**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KỸ THUẬT MÁY TÍNH**

**DƯƠNG ANH KHÔI**

**LÂM XUÂN PHƯỚC**

**BÁO CÁO ĐỒ ÁN 1**

**THIẾT KẾ BỘ MÁY HỌC SÂU**

**Design of Deep Learning Machines**

**CỬ NHÂN KỸ THUẬT MÁY TÍNH**

**TP. HỒ CHÍ MINH, 06/2024**

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KỸ THUẬT MÁY TÍNH**

**DƯƠNG ANH KHÔI - 22520696**

**LÂM XUÂN PHƯỚC - 22521154**

**BÁO CÁO ĐỒ ÁN 1**

**THIẾT KẾ BỘ MÁY HỌC SÂU**

**Design of Deep Learning Machines**

**CỬ NHÂN KỸ THUẬT MÁY TÍNH**

**GIẢNG VIÊN HƯỚNG DẪN**

**TRẦN ĐẠI DƯƠNG**

**TP. HỒ CHÍ MINH, 06/2024**

**LỜI CẢM ƠN**

Lời đầu tiên, nhóm chúng em gồm Dương Anh Khôi và Lâm Xuân Phước xin gửi lời cảm ơn chân thành đến thầy hướng dẫn – Ths. Trần Đại Dương. Thầy đã tận tình dành thời gian quý báu để hướng dẫn, hỗ trợ, góp ý và chỉnh sửa cho nhóm trong suốt thời gian hoàn thành Đồ Án 1.

Bên cạnh đó, nhóm chúng em cũng xin chân thành cảm ơn đến Thầy/Cô, bạn bè khoa Kỹ thuật Máy tính tính nói riêng và trường Đại học Công nghệ Thông tin – Đại học quốc gia Thành phố Hồ Chí Minh nói chung đã tận tình hỗ trợ, giảng dạy những kiến thức bổ ích để nhóm có thể hoàn thành Đồ Án 1 một cách tốt đẹp.

Trong suốt quá trình thực hiện Đồ Án, nhóm không thể tránh khỏi những khó khăn, thách thức cũng như là giới hạn và thiếu sót do một số giới hạn về mặt thời gian và kiến thức chuyên sâu. Vì thế mong quý Thầy/Cô và các bạn thông cảm, và có thể đóng góp để đề tài Đồ Án 1 này được phát triển và hoàn thiện hơn trong tương lai.

Một lần nữa nhóm xin chân thành cảm ơn.

TP. Hồ Chí Minh, ngày 23 tháng 03 năm 2025

Sinh viên thực hiện

**Dương Anh Khôi**

**Lâm Xuân Phước**

**MỤC LỤC**

[CHƯƠNG 1. GIỚI THIỆU TỔNG QUAN ĐỀ TÀI 1](#_Toc201743410)

[1.1. Tổng quan và lý do thực hiện đề tài 1](#_Toc201743411)

[1.2. Các nghiên cứu liên quan 2](#_Toc201743412)

[1.3. Mục tiêu của đề tài 3](#_Toc201743413)

[1.4. Phương pháp nghiên cứu 4](#_Toc201743414)

[1.5. Giới hạn của đề tài và phương pháp đánh giá hệ thống 5](#_Toc201743415)

[CHƯƠNG 2. KIẾN TRÚC BỘ TĂNG TỐC 6](#_Toc201743416)

[2.1. Định luật Moore 6](#_Toc201743417)

[2.2. Định luật Dennard Scaling 6](#_Toc201743418)

[2.2.1. Giới hạn và Thay đổi 6](#_Toc201743419)

[2.2.2. Giải pháp 7](#_Toc201743420)

[2.3. Bộ gia tốc (accelerator) 7](#_Toc201743421)

[2.4. Trình tự hoạt động (Operation Sequencing) 8](#_Toc201743422)

[2.5. Kiến Trúc Spatial cho Deep Learning 9](#_Toc201743423)

[2.5.1. Các khối logic cấu hình (Configurable Logic Blocks - CLB) 9](#_Toc201743424)

[2.5.2. Ứng dụng 10](#_Toc201743425)

[2.6. Khối không đồng nhất (Heterogeneous Blocks) trong FPGA 10](#_Toc201743426)

[2.7. Accelerator Taxonomy 10](#_Toc201743427)

[2.7.1. Coarse (ALU) Grained 11](#_Toc201743428)

[CHƯƠNG 3. DATAFLOW 12](#_Toc201743429)

[3.1 Kiến trúc tính toán không gian và phân cấp bộ nhớ 12](#_Toc201743430)

[3.1.1 Mô hình tính toán không gian (Spatial Compute Paradigm) 12](#_Toc201743431)

[3.1.2. Truy xuất bộ nhớ là nút thắt cổ chai 13](#_Toc201743432)

[3.1.3. Các loại tái sử dụng dữ liệu trong mạng neural sâu (DNN) 14](#_Toc201743433)

[3.1.4. Phương pháp cải thiện hiệu năng truy cập DRAM 15](#_Toc201743434)

[3.1.5 Leverage Parallelism - Tận dụng song song 16](#_Toc201743435)

[3.1.5.1. Higher performance - Cải thiện hiệu suất 16](#_Toc201743436)

[3.1.5.2. Spatial Data Reuse - Tối ưu tái dữ liệu không gian 18](#_Toc201743437)

[3.2 Các chiến lược luồng dữ liệu trong tính toán 20](#_Toc201743438)

