này cung cấp các kiến thức nền tảng về Mạng nơ-ron tích chập (CNN), các kỹ thuật thiết kế phần cứng cho AI và kiến trúc System-on-Chip trên FPGA.

2.1 Tổng quan về Mạng nơ-ron tích chập (CNN)

2.2 Kỹ thuật thiết kế bộ tăng tốc phần cứng (AI Accelerator)

2.3 Kiến trúc System-on-Chip

Chương 3

Phân tích yêu cầu và Kiến trúc tổng quan

Chương này phân tích các ràng buộc thiết kế từ đó đề xuất kiến trúc tổng thể của hệ thống SoC tích hợp AI Accelerator.

3.1 Phân tích yêu cầu thiết kế

3.2 Kiến trúc hệ thống tổng quan

3.3 Đặc tả giao diện kết nối

Chương 4

Thiết kế kiến trúc Accelerator và Chiến lược dòng dữ liệu

Chương này trình bày chi tiết thiết kế của lõi IP Accelerator, bao gồm phân tích toán học, chiến lược tối ưu dòng dữ liệu và kiến trúc vi mô.

4.1 Phân tích toán học của phép tính tích chập

Để đảm bảo tính linh hoạt cho kiến trúc phần cứng, giúp hệ thống có khả năng hỗ trợ đa dạng các mô hình mạng nơ-ron từ kinh điển (như VGG16) đến các mô hình tối ưu cho thiết bị biên (như MobileNet), nhóm thực hiện đề tài đã tập trung phân tích đặc tả toán học của hai loại phép tính cốt lõi: **Standard Convolution** và **Depthwise Separable Convolution**. Việc hiểu rõ bản chất toán học và cấu trúc dữ liệu của các phép tính này (bao gồm cả cơ chế xử lý biên - Padding) là cơ sở quan trọng để chúng tôi thiết kế nên một kiến trúc thống nhất (Unified Architecture).

4.1.1 Standard Convolution (Tích chập tiêu chuẩn)

Đây là phép tính nền tảng trong hầu hết các mạng CNN truyền thống. Về mặt toán học, tích chập tiêu chuẩn thực hiện việc trượt bộ lọc (filter) trên không gian đầu vào (H, W), đồng thời tích lũy giá trị qua toàn bộ chiều sâu của kênh (Channels).

4.1.1.1 Công thức toán học tổng quát

Xét một lớp tích chập với đầu vào I có kích thước $C \times H_{in} \times W_{in}$ và bộ trọng số W có kích thước $M \times C \times R \times S$. Tham số Padding (P) được sử dụng để giữ nguyên kích thước không gian hoặc kiểm soát việc giảm kích thước. Giá trị đầu ra O tại kênh m, vị trí (h, w) được xác định bởi:

$$O[m][h][w] = B[m] + \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} I[c][h \cdot U + r - P][w \cdot U + s - P] \times W[m][c][r][s] \tag{4.1}$$

Trong đó:

- U: Bước trượt (Stride).
- P: Số lượng điểm ảnh đệm thêm vào mỗi cạnh (Padding).
- Điều kiện biên: Nếu chỉ số truy cập I nằm ngoài phạm vi $[0, H_{in} - 1]$ hoặc $[0, W_{in} - 1]$, giá trị trả về là 0 (Zero-padding).

4.1.1.2 Cấu trúc vòng lặp (Loop Nest)

Với giả thiết kích thước batch $N = 1$, chúng tôi mô hình hóa phép tính này dưới dạng 6 vòng lặp lồng nhau. Việc xử lý Padding thường được thực hiện bằng phần cứng chuyên dụng (Padding Logic) để tránh truy cập bộ nhớ ngoài vùng cho phép.

Algorithm 1: Standard Convolution (Standard Conv2D)

Input: $I[C][H_{in}][W_{in}]$, $W[M][C][R][S]$, Padding P, Stride U

Output: $O[M][H_{out}][W_{out}]$

for $m = 0$ ***to*** $M - 1$ **do**
  for $c = 0$ ***to*** $C - 1$ **do**
    for $h = 0$ ***to*** $H_{out} - 1$ **do**
      for $w = 0$ ***to*** $W_{out} - 1$ **do**
        for $r = 0$ ***to*** $R - 1$ **do**
          for $s = 0$ ***to*** $S - 1$ **do**
            $h_{in} = h \cdot U + r - P$
            $w_{in} = w \cdot U + s - P$
            if $h_{in} \geq 0 \wedge h_{in} < H_{in} \wedge w_{in} \geq 0 \wedge w_{in} < W_{in}$ **then**
              $val = I[c][h_{in}][w_{in}]$
            else
              $val = 0$ `/* Zero Padding */`
            end
            $O[m][h][w] \leftarrow O[m][h][w] + val \times W[m][c][r][s]$
          end
        end
      end
    end
  end
end

4.1.2 Depthwise Separable Convolution

Để giảm chi phí tính toán cho các thiết bị biên, các mô hình như MobileNet sử dụng kỹ thuật **Depthwise Separable Convolution**, tách phép chập chuẩn thành hai bước: **Depthwise (DW)** và **Pointwise (PW)**.

4.1.2.1 Depthwise Convolution (DW)

Phép tính này áp dụng bộ lọc riêng cho từng kênh đầu vào. Công thức tính toán bao gồm tham số Padding như sau:

$$O_{dw}[c][h][w] = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} I[c][h \cdot U + r - P][w \cdot U + s - P] \times W_{dw}[c][r][s] \quad (4.2)$$

Nhận xét: Việc xử lý Padding trong Depthwise cũng tương tự như Standard Conv, tuy nhiên do tính độc lập giữa các kênh, bộ điều khiển (Controller) cần đảm bảo logic Padding hoạt động chính xác cho từng luồng tính toán song song.