[3.2.1. Output Stationary (OS) 20](#_Toc201743439)

[3.2.2. Weight Stationary (WS) 21](#_Toc201743440)

[3.2.3 Input Stationary (IS) 22](#_Toc201743441)

[3.2.4. Luồng dữ liệu tiết kiệm năng lượng: Row Stationary (RS) 23](#_Toc201743442)

[3.2.5. So sánh với các luồng dữ liệu khác 24](#_Toc201743443)

[3.3. Quá trình tính toán tích chập 25](#_Toc201743444)

[3.3.1 Tính toán 1D theo phương pháp Row Stationary 25](#_Toc201743445)

[3.3.2 Tính toán tích chập 2D trong mảng PE 26](#_Toc201743446)

[3.3.3 Tái sử dụng dữ liệu trong tính toán tích chập 27](#_Toc201743447)

[3.3.3.1 Tái sử dụng hàng bộ lọc theo chiều ngang giữa các PE 27](#_Toc201743448)

[3.3.3.2 Tái sử dụng hàng đặc trưng đầu vào theo đường chéo 28](#_Toc201743449)

[3.3.3.3. Tích lũy tổng một phần (partial sums) theo chiều dọc 28](#_Toc201743450)

[CHƯƠNG 4. ÁNH XẠ TỚI PHẦN CỨNG 30](#_Toc201743451)

[4.1. Điều phối dữ liệu 30](#_Toc201743452)

[4.2. Các phương pháp điều phối dữ liệu 30](#_Toc201743453)

[4.3. EDDO Strategies 31](#_Toc201743454)

[4.3.1. Buffets 31](#_Toc201743455)

[4.3.2. Các Đặc Tính Hành Vi của Buffet (Buffet Behavioral Attributes) 31](#_Toc201743456)

[4.4. Ánh xạ đầu ra cố định sang phần cứng (Mapping Output Stationary (OS) to Hardware) 32](#_Toc201743457)

[4.4.1. Tích chập 1D (1-D Convolution) 32](#_Toc201743458)

[4.4.2. Luồng dữ liệu trong một PE ((Single PE Output Stationary Flow) 32](#_Toc201743459)

[4.4.3. Mô hình tham chiếu (Reference Pattern) 32](#_Toc201743460)

[4.4.4. Truy cập dữ liệu cấp L1 (L1 Data Accesses) 32](#_Toc201743461)

[4.4.5. Bộ nhớ đệm trung gian (Intermediate Buffering) 33](#_Toc201743462)

[4.4.6. Ánh xạ không gian (Spatial Mapping) 33](#_Toc201743463)

[4.4.7. Trọng số cố định trong không gian (Spatial Weight Stationary - Skewed) 33](#_Toc201743464)

[4.4.8. Ánh xạ ( Mapping) 34](#_Toc201743465)

[4.4.9. Mô hình tính toán CPU vs. DNN (CPU vs. DNN Compute Models) 34](#_Toc201743466)

[4.4.10. Công cụ hỗ trợ ánh xạ (Mapping Tools) 35](#_Toc201743467)

[4.4.11. Các đặc tả cần thiết cho ánh xạ (Specifications for Mapping) 35](#_Toc201743468)

[CHƯƠNG 5. CO-DESIGN OF DNN MODELS AND HARDWARE: SPARSITY 36](#_Toc201743469)

[5.1. Nguồn gốc của độ thưa: 36](#_Toc201743470)

[5.2. Khai thác tính thưa thớt: 36](#_Toc201743471)

[5.3. Các kỹ thuật cụ thể 37](#_Toc201743472)

[5.3.1. Nén dữ liệu thưa 37](#_Toc201743473)

[5.3.2. Cnvlutin 37](#_Toc201743474)

[5.3.3. Pruning activation 37](#_Toc201743475)

[5.3.4. Khai thác ReLU 38](#_Toc201743476)

[5.3.5. Tương quan không gian/thời gian 39](#_Toc201743477)

[5.4. Graph Neural Networks (GNN) 39](#_Toc201743478)

[5.5. Khai thác trọng số dư thừa: 40](#_Toc201743479)

[5.5.1. Pruning – Make Weights Sparse 40](#_Toc201743480)

[5.5.2. Loại bỏ trọng số: Chấm điểm 41](#_Toc201743481)

[5.5.3. Gom nhóm 42](#_Toc201743482)

[5.5.4. Các loại cắt tỉa khác 42](#_Toc201743483)

[5.6. Tinh chỉnh và Lập lịch trình 42](#_Toc201743484)

[5.7. Hiệu quả và tradeoff 42](#_Toc201743485)

[5.8. Phần cứng hỗ trợ 43](#_Toc201743486)

[5.9. Kết luận 43](#_Toc201743487)

[CHƯƠNG 6. SPARSE ARCHITECTURES 44](#_Toc201743488)

[6.1 Tổng quan về Sparse Tensor 44](#_Toc201743489)

[6.1.1 Các loại Tensor 45](#_Toc201743490)

[6.1.2 Thuật ngữ dữ liệu Tensor 45](#_Toc201743491)

[6.1.3 Biểu diễn Tensor dựa trên cây và sợi (Tree-based and Fiber) 46](#_Toc201743492)

[6.2. Phương pháp biểu diễn tọa độ ngầm 48](#_Toc201743493)

[6.2.1. Run-Length Encoding (RLE): 49](#_Toc201743494)

[6.2.2. Significance Map Coding (Bitmask Coding) 49](#_Toc201743495)

[6.2.3. Huffman Encoding 49](#_Toc201743496)

[6.2.4. So sánh các phương pháp 50](#_Toc201743497)

[6.2.5 Ứng dụng các phương pháp nén trong thực tế 50](#_Toc201743498)

[6.3. Phép nhân ma trận thưa 55](#_Toc201743499)

[6.3.1. Kiến trúc với trọng số thưa (Sparse Weights) 56](#_Toc201743500)

[6.3.1.1. Fiber Splitting trong không gian vị trí 58](#_Toc201743501)

[6.3.2 Kiến trúc với đầu vào thưa (Sparse Inputs) 59](#_Toc201743502)

[6.3.2.1 Sparse Sliding Window 60](#_Toc201743503)

[6.4 Mạng CNN thưa (Sparse CNN) 61](#_Toc201743504)

[6.4.1 Kiến trúc của SCNN 61](#_Toc201743505)

[6.4.2 Phép chiếu và giao tọa độ sợi 62](#_Toc201743506)

[6.4.3 Mô hình luồng dữ liệu IS-OS 64](#_Toc201743507)

[CHƯƠNG 7. CHƯƠNG 7. SPARSE MATRIX MULTIPLICATION ACCELERATOR ARCHITECTURE 66](#_Toc201743508)

[7.1. Fully-Connected (FC) Layer 66](#_Toc201743509)

[7.2. Einsum 66](#_Toc201743510)

[7.3. ExTensor - Tổng quan 68](#_Toc201743511)

[7.4. Quan sát từ hoạt hình hai cấp 68](#_Toc201743512)

[7.5. OuterSPACE - Tổng quan 68](#_Toc201743513)

[7.6. Gamma: 69](#_Toc201743514)

[7.6.1. Tạo Tensor Trung gian T 69](#_Toc201743515)

[7.6.2. Rank Swizzled T 70](#_Toc201743516)

[7.6.3. FiberCache 71](#_Toc201743517)

[7.7. Mô hình hóa các luồng dữ liệu thưa thớt – TeAAL 71](#_Toc201743518)

[CHƯƠNG 8. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 72](#_Toc201743519)

[8.1. Quá trình học tập và tìm hiểu 72](#_Toc201743520)

[8.2. Đánh giá Đồ án 1 72](#_Toc201743521)

[8.3. Hướng phát triển trong Đồ án 2 72](#_Toc201743522)

**DANH MỤC HÌNH**

[Hình 2.1: Định Luật Moore 6](#_Toc201743591)

[Hình 2.2: Định Luật Scaling 7](#_Toc201743592)

[Hình 2.3: Kiến trúc Temporal 8](#_Toc201743593)

[Hình 2.4: Kiến trúc Spatial 8](#_Toc201743594)

[Hình 2.5: Kiến trúc Spatial dành cho CNN 9](#_Toc201743595)

[Hình 2.6: Khối logic có thể cấu hình 10](#_Toc201743596)

[Hình 2.7: Khối không đồng nhất 10](#_Toc201743597)

[Hình 3.1: Tổ chức ma trận ALU và phân cấp bộ nhớ 12](#_Toc201743598)

[Hình 3.2: Sơ đồ khối phép tính MAC 13](#_Toc201743599)

[Hình 3.3: Sơ đồ Truy xuất bộ nhớ và phép tính MAC 13](#_Toc201743600)

[Hình 3.4: Sơ đồ phân cấp Bộ nhớ Cục bộ Bổ sung 14](#_Toc201743601)

[Hình 3.5: Sơ đồ minh họa các phương pháp tái sử dụng dữ liệu 14](#_Toc201743602)

[Hình 3.6: Sơ đồ minh họa nút thắt cổ chai trong truy cập bộ nhớ 16](#_Toc201743603)

[Hình 3.7: Sơ đồ tận dụng song song các Module ALU và bộ nhớ 17](#_Toc201743604)

[Hình 3.8: Sơ đồ tối ưu hóa tái sử dụng dữ liệu không gian 18](#_Toc201743605)

[Hình 3.9: Phân tích chi phí năng lượng theo cấu trúc bộ nhớ 19](#_Toc201743606)

[Hình 3.10: Kiến trúc Output Stationary 20](#_Toc201743607)

[Hình 3.11: Kiến trúc Weight Stationary 21](#_Toc201743608)

[Hình 3.12: Kiến trúc Input Stationary 22](#_Toc201743609)

[Hình 3.13: Kiến trúc Row Stationary 23](#_Toc201743610)

[Hình 3.14: Quá trình tính toán tích chập 1D theo phương pháp Row Stationary 26](#_Toc201743611)

[Hình 3.15: Quá trình tính toán tích chập 2D theo phương pháp Row Stationary 27](#_Toc201743612)

[Hình 3.16: Phân phối hàng bộ lọc trên các PE theo chiều ngang 27](#_Toc201743613)

[Hình 3.17: Phân phối hàng bộ lọc trên các PE theo đường chéo 28](#_Toc201743614)

[Hình 3.18: Phân phối hàng bộ lọc trên các PE theo chiều dọc 29](#_Toc201743615)

[Hình 4.1: Nguyên tắc hướng dẫn sắp xếp dữ liệu 30](#_Toc201743616)

[Hình 4.2: Ví dụ về điều khiển Buffer 31](#_Toc201743617)