Algorithm 2: Depthwise Convolution (với Padding)

for $c = 0$ ***to*** $C - 1$ `/* Parallel Channels */`
do
  for $h = 0$ ***to*** $H_{out} - 1$ **do**
    for $w = 0$ ***to*** $W_{out} - 1$ **do**
      for $r = 0$ ***to*** $R - 1$ **do**
        for $s = 0$ ***to*** $S - 1$ **do**
          $h_{in} = h \cdot U + r - P$
          $w_{in} = w \cdot U + s - P$
          if $h_{in} \in [0, H_{in}) \wedge w_{in} \in [0, W_{in})$ **then**
            $O_{dw}[c][h][w] += I[c][h_{in}][w_{in}] \times W_{dw}[c][r][s]$
          end
        end
      end
    end
  end
end

4.1.2.2 Pointwise Convolution (PW)

Pointwise Convolution là tích chập chuẩn với kernel 1×1. Do kích thước kernel là 1×1, tham số Padding thường được đặt bằng 0 ($P = 0$) và Stride $U = 1$ để giữ nguyên kích thước không gian (H, W), chỉ thay đổi số kênh từ C sang M.

$$O_{pw}[m][h][w] = \sum_{c=0}^{C-1} I[c][h][w] \times W_{pw}[m][c] \tag{4.3}$$

4.1.3 Yêu cầu đối với Kiến trúc thống nhất (Unified Architecture)

Từ các phân tích trên, nhóm nhận thấy rằng để bộ tăng tốc hoạt động hiệu quả cho cả hai trường hợp, kiến trúc phần cứng cần giải quyết được bài toán "kép":

1. **Cơ chế xử lý Padding động:** Phần cứng cần có khối logic để tự động chèn giá trị 0 khi chỉ số tính toán $(h \cdot U + r - P)$ bị âm hoặc vượt quá kích thước ảnh, thay vì phải tốn tài nguyên bộ nhớ để lưu trữ các viền số 0 thực tế.

2. **Tính linh hoạt của Mảng PE:** Các đơn vị tính toán cần có khả năng chuyển đổi chế độ giữa tích lũy theo không gian (Standard/Pointwise) và tính toán độc lập theo kênh (Depthwise).

4.1.4 Kỹ thuật Gập Batch Normalization (BN Folding)

Trong các mạng CNN hiện đại như MobileNet, lớp Batch Normalization (BN) thường được đặt ngay sau lớp Convolution để chuẩn hóa phân phối dữ liệu, giúp mạng hội tụ nhanh hơn. Công thức tính toán của lớp BN trong quá trình suy luận (Inference) cho một kênh m là:

$$y = \frac{x - \mu_m}{\sqrt{\sigma_m^2 + \epsilon}} \cdot \gamma_m + \beta_m \tag{4.4}$$

Trong đó:

- x: Giá trị đầu ra từ lớp Convolution (trước khi qua hàm kích hoạt).
- μ_m, σ_m: Giá trị trung bình (mean) và phương sai (variance) động (running statistics) của kênh m.
- γ_m, β_m: Tham số tỉ lệ (scale) và dịch chuyển (shift) được học trong quá trình huấn luyện.
- ϵ: Hằng số nhỏ để tránh chia cho 0.

Việc thực hiện trực tiếp công thức này trên phần cứng rất tốn kém do yêu cầu các phép toán phức tạp như căn bậc hai và phép chia. Tuy nhiên, do tại thời điểm suy luận, các tham số $\mu, \sigma, \gamma, \beta$ đều là hằng số, chúng tôi áp dụng kỹ thuật **BN Folding** để gộp toàn bộ phép tính BN vào trong trọng số (W) và bias (B) của lớp Convolution đi trước nó.

4.1.4.1 Công thức biến đổi trọng số

Giả sử đầu ra của lớp Convolution là $x = W_{orig} \cdot Input + B_{orig}$. Khi thay vào công thức BN, ta có:

$$y = \frac{(W_{orig} \cdot Input + B_{orig}) - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta \tag{4.5}$$

Phương trình trên có thể được viết lại dưới dạng một phép Convolution mới với trọng số W' và bias B':

$$y = W' \cdot Input + B' \tag{4.6}$$

Trong đó, các tham số mới được tính toán trước (offline) bởi phần mềm

(driver) trước khi nạp xuống phần cứng:

$$W'[m][c][r][s] = W_{orig}[m][c][r][s] \cdot \frac{\gamma_m}{\sqrt{\sigma_m^2 + \epsilon}} \tag{4.7}$$

$$B'[m] = \left(B_{orig}[m] - \mu_m\right) \cdot \frac{\gamma_m}{\sqrt{\sigma_m^2 + \epsilon}} + \beta_m \tag{4.8}$$

Kết luận thiết kế: Nhờ kỹ thuật BN Folding, kiến trúc phần cứng của chúng tôi **không cần** thiết kế khối chức năng riêng cho Batch Normalization. Accelerator chỉ cần thực hiện phép tính Convolution bình thường với bộ trọng số (W', B') đã được tinh chỉnh, giúp tiết kiệm đáng kể tài nguyên DSP và giảm độ trễ xử lý.