[Hình 5.1: Sự thưa thớt giữa có và không có hàm kích hoạt 36](#_Toc201743618)

[Hình 5.2: Cnvlutin 37](#_Toc201743619)

[Hình 5.3: Biểu đồ của hàm kích hoạt cắt tỉa 38](#_Toc201743620)

[Hình 5.4: Khai thác ReLU 39](#_Toc201743621)

[Hình 5.5: Khai thác tương quan không gian đầu vào 39](#_Toc201743622)

[Hình 5.6: Mạng nơ-ron đồ thị 40](#_Toc201743623)

[Hình 5.7: Khai thác trọng số dư thừa 40](#_Toc201743624)

[Hình 5.8: Biểu đồ khi áp dụng phương pháp cắt tỉa 41](#_Toc201743625)

[Hình 5.9: Làm cho trọng lượng thưa thớt bằng phương pháp cắt tỉa 41](#_Toc201743626)

[Hình 5.10: Quy trình áp dụng phương pháp cắt tỉa 41](#_Toc201743627)

[Hình 5.11: Ví dụ về loại bỏ gom nhóm 42](#_Toc201743628)

[Hình 5.12: Cắt tỉa và các loại lớp 43](#_Toc201743629)

[Hình 6.1: Thang đo mật độ của Sparse Tensor trên các ứng dụng khác nhau 44](#_Toc201743630)

[Hình 6.2: Biểu diễn tọa độ và Ranks trong ma trận 46](#_Toc201743631)

[Hình 6.3: Cấu trúc cây Tensor 47](#_Toc201743632)

[Hình 6.4: Biểu diễn Fiber trong cấu trúc Tensor 47](#_Toc201743633)

[Hình 6.5: Biểu diễn Tensor thưa - Loại bỏ các Fiber bằng không 48](#_Toc201743634)

[Hình 6.6: Biểu diễn CSR 51](#_Toc201743635)

[Hình 6.7: Biểu diễn CSR và CSC 52](#_Toc201743636)

[Hình 6.8: Biểu diễn Merging Ranks 52](#_Toc201743637)

[Hình 6.9: Phân chia Coordinate Space 53](#_Toc201743638)

[Hình 6.10: Phân chia Position Space 54](#_Toc201743639)

[Hình 6.11: Minh họa điều kiện nhân ma trận - Chiều chia sẻ K 55](#_Toc201743640)

[Hình 6.12: So sánh trọng số nén và không nén 56](#_Toc201743641)

[Hình 6.13: So sánh kiến trúc Weight Stationary và Output Stationary - Sparse Weight 57](#_Toc201743642)

[Hình 6.14: Minh họa Fiber Splitting 58](#_Toc201743643)

[Hình 6.15: Kiến trúc Weight Stationary - Sparse Inputs 59](#_Toc201743644)

[Hình 6.16: Kiến trúc Output Stationary – Sparse Inputs 60](#_Toc201743645)

[Hình 6.17: Minh họa Sparse Sliding Window 60](#_Toc201743646)

[Hình 6.18: Kiến trúc SCNN 61](#_Toc201743647)

[Hình 6.19: Minh họa phép chiếu tọa độ Fiber – tịnh tiến vị trí 62](#_Toc201743648)

[Hình 6.20: Minh họa phép chiếu tọa độ Fiber – Tìm phần tử chung 63](#_Toc201743649)

[Hình 6.21: Kiến trúc luồng dữ liệu IS-OS 64](#_Toc201743650)

[Hình 7.1: Lớp kết nối đầy đủ 66](#_Toc201743651)

[Hình 7.2: Chỉ số chia sẻ tác động đến giao nhau 67](#_Toc201743652)

[Hình 7.3: Chỉ số không được thu gọn ảnh hưởng đến đầu ra 67](#_Toc201743653)

[Hình 7.4: Biểu diễn phép nhân ma trận thưa 67](#_Toc201743654)

[Hình 7.5: Thiết kế OuterSpace 69](#_Toc201743655)

[Hình 7.6: Tensor trung gian T 70](#_Toc201743656)

[Hình 7.7: Rank Swizzled T 70](#_Toc201743657)

[Hình 7.8: Mô hình TeAAL 71](#_Toc201743658)

**DANH MỤC BẢNG**

[Bảng 3.1: Tiêu chí đánh giá các luồng dữ liệu 24](#_Toc195896751)

[Bảng 4.1: So sánh giữa mô hình CPU và CNN........................................................... 34](#_Toc195896757)

**DANH MỤC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **Viết tắt** | **Viết đầy đủ** | **Ý nghĩa** |
| CNN | Convolutional Neural Network | Mạng nơ-ron tích chập, một loại mạng nơ-ron được thiết kế đặc biệt cho xử lý dữ liệu có cấu trúc lưới như hình ảnh. |
| OS | Output Stationary | Chiến lược giữ cố định đầu ra tại mỗi phần tử xử lý để tối ưu năng lượng và hiệu suất. |
| WS | Weight Stationary | Chiến lược giữ cố định trọng số trong PE, giúp giảm truy cập bộ nhớ. |
| IS | Input Stationary | Chiến lược giữ cố định dữ liệu đầu vào để tái sử dụng trong tính toán, hiệu quả với mạng thưa. |
| PE | Processing Element | Phần tử xử lý cơ bản trong kiến trúc phần cứng, thực hiện phép toán như tích chập hoặc nhân–cộng. |
| MAC | Multiply-Accumulate Operation | Phép toán nhân rồi cộng vào tổng, cơ bản trong xử lý học sâu và tín hiệu số. |
| GNNs | Graph Neural Networks | Mạng nơ-ron đồ thị, xử lý dữ liệu dạng đồ thị như mạng xã hội, phân tích mối liên kết. |
| RLE | Run-Length Encoding | Kỹ thuật nén không mất mát bằng cách mã hóa các chuỗi lặp lại thành (giá trị, số lần). |
| FPGA | Field-Programmable Gate Array | Mạch logic khả trình, cấu hình lại sau sản xuất, dùng mô phỏng hoặc tăng tốc phần cứng. |
| RNN | Recurrent Neural Network | Mạng nơ-ron hồi tiếp, dùng cho dữ liệu chuỗi như văn bản, âm thanh, dữ liệu thời gian thực. |
| AI | Artificial Intelligence | Trí tuệ nhân tạo, mô phỏng hành vi thông minh của con người bằng máy móc. |
| GPU | Graphics Processing Unit | Bộ xử lý đồ họa, dùng cho tính toán song song trong AI, học sâu. |
| TPU | Tensor Processing Unit | Bộ xử lý tensor của Google, tăng tốc học sâu chuyên biệt. |
| ASIC | Application-Specific Integrated Circuit | Mạch tích hợp chuyên dụng, hiệu suất cao cho một nhiệm vụ cụ thể. |
| SIMD | Single Instruction, Multiple Data | Một lệnh thực hiện trên nhiều dữ liệu đồng thời, hiệu quả trong xử lý ma trận, vector. |
| CLB | Configurable Logic Block | Khối logic có thể lập trình trong FPGA, thực hiện các phép toán logic. |
| BLE | Basic Logic Elements | Phần tử logic cơ bản, gồm LUT, thanh ghi, multiplexer bên trong CLB. |
| LUT | Look-Up Table | Bảng tra giá trị thực hiện logic thông qua cơ chế tra cứu nhanh, linh hoạt. |
| FC | Fully-Connected | Kết nối đầy đủ, loại lớp trong mạng nơ-ron mà mỗi nơ-ron kết nối với tất cả các nơ-ron ở lớp trước. |
| EDDO | Explicit Decoupled Data Orchestration | Điều phối dữ liệu tách rời tường minh, phương pháp quản lý dữ liệu trong các kiến trúc tăng tốc. |
| BRAM | Block RAM | RAM khối, bộ nhớ tích hợp trong FPGA. |

**TÓM TẮT ĐỒ ÁN 1**

Nghiên cứu Thiết kế Bộ Tăng Tốc Tính Toán Ma Trận Thưa

Đồ án 1 tập trung vào nghiên cứu các phương pháp thiết kế bộ tăng tốc phần cứng chuyên dụng cho các phép tính trên ma trận thưa, một thành phần cốt lõi trong nhiều ứng dụng quan trọng như mạng nơ-ron sâu (DNNs), đồ thị học và khoa học tính toán. Việc xử lý hiệu quả các ma trận thưa là một thách thức do sự phân bố không đồng đều của các phần tử khác không, dẫn đến sự lãng phí tài nguyên tính toán và bộ nhớ nếu không được quản lý đúng cách.

Độ thưa (Sparsity) là sự tồn tại của nhiều phần tử bằng không trong ma trận, phổ biến trong nhiều tác vụ tính toán. Khai thác hiệu quả độ thưa giúp giảm số phép toán, tiết kiệm bộ nhớ, tăng tốc độ tính toán và giảm tiêu thụ năng lượng. Trong mạng nơ-ron sâu (DNNs), độ thưa xuất hiện ở cả trọng số (weights) và dữ liệu kích hoạt (activations). Các kỹ thuật khai thác gồm: Tỉa thưa (Pruning): Loại bỏ trọng số hoặc kết nối không quan trọng, có thể dựa trên độ lớn trọng số (magnitude-based) hoặc mức độ ảnh hưởng đến mô hình (feature-based). Tinh chỉnh (Fine-tuning): Khôi phục độ chính xác sau khi tỉa thưa.

Để khai thác độ thưa hiệu quả, cần sử dụng các cấu trúc dữ liệu phù hợp cho ma trận thưa. Các phương pháp gồm: Biểu diễn dựa trên tọa độ ngầm định, chỉ lưu trữ phần tử khác không với tọa độ được ngầm định qua thứ tự; Mã hóa độ dài chạy (RLC), gửi độ dài chuỗi các giá trị không thay vì lưu trữ trực tiếp; Bản đồ độ quan trọng, dùng một bit cho mỗi phần tử để xác định giá trị khác không và chỉ gửi giá trị khi cần; Trừu tượng Tensor Fibertree, cấu trúc linh hoạt cho tensor thưa đa chiều, giúp truy cập và thao tác hiệu quả trên phần tử khác không, hỗ trợ tối ưu hóa như chia tách và sắp xếp lại dữ liệu.

Tổ chức luồng dữ liệu (Dataflow) ảnh hưởng đến hiệu suất bộ tăng tốc, quyết định cách dữ liệu di chuyển giữa đơn vị xử lý và bộ nhớ. Các kiểu chính gồm: OS (Output Stationary) – giữ đầu ra trong đơn vị xử lý, giảm năng lượng ghi/đọc trung gian; WS (Weight Stationary) – giữ trọng số và tái sử dụng với nhiều đầu vào; IS (Input Stationary) – giữ đầu vào để tạo nhiều đầu ra. Với tính toán thưa, cần bỏ qua phép toán liên quan đến phần tử không, chỉ thực hiện nhân khi cả đầu vào và trọng số khác không.