4.2 Chiến lược phân mảnh và Dòng dữ liệu đề xuất

Để xử lý các Feature Map kích thước lớn trên tài nguyên phần cứng giới hạn, chúng tôi áp dụng chiến lược phân mảnh dữ liệu (Tiling) không chồng lấn. Mục này sẽ trình bày chi tiết cách thức chia nhỏ không gian dữ liệu và thuật toán điều phối các bước tính toán (Passes).

4.2.1 Định nghĩa khái niệm "Tile" (Mảnh dữ liệu)

Trong kiến trúc này, một "Tile" được định nghĩa là một phần nhỏ của khối dữ liệu gốc với kích thước được tối ưu hóa cho dung lượng bộ nhớ on-chip. Hệ thống xử lý ba loại Tile chính tương ứng với ba luồng dữ liệu. Đầu tiên, Input Tile (Mảnh đầu vào) là khối dữ liệu được cắt ra từ Input Feature Map gốc với kích thước $T_c \times T_h \times W$. Việc giới hạn số kênh nạp vào là T_c và chiều cao là T_h giúp dữ liệu vừa vặn với bộ nhớ đệm, trong khi chiều rộng W được giữ nguyên để tận dụng tính liên tục của dữ liệu trong bộ nhớ (Burst Read). Tiếp theo, Weight Tile (Mảnh trọng số) tập hợp các bộ lọc

cần thiết để xử lý cho Input Tile hiện tại, có kích thước $T_m \times T_c \times R \times S$. Cuối cùng, kết quả tính toán tương ứng tạo ra Output Tile (Mảnh đầu ra) với kích thước $T_m \times T_h \times W$.

4.2.2 Phương pháp Phân mảnh không gian dữ liệu (Space Partitioning)

Không gian tính toán của một lớp tích chập được định nghĩa bởi ba chiều chính: Chiều cao không gian (H), Chiều sâu kênh đầu vào (C), và Số lượng bộ lọc/kênh đầu ra (M). Chúng tôi chia nhỏ không gian này thành các khối (Block/Tile) độc lập dựa trên tham số phần cứng.

4.2.2.1 Công thức chia khối (Block Calculation)

Giả sử phần cứng có khả năng xử lý song song một khối dữ liệu kích thước $T_c \times T_h \times W$ và tạo ra T_m kênh đầu ra. Số lượng khối (Blocks) trên mỗi chiều được tính như sau:

- **Số khối theo chiều dọc (N_h):** Ảnh đầu vào chiều cao H được cắt thành N_h phần không chồng lấn.

$$N_h = \lceil \frac{H}{T_h} \rceil \tag{4.9}$$

  Ví dụ: Với $H = 21, T_h = 11$, ta có $N_h = 2$ khối (Khối 0: hàng 0-10; Khối 1: hàng 11-20).

- **Số khối kênh đầu vào (N_c):** Tổng C kênh được chia thành N_c nhóm.

$$N_c = \lceil \frac{C}{T_c} \rceil \tag{4.10}$$

- **Số khối kênh đầu ra (N_m):** Tổng M bộ lọc được chia thành N_m nhóm.

$$N_m = \lceil \frac{M}{T_m} \rceil \tag{4.11}$$

Một đơn vị xử lý cơ sở, gọi là **1 Pass**, chính là quá trình hệ thống xử lý hoàn tất cho một cặp Input Tile và Weight Tile để cập nhật giá trị cho một Output Tile.

4.2.3 Thuật toán Điều phối Pass (Pass Scheduling)

Trình tự thực hiện các Pass phụ thuộc vào loại tích chập (Standard hay Depthwise) để tối ưu hóa việc tái sử dụng dữ liệu biên (như đã phân tích ở mục Dữ liệu dôi ra).

4.2.3.1 Thuật toán cho Standard Convolution

Trong Standard Convolution, một điểm ảnh đầu ra cần tổng hợp dữ liệu từ **tất cả** các khối kênh đầu vào (N_c). Do đó, ta cần vòng lặp tích lũy (Reduction Loop) chạy qua N_c trước khi chuyển sang khối chiều cao khác.

Algorithm 3: Lịch trình Pass cho Standard Convolution

Input: N_m (Output Blocks), N_h (Height Blocks), N_c (Input Blocks)

for $m = 0$ ***to*** $N_m - 1$ **do**

1. Load Weights for Output Block m (Weight Stationary)

for $h = 0$ ***to*** $N_h - 1$ **do**

for $c = 0$ ***to*** $N_c - 1$ **do**

Pass (m, h, c)**:**

- Nạp Input Block (c, h) kích thước $T_c \times T_h$
- Tính toán với Weight Block (m, c)
- Cộng dồn kết quả vào Buffer hiện tại (A hoặc B)

end

2. Xử lý biên & Ghi Output:

- Sau khi cộng đủ N_c passes: Output Block (m, h) đã hoàn tất (Valid).
- Ghi phần Valid xuống DRAM.
- Hoán đổi Ping-Pong Buffer (để dùng phần Dôi ra cho $h+1$).

end

end

4.2.3.2 Thuật toán cho Depthwise Convolution

Trong Depthwise Convolution, kênh Input thứ i chỉ tính toán với kênh Filter thứ i. Do đó $N_c = N_m$ (số nhóm kênh Input bằng số nhóm kênh Output) và không có sự cộng dồn chéo giữa các nhóm. Vòng lặp tích lũy biến mất.

Algorithm 4: Lịch trình Pass cho Depthwise Convolution

Input: N_m (Channel Groups), N_h (Height Blocks)

for $g = 0$ ***to*** $N_m - 1$ **do**

1. Load Weights for Group g

for $h = 0$ ***to*** $N_h - 1$ **do**

Pass (g, h)**:**

- Nạp Input Block (g, h) kích thước $T_c \times T_h$

- Tính toán với Weight Block g

- Tạo ra ngay kết quả Output Block (g, h) (không cần cộng dồn)

2. Xử lý biên & Ghi Output:

- Ghi ngay phần Valid xuống DRAM.

- Hoán đổi Ping-Pong Buffer (lưu phần Dôi ra cho $h+1$).

end

end

Pass 0 --- row 0-10, channel 0-10, filter 0	Pass 1 --- row 0-10, channel 11-20, filter 0	Pass 2 --- row 11-20, channel 0-10, filter 0	Pass 3 --- row 11-20, channel 11-20, filter 0
Pass 4 --- row 0-10, channel 0-10, filter 1	Pass 5 --- row 0-10, channel 11-20, filter 1	Pass 6 --- row 11-20, channel 0-10, filter 1	Pass 7 --- row 11-20, channel 11-20, filter 1

(a) Standard Convolution ($H = 21, M = 2$)

Pass 0 --- row 0-10, channel 0-10, filter 0-10	Pass 1 --- row 11-20, channel 0-10, filter 0-10	Pass 2 --- row 0-10, channel 11-20, filter 11-20	Pass 3 --- row 11-20, channel 11-20, filter 11-20

(b) Depthwise Convolution ($H = 21, M = 21$)

Hình 4.1: Minh họa chiến lược phân chia Pass cho hai loại tích chập với $T_h = 11$. (a) Standard Convolution chia thành 2 phần theo chiều dọc và tích lũy theo chiều sâu. (b) Depthwise Convolution xử lý độc lập từng nhóm kênh và chia 2 phần theo chiều dọc.

4.2.4 Phân tích vấn đề tại biên và Dữ liệu dôi ra

Mặc dù chiến lược phân mảnh dữ liệu được áp dụng trên cả chiều kênh và chiều không gian, tác động của chúng lên luồng dữ liệu là khác nhau.

- Việc chia nhỏ chiều kênh (C, M) dẫn đến bài toán tích lũy tổng riêng

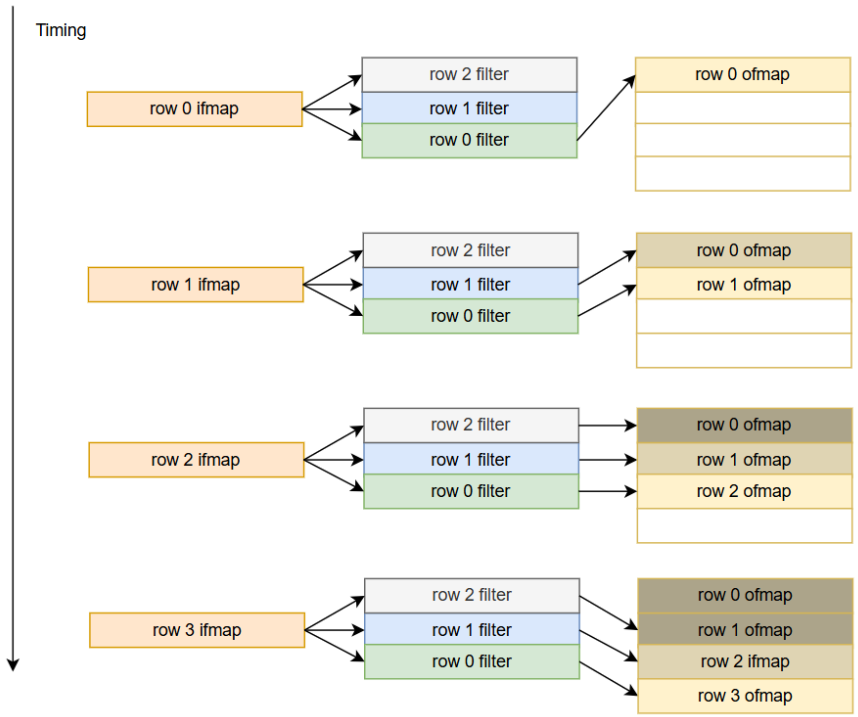
(Partial Sum Accumulation).

- Việc chia nhỏ chiều không gian (H) dẫn đến bài toán thiếu dữ liệu vùng lân cận cho cửa sổ trượt (Sliding Window Boundary).

Mục này tập trung phân tích vấn đề tại biên không gian, nguyên nhân chính dẫn đến sự cần thiết của cơ chế Ping-Pong Buffer đặc thù.

4.2.4.1 Cơ sở hình thành Dữ liệu dôi ra

Khi bộ lọc trượt theo chiều dọc, tại các hàng cuối cùng của một Tile không gian (gọi là Tile H_k), bộ lọc cần dữ liệu của các hàng tiếp theo (thuộc Tile H_{k+1}) để hoàn thành phép tính.



Hình 4.2: Sơ đồ minh họa quá trình tính toán tích chập và sự hình thành dữ liệu dôi ra (Residual Data) trong một pass với tile đầu vào $T_h = 4$ và bộ lọc kích thước 3×3 trong trường hợp số kênh của ifmap feature là 1.

Xét ví dụ cụ thể với Tile đầu vào có kích thước chiều cao $T_h = 4$ (các hàng 0, 1, 2, 3) và bộ lọc kích thước 3×3. Khi thực hiện tích chập:

- **Hàng 0, 1:** Có đầy đủ dữ liệu lân cận (trong phạm vi Tile hiện tại)

$\rightarrow$ Tạo ra kết quả hoàn chỉnh (*Valid Output*).