Thiết kế bộ tăng tốc tính toán ma trận thưa cần xem xét kiến trúc PE, tổ chức bộ nhớ và điều khiển luồng dữ liệu. Kiến trúc PE: Loại bỏ phép toán nhân với không bằng cách kiểm tra trước. Tổ chức bộ nhớ: Dùng cấu trúc dữ liệu thưa và bộ nhớ phân cấp để giảm dung lượng và băng thông. Điều khiển luồng dữ liệu: Đảm bảo PE chỉ thực hiện phép toán hữu ích, sử dụng kỹ thuật như sparse data tiling để chia nhỏ bài toán, tối ưu xử lý cục bộ.

Các nghiên cứu tiêu biểu về bộ tăng tốc tính toán thưa gồm EIE, bộ tăng tốc DNN thưa tối ưu hóa suy luận với deep compression, lưu trữ trọng số theo column-wise run length; ExTensor, kiến trúc dựa trên Fibertree hỗ trợ khai thác độ thưa ở nhiều cấp độ; Gamma Dataflow, luồng dữ liệu tối ưu cho nhân ma trận thưa, tận dụng concordant traversal và fibercache; TeAAL, framework khai báo mô hình hóa tăng tốc tensor thưa.

Đánh giá hiệu năng bộ tăng tốc ma trận thưa không chỉ dựa vào số lượng phép toán (MACs) mà cần xem xét các chỉ số quan trọng như độ trễ (latency), thông lượng (throughput) và hiệu quả năng lượng. Do tính chất thưa, nhiều phép toán có thể bị bỏ qua, làm cho MACs không còn phản ánh chính xác hiệu suất thực tế. Thay vào đó, các metrics phần cứng như mức tiêu thụ năng lượng, băng thông bộ nhớ và độ trễ thực thi cần được ưu tiên. Ngoài ra, các kỹ thuật tối ưu như energy-aware pruning giúp tỉa thưa mô hình theo cách giảm tiêu thụ năng lượng mà vẫn duy trì hiệu suất suy luận.

Thiết kế bộ tăng tốc tính toán ma trận thưa là một lĩnh vực quan trọng, yêu cầu sự kết hợp giữa kỹ thuật biểu diễn dữ liệu, tổ chức luồng dữ liệu tối ưu và kiến trúc phần cứng chuyên dụng. Các nghiên cứu cung cấp cái nhìn toàn diện từ lý thuyết đến thực tiễn, giúp định hướng phát triển các bộ tăng tốc hiệu quả. Đồ án 1 sẽ tiếp tục nghiên cứu các kỹ thuật này để đề xuất một thiết kế phù hợp cho ứng dụng tính toán ma trận thưa.

# GIỚI THIỆU TỔNG QUAN ĐỀ TÀI

## Tổng quan và lý do thực hiện đề tài

Hiện nay, AI phát triển mạnh mẽ, đặc biệt trong học sâu, thúc đẩy nhiều lĩnh vực như y tế, tài chính và giao thông. Những tiến bộ đáng kể trong các mô hình học sâu đã mang lại khả năng chẩn đoán bệnh chính xác hơn, hệ thống giao dịch tài chính thông minh, và giải pháp vận tải tự động hiệu quả. Các ứng dụng này không chỉ cải thiện chất lượng cuộc sống mà còn mở ra những cơ hội kinh tế mới trong kỷ nguyên số.

Tuy nhiên, các mô hình AI phức tạp đòi hỏi tính toán lớn, tiêu thụ nhiều năng lượng và kém hiệu quả trên phần cứng truyền thống khi xử lý ma trận thưa – dữ liệu phổ biến với nhiều phần tử bằng 0. Trong thực tế, các mạng nơ-ron hiện đại như GPT, ResNet hay BERT có thể chứa hàng tỷ tham số, đòi hỏi hàng nghìn tỷ phép tính cho mỗi lần suy luận. Các GPU và CPU tiêu chuẩn, mặc dù đã được tối ưu hóa theo thời gian, vẫn phải thực hiện nhiều phép tính không cần thiết với các giá trị bằng không, gây lãng phí năng lượng và làm giảm thông lượng xử lý. Nghiên cứu gần đây chỉ ra rằng trong nhiều mô hình học sâu, đặc biệt là sau khi áp dụng các kỹ thuật pruning (cắt tỉa), tỷ lệ các tham số bằng không có thể lên đến 70-90%, tạo ra một không gian tối ưu hóa khổng lồ chưa được khai thác đầy đủ.

Điều này gây chậm trễ và lãng phí tài nguyên, cản trở ứng dụng AI tiên tiến trong nhiều bối cảnh quan trọng. Đối với các thiết bị di động và IoT, giới hạn về năng lượng và tài nguyên tính toán khiến việc triển khai các mô hình phức tạp trở nên bất khả thi. Trong các trung tâm dữ liệu, chi phí điện năng và làm mát cho các hệ thống AI quy mô lớn đã trở thành một thách thức kinh tế và môi trường đáng kể. Hơn nữa, độ trễ cao trong các ứng dụng thời gian thực như xe tự hành hay robot hỗ trợ phẫu thuật có thể dẫn đến những hậu quả nghiêm trọng.