- **Hàng 2:** Cần dữ liệu hàng [2, 3, **4**]. Thiếu hàng 4 (thuộc Tile H_{k+1}) $\rightarrow$ Kết quả chưa hoàn thiện.
- **Hàng 3:** Cần dữ liệu hàng [3, **4**, **5**]. Thiếu hàng 4, 5 (thuộc Tile H_{k+1}) $\rightarrow$ Kết quả chưa hoàn thiện.

Các kết quả tại hàng 2 và 3 được gọi là **Dữ liệu dôi ra (Residual Data)**. Số lượng hàng dôi ra luôn là $R-1$. Để đảm bảo tính đúng đắn mà không cần nạp lại phần dữ liệu Input [2, 3] khi xử lý Tile H_{k+1}, hệ thống cần lưu trữ các giá trị dôi ra này và cộng dồn chúng với kết quả tính toán từ Tile tiếp theo.

4.2.5 Cơ chế Ping-Pong Buffer và Logic xử lý hàng hợp lệ

Để xử lý dữ liệu dôi ra (Residual Data) tại biên dưới của mỗi tile mà không cần nạp lại Input, hệ thống sử dụng hai bộ đệm đầu ra $Buffer_A$ và $Buffer_B$ hoạt động luân phiên.
Điểm quan trọng trong chiến lược này là số lượng hàng đầu ra hợp lệ (Valid Rows) sẽ khác nhau giữa Tile đầu tiên và các Tile tiếp theo:

- **Tile đầu tiên ($h = 0$):** Do không có dữ liệu tích lũy từ phía trên, bộ lọc trượt qua T_h hàng đầu vào chỉ tạo ra được $T_h - R + 1$ hàng đầu ra hoàn chỉnh. $R-1$ hàng cuối cùng là dữ liệu dôi ra.
- **Các Tile tiếp theo ($h > 0$):** Nhờ tận dụng $R-1$ hàng dôi ra từ bước trước (đã lưu trong Buffer), hệ thống sẽ hoàn thiện được các hàng này. Tổng số hàng hoàn chỉnh được ghi xuống DRAM trong bước này là đủ T_h hàng.

4.2.6 Thuật toán Điều phối và Xoay vòng bộ nhớ

Thuật toán 5 và 6 mô tả chi tiết quy trình quản lý bộ nhớ và luồng dữ liệu, minh họa rõ sự khác biệt khi xử lý Tile đầu tiên và các Tile sau.

Algorithm 5: Lịch trình Pass cho Standard Conv (Phần 1: Tích lũy)

Input: N_m, N_h, N_c

for $m = 0$ ***to*** $N_m - 1$ **do**

  1. Load Weights...

  for $h = 0$ ***to*** $N_h - 1$ **do**

    for $c = 0$ ***to*** $N_c - 1$ **do**

      Pass (m, h, c)**:** ...

      ... (Code phần tích lũy) ...

    end

    (Xem tiếp xử lý biên ở Thuật toán 6)

  end

end

Algorithm 6: Lịch trình Pass cho Standard Conv (Phần 2: Xử lý biên)

...Tiếp tục từ vòng lặp h của Thuật toán 5

foreach *Tile h đã hoàn tất tích lũy* **do**

  2. Xử lý biên & Ghi Output:

  - Kiểm tra điều kiện biên...

  - Ghi phần Valid xuống DRAM.

  - Hoán đổi Ping-Pong Buffer.

end

Giải thích cơ chế:

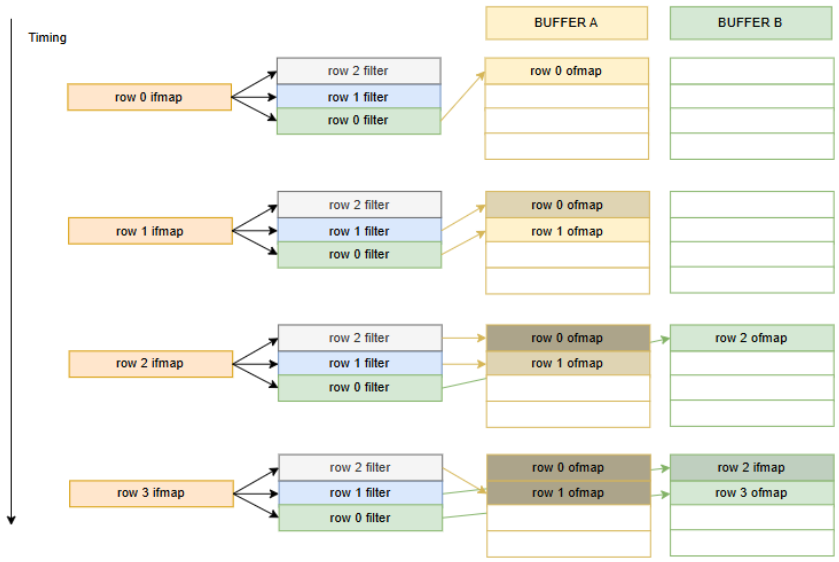
1. Tại vòng lặp h, Buf_{curr} đóng vai trò tích lũy kết quả chính, còn Buf_{next} đóng vai trò hứng các giá trị dôi ra (Residual) cho tương lai.

2. Khi $h = 0$: Chúng ta ghi các hàng dôi ra vào đầu Buf_{next}.

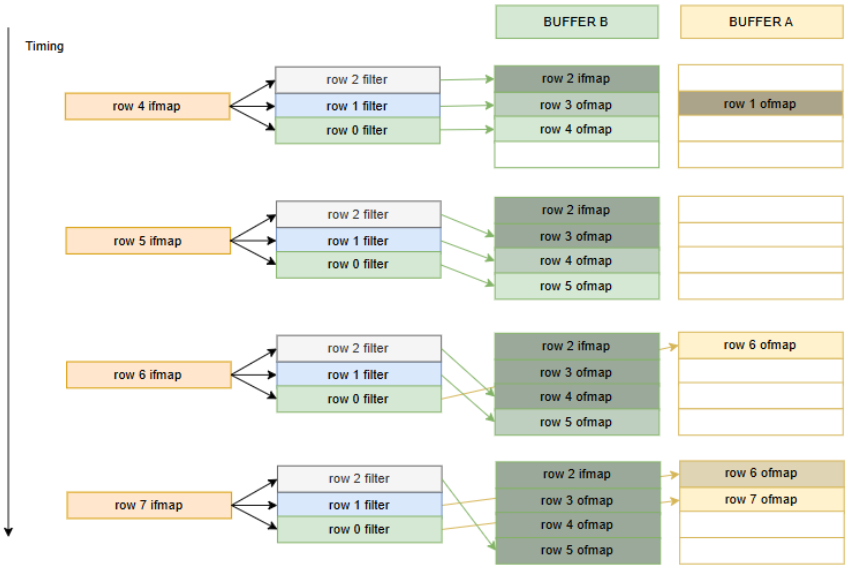
3. Khi chuyển sang $h = 1$: Ta thực hiện Swap. Lúc này Buf_{curr} (vốn

là Buf_{next} cũ) đã có sẵn dữ liệu dôi ra ở các hàng đầu. Việc tính toán tiếp tục cộng dồn vào đó, biến chúng thành kết quả hoàn chỉnh (Valid).

4. Quá trình ghi xuống DRAM (Drain) ở $h > 0$ sẽ ghi toàn bộ T_h hàng, bao gồm cả những hàng vừa được hoàn thiện từ dữ liệu dôi ra.



(a) Giai đoạn 1 (Xử lý Tile H_k): Buffer A tích lũy kết quả Valid, Buffer B lưu trữ dữ liệu Dôi ra (Residual).



(b) Giai đoạn 2 (Xử lý Tile H_{k+1}): Buffer B hoàn thiện kết quả biên (từ Residual cũ), Buffer A lưu trữ dữ liệu Dôi ra mới.

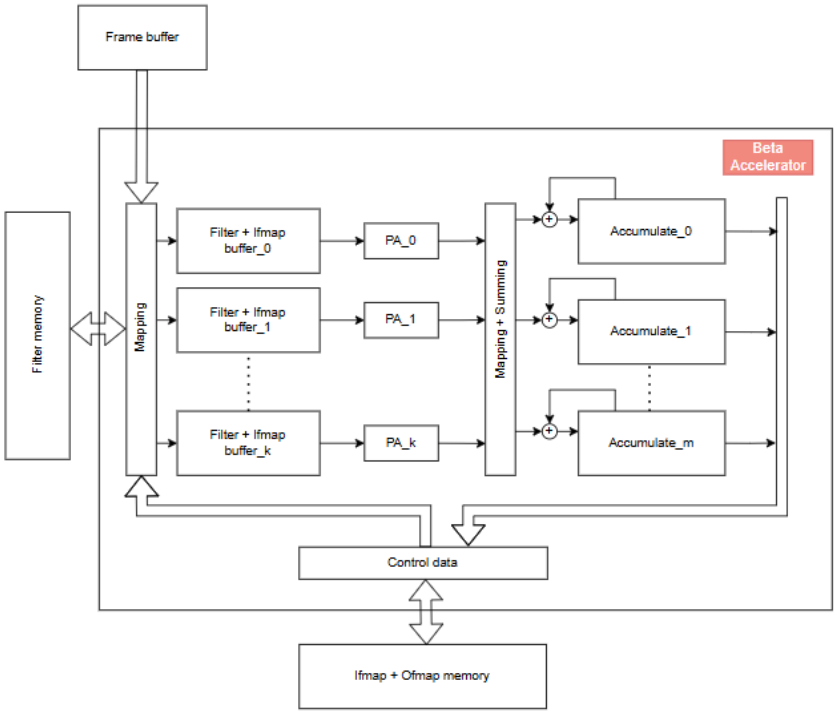
Hình 4.3: Sơ đồ luồng dữ liệu minh họa cơ chế Ping-Pong Buffer dùng để quản lý vùng dữ liệu dôi ra (Residual Data). Hệ thống luân phiên vai trò của Buffer A và B để đảm bảo tính liên tục của phép tính biên mà không cần nạp lại dữ liệu đầu vào.

4.3 Thiết kế kiến trúc vi mô (Micro-architecture)

Dựa trên chiến lược dòng dữ liệu đã phân tích, nhóm đề xuất kiến trúc phần cứng chuyên dụng mang tên **Beta Accelerator**. Kiến trúc này được thiết kế để tối ưu hóa khả năng tính toán song song ở mức bộ lọc (Filter parallelism) và mức kênh (Channel parallelism), đồng thời hỗ trợ cơ chế quản lý bộ nhớ Ping-Pong để che giấu độ trễ truy cập.

4.3.1 Sơ đồ khối tổng quát hệ thống

Sơ đồ tổng thể của Beta Accelerator được trình bày trong Hình 4.4. Hệ thống bao gồm các khối chức năng chính sau:



Hình 4.4: Sơ đồ khối tổng quát kiến trúc Beta Accelerator

- **Khối Control Data (Controller):** Đóng vai trò bộ điều khiển trung tâm và quản lý giao tiếp với bộ nhớ ngoài. Do hệ thống chỉ sử dụng một bus dữ liệu chung (shared data bus) cho cả luồng vào và luồng ra, khối này chịu trách nhiệm điều phối (arbitration) tài nguyên bus,

quyết định thời điểm thực hiện nạp dữ liệu đầu vào (Load IFM) hoặc ghi kết quả đầu ra (Store OFM) để tránh xung đột dữ liệu.