Vì vậy, bộ tăng tốc nhân ma trận thưa ra đời để tối ưu hóa tính toán, tập trung vào phần tử khác 0. Các kiến trúc phần cứng chuyên biệt này áp dụng nhiều chiến lược tiên tiến như cấu trúc dữ liệu thưa tối ưu (CSR, CSC, Fibertree), thiết kế dataflow thông minh (Weight Stationary, Output Stationary, Row Stationary), và các kỹ thuật nén không mất mát (Run-Length Encoding, Significance Map Coding). Thay vì xử lý tất cả các phần tử, các bộ tăng tốc này chỉ thực hiện tính toán trên các phần tử có giá trị khác không, đồng thời tận dụng tính song song vốn có trong các phép nhân ma trận.

Công nghệ này giúp tăng tốc độ, giảm năng lượng, nâng cao hiệu suất, hỗ trợ sự đổi mới AI trong tương lai. Các thử nghiệm ban đầu trên các bộ tăng tốc chuyên dụng như Google TPU, NVIDIA TensorCore và các thiết kế học thuật như ExTensor, OuterSPACE hay SCNN đã chứng minh khả năng cải thiện hiệu suất lên đến 10-100 lần và giảm tiêu thụ năng lượng đến 90% so với các giải pháp phần cứng thông thường. Những lợi ích này không chỉ mở rộng khả năng triển khai AI trên các thiết bị cạnh biên mà còn hỗ trợ phát triển các mô hình lớn hơn, phức tạp hơn trên các hạ tầng hiện có, thúc đẩy chu kỳ đổi mới tiếp theo trong trí tuệ nhân tạo.

Đề tài "Thiết kế bộ máy học sâu" nghiên cứu sâu về các nguyên lý và kỹ thuật nền tảng để phát triển bộ tăng tốc ma trận thưa hiệu quả. Từ việc phân tích các kiến trúc phần cứng cơ bản đến các chiến lược điều phối dữ liệu tiên tiến, đề tài hướng tới việc đề xuất và xây dựng một mô hình phần cứng tối ưu có thể triển khai thực tế. Trong đó, đặc biệt tập trung vào phương pháp nén Compressed Sparse Row (CSR) - một kỹ thuật biểu diễn ma trận thưa hiệu quả đã được chứng minh trong nhiều ứng dụng tính toán khoa học và học máy. Phương pháp CSR chỉ lưu trữ các phần tử khác không cùng với thông tin chỉ mục hàng và cột, giúp giảm đáng kể yêu cầu bộ nhớ và tăng tốc các phép nhân ma trận-vector thưa. Bằng cách tối ưu hóa cấu trúc CSR cho các bộ tăng tốc phần cứng chuyên biệt, nhóm dự kiến sẽ đạt được hiệu suất vượt trội cho các ứng dụng học sâu, đồng thời duy trì tính linh hoạt khi triển khai trên nhiều nền tảng phần cứng khác nhau. Trong bối cảnh nhu cầu tính toán cho AI tiếp tục tăng theo cấp số nhân, trong khi định luật Moore dần đạt đến giới hạn vật lý, các bộ tăng tốc chuyên biệt dựa trên CSR và các kỹ thuật tương tự sẽ đóng vai trò then chốt trong việc duy trì và thúc đẩy quỹ đạo phát triển của công nghệ AI, mở ra những ứng dụng đột phá và khả năng mới trong tương lai.

## Các nghiên cứu liên quan

Năm 2011, nhà nghiên cứu Ngô Thị Nhạn đã công bố luận văn "Thiết kế nhân ma trận thưa với vector trong tính toán song song và ứng dụng" tập trung vào tối ưu hóa phép nhân ma trận thưa-vector (SpMV) bằng phương pháp song song. Nghiên cứu này mang lại những ưu điểm như đạt được hiệu suất tính toán cao nhờ thuật toán Mondrian phân hoạch 2D thông minh kết hợp với thuật toán Biên địa phương (LA), giúp cân bằng tải (ε=0.03) và giảm 30-50% lượng truyền thông so với phương pháp truyền thống. Tuy nhiên, nghiên cứu vẫn còn một số nhược điểm cần khắc phục như hiệu suất giảm 15-20% với các ma trận có cấu trúc không đều. Về mặt triển khai, độ phức tạp bộ nhớ O(nz+P²) gây khó khăn khi mở rộng quy mô. Ngoài ra, giải pháp mới dừng lại ở SpMV.

Năm 2021, Matthew Denton trong luận văn "Acceleration of Sparse Matrix Multiplication Using Bit-Serial Arithmetic" đã phát triển bộ tăng tốc nhân ma trận thưa (SpMV) trên FPGA bằng kiến trúc bit-serial, tập trung vào ứng dụng học sâu. Tác giả sử dụng phép cộng bit-serial và biểu diễn Canonical Signed Digit (CSD) để loại bỏ phép tính thừa với giá trị 0, đồng thời tối ưu hóa phần cứng thông qua phân phối dữ liệu song song và tận dụng tính chất cố định của ma trận trọng số. Ưu điểm của giải pháp này là đạt được độ trễ cực thấp dưới 120ns, nhanh hơn 50–86 lần so với GPU Volta V100 và 4–47 lần so với bộ tăng tốc SIGMA, đồng thời giảm 17% tài nguyên FPGA nhờ biểu diễn CSD. Thiết kế phù hợp cho các ứng dụng đòi hỏi thời gian phản hồi tức thời trong môi trường điện toán biên như hệ thống nhúng và IoT. Nhược điểm chính của thiết kế là hiện tại chỉ hỗ trợ ma trận trọng số cố định trong quá trình suy luận, hạn chế khả năng áp dụng cho các mô hình cần cập nhật động.