- **Khối Mapping (Dispatcher):** Chịu trách nhiệm phân phối dữ liệu IFM và trọng số (Weights) từ các bus dữ liệu chính tới các bộ nhớ đệm cục bộ của từng đơn vị tính toán, đảm bảo băng thông và tính đồng bộ.
- **Hệ thống Bộ đệm (Filter + Ifmap Buffer):** Được tổ chức theo cơ chế **Ping-Pong Buffer** (Double Buffering) để cho phép nạp dữ liệu cho Pass $k+1$ trong khi Pass k đang được tính toán.
  - Mỗi khối buffer lưu trữ một tile IFM kích thước $T_h \times W$ của 1 kênh.
  - Đồng thời lưu trữ bộ trọng số kích thước $S \times R \times T_m$ (trong đó S, R là kích thước filter, T_m là số filter tính song song).
- **Mảng xử lý (Process Array - PA):** Là trái tim tính toán của hệ thống, bao gồm T_c khối PA hoạt động song song. Mỗi khối PA phụ trách xử lý 1 kênh đầu vào (Input Channel) và T_m bộ lọc tương ứng.
- **Khối Tổng hợp (Reduction Unit - Mapping + Summary):** Thực hiện chức năng cộng dồn (Reduction) kết quả từ T_c khối PA. Do tích chập là phép tổng trọng số qua các kênh, khối này sẽ cộng giá trị Partial Sum từ các kênh IFM khác nhau để tạo ra T_m giá trị OFM bán hoàn chỉnh.
- **Khối Tích lũy (Accumulator):** Sử dụng bộ nhớ Ping-Pong để lưu trữ và cộng dồn kết quả qua các Pass (theo chiều sâu kênh C). Khi một điểm ảnh OFM đã được tích lũy đủ số kênh cần thiết, nó sẽ được gửi đi thông qua bus dữ liệu chung và vị trí nhớ đó sẽ được reset về 0 để chuẩn bị cho lượt tính mới.

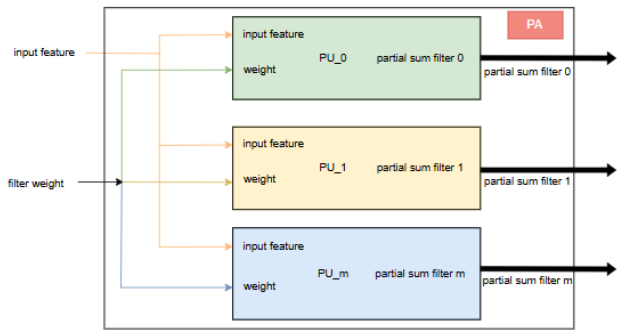
4.3.2 Tổ chức Mảng tính toán (Processing Hierarchy)

Kiến trúc tính toán được tổ chức theo mô hình phân cấp gồm 3 tầng: Process Array (PA), Process Unit (PU) và Process Element (PE).

4.3.2.1 Mảng xử lý (Process Array - PA)

Khối PA (Hình 4.5) được thiết kế để khai thác tính song song mức kênh đầu ra (Output Channel Parallelism).

- Mỗi PA chịu trách nhiệm tính toán cho 1 kênh đầu vào (Input Channel) duy nhất nhưng tạo ra kết quả cho T_m bộ lọc (Filters) khác nhau.
- **Luồng dữ liệu:** Trọng số đầu vào (Input Filter Weights) được rẽ nhánh (demultiplex) tới các PU cụ thể (ví dụ: PU_0 nhận trọng số của Filter 0). Ngược lại, dữ liệu IFM được quảng bá (broadcast) dùng chung cho tất cả các PU trong cùng một PA, giúp tiết kiệm băng thông đọc IFM.

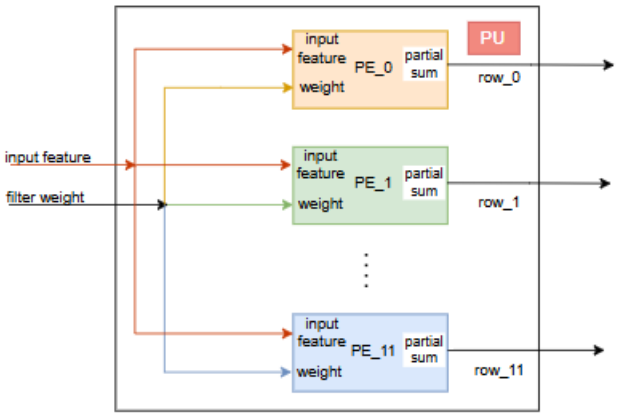


Hình 4.5: Kiến trúc bên trong khối Process Array (PA)

4.3.2.2 Đơn vị xử lý (Process Unit - PU)

Mỗi PU (Hình 4.6) bao gồm 11 phần tử xử lý (PE) hoạt động song song, tương ứng với khả năng hỗ trợ kích thước bộ lọc tối đa là 11×11 (chiều cao $R = 11$).

- Mỗi PE trong PU chịu trách nhiệm tính toán tích chập cho **1 hàng** của bộ lọc (Filter Row).
- Các PE hoạt động đồng bộ. Sau mỗi khoảng thời gian ΔT chu kỳ, PU sẽ tạo ra một cột kết quả gồm R giá trị tương ứng với R hàng của bộ lọc.

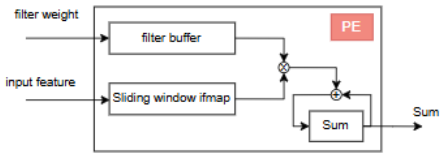


Hình 4.6: Kiến trúc khối Process Unit (PU) với các PE hoạt động song song

4.3.2.3 Phần tử xử lý (Process Element - PE)

PE là đơn vị tính toán cơ sở nhỏ nhất (Hình 4.7), thực hiện phép tính nhân-cộng (MAC).

- **Filter Buffer:** Lưu trữ S giá trị trọng số của một hàng filter ($1 \times S$). Buffer này hoạt động theo chế độ Weight Stationary, giữ giá trị không đổi trong suốt quá trình thực hiện 1 Pass.
- **Sliding Window Register:** Chứa S giá trị IFM ($1 \times S$). Đây là thanh ghi dịch, sau mỗi ΔT chu kỳ, dữ liệu sẽ dịch đi 1 vị trí (stride = 1) để thực hiện phép trượt cửa sổ.
- Vì mỗi PE chứa 1 bộ nhân và 1 bộ cộng, để tính tích chập 1 hàng kích thước S, hệ thống cần $\Delta T = S$ chu kỳ.



Hình 4.7: Cấu trúc bên trong một Process Element (PE)

4.3.3 Đánh giá thời gian thực thi (Performance Estimation)

Thời gian thực thi của hệ thống phụ thuộc vào loại lớp tích chập (Standard hay Depthwise) do sự khác biệt trong chiến lược luồng dữ liệu.

4.3.3.1 Thời gian xử lý một Pass cơ sở (T_{pass})

Dựa trên kiến trúc Pipeline của các Process Element (PE), thời gian để hoàn thành tính toán cho một tile có chiều cao T_h và độ rộng OFM W_{out} được xác định bởi:

$$T_{pass} = \Big[(W_{out} - 1) \times (S + U - 1) + S\Big] \times T_h \tag{4.12}$$

Trong đó:

- S: Kích thước bộ lọc (Filter width).
- U: Bước trượt (Stride).
- W_{out}: Chiều rộng của OFM.
- $(S+U-1)$: Số chu kỳ trung bình để tính một điểm ảnh tiếp theo nhờ tối ưu hóa Pipeline (khi $U = 1$, thời gian này là S).

4.3.3.2 Tổng thời gian thực thi (T_{total})

Trường hợp 1: Standard Convolution

Với tích chập tiêu chuẩn, mỗi điểm ảnh đầu ra là tổng hợp của tất cả C kênh đầu vào. Hệ thống phải thực hiện vòng lặp tích lũy qua các khối kênh T_c.

$$T_{total\_std} = \underbrace{\left\lceil \frac{N_f}{T_m} \right\rceil}_{\text{Output Blocks}} \times \underbrace{\left\lceil \frac{C}{T_c} \right\rceil}_{\text{Input Blocks}} \times \underbrace{\left\lceil \frac{H}{T_h} \right\rceil}_{\text{Height Blocks}} \times T_{pass} \tag{4.13}$$

Trường hợp 2: Depthwise Convolution

Với tích chập chiều sâu, các kênh hoạt động độc lập ($N_f = C$). Hệ thống không cần thực hiện vòng lặp tích lũy kênh đầu vào ($\lceil C/T_c \rceil$ bị loại bỏ). Các nhóm kênh được xử lý song song dựa trên khả năng của phần cứng (T_m).

$$T_{total\_dw} = \underbrace{\left\lceil \frac{N_f}{T_m} \right\rceil}_{\text{Channel Groups}} \times \underbrace{\left\lceil \frac{H}{T_h} \right\rceil}_{\text{Height Blocks}} \times T_{pass} \tag{4.14}$$

Nhận xét: So với Standard Convolution, Depthwise Convolution giảm được hệ số $\lceil C/T_c \rceil$ lần số lượng tính toán, giúp tăng tốc độ xử lý đáng kể đối với các mạng nhẹ (Lightweight CNNs) như MobileNet.

Chương 5

Hiện thực nền tảng SoC

Chương này trình bày quá trình xây dựng hệ thống SoC cơ sở trên FPGA, bao gồm cấu hình vi xử lý, hệ thống bus và tích hợp các ngoại vi.

5.1 Môi trường và Công cụ hiện thực

5.2 Cấu hình hệ thống xử lý (Processing System)

5.3 Thiết kế hệ thống kết nối (Interconnect Subsystem)

5.4 Tích hợp và Kiểm thử nền tảng cơ sở

Chương 6

Đánh giá hiệu năng lý thuyết

Chương này sử dụng các mô hình giải tích để ước lượng hiệu năng, độ trễ và tài nguyên tiêu thụ của kiến trúc đề xuất.

6.1 Phương pháp đánh giá: Mô hình Roofline

6.2 Ước lượng độ trễ và Tài nguyên

6.3 So sánh với các nghiên cứu liên quan

Chương 7

Kế hoạch phát triển

Chương này tổng kết các kết quả đạt được trong Giai đoạn 1 và đề ra kế hoạch chi tiết cho việc hiện thực và kiểm thử trong Giai đoạn 2.

7.1 Đánh giá mức độ hoàn thành Giai đoạn 1

7.2 Kế hoạch thực hiện Giai đoạn 2

7.3 Tiến độ dự kiến