Năm 2024, nhóm tác giả Trần Như Ý, Huỳnh Trung Hiếu và Phạm Thế Bảo trong bài báo "An Overview of Parallel Computing Models for Image Segmentation with Neut" đã nghiên cứu việc tăng tốc thuật toán Neut (Normalized Cuts) cho phân đoạn ảnh bằng các mô hình song song trên GPU và CPU. Trọng tâm là tối ưu phép tính ma trận thưa-vector (SpMV) trong bài toán trị riêng của ma trận Laplace, vốn chiếm phần lớn thời gian xử lý. Kết quả thực nghiệm cho thấy GPU (sử dụng CUDA/CUBLAS) giảm thời gian tính toán xuống 2.8 giây cho ảnh 1024×1024 so với 70.4 giây trên CPU, nhờ tận dụng kiến trúc đa lõi và bộ nhớ chia sẻ. Tuy nhiên, nghiên cứu chưa đi sâu vào các kỹ thuật cụ thể để tối ưu định dạng lưu trữ ma trận thưa (CSR, COO) hay cân bằng tải giữa các luồng. Ưu điểm là duy trì chất lượng phân đoạn ảnh, nhưng hạn chế nằm ở độ chính xác giảm khi kích thước ảnh quá lớn do giới hạn bộ nhớ GPU. Công trình mở ra hướng ứng dụng song song hóa cho các bài toán đại số tuyến tính quy mô lớn.

Dựa vào những ưu điểm và nhược điểm của các đề tài trên, nhóm quyết định tập trung vào thiết kế bộ tăng tốc nhân ma trận thưa sử dụng định dạng nén CSR (Compressed Sparse Row). Từ những ưu điểm và hạn chế của các công trình của Ngô Thị Nhạn (2011), Matthew Denton (2021) và nhóm Trần Như Ý (2024), nhóm nhận thấy định dạng CSR mang lại sự cân bằng tối ưu giữa hiệu năng, khả năng mở rộng và linh hoạt trong triển khai. Phương pháp CSR giải quyết được vấn đề tiêu tốn bộ nhớ O(nz+P²) trong nghiên cứu của Nhạn mà vẫn duy trì khả năng xử lý hiệu quả cho các ma trận có cấu trúc không đồng nhất. Đồng thời, định dạng này có thể được tối ưu hóa cho kiến trúc bit-serial như trong công trình của Denton để đạt độ trễ thấp, nhưng không bị giới hạn ở trọng số cố định, cho phép áp dụng rộng rãi hơn trong cả quá trình huấn luyện lẫn suy luận. Thêm vào đó, CSR đã được chứng minh là có hiệu quả cao khi triển khai trên các nền tảng song song như GPU trong nghiên cứu của nhóm Trần Như Ý, đặc biệt khi kết hợp với các thuật toán cân bằng tải và tối ưu hóa truy cập bộ nhớ. Phương pháp tiếp cận của nhóm sẽ khắc phục các hạn chế của các nghiên cứu trước đó bằng cách thiết kế một kiến trúc linh hoạt, có khả năng xử lý hiệu quả cả ma trận thưa và ma trận không đồng nhất, đồng thời tận dụng tối đa kiến trúc phần cứng hiện đại để đạt được hiệu suất cao nhất cho các ứng dụng học sâu.

## Mục tiêu của đề tài

Mục tiêu chính là nghiên cứu và phát triển một kiến trúc phần cứng tăng tốc hiệu quả cho phép nhân ma trận thưa (Sparse Matrix-Vector Multiplication – SpMV), tập trung vào ứng dụng kỹ thuật nén dữ liệu để giảm lưu lượng truy cập bộ nhớ, tăng hiệu suất xử lý và tiết kiệm điện năng cho các hệ thống nhúng. Đặc biệt, nhóm lựa chọn khảo sát và tích hợp phương pháp Compressed Sparse Row (CSR) như một giải pháp nhẹ, hiệu quả và phù hợp với kiến trúc phần cứng.

Từ mục tiêu tổng quát, nhóm chia thành 5 mục tiêu cụ thể như sau:

1. Nghiên cứu tổng quan các phương pháp nén dữ liệu hiện nay như Huffman, LZW, và Compressed Sparse Row (CSR), nhằm xác định đặc điểm phù hợp với bài toán nén chuỗi dữ liệu trong ma trận thưa, đặc biệt là khả năng hiện thực hóa bằng phần cứng.
2. Đánh giá chi tiết ưu và nhược điểm của từng phương pháp nén theo các tiêu chí: tỷ lệ nén, độ phức tạp thuật toán, khả năng hiện thực phần cứng, và tính tương thích với đặc trưng của dữ liệu ma trận thưa.
3. So sánh hiệu quả của kỹ thuật CSR với các định dạng lưu trữ ma trận thưa như CSC, COO và RLE theo các tiêu chí: mức độ nén, số lượng truy cập bộ nhớ, độ phức tạp khi tích hợp vào pipeline nhân SpMV.
4. Thiết kế và hiện thực mô hình thử nghiệm của phương pháp CSR trên các tập dữ liệu ma trận thưa thực tế (ví dụ: từ thư viện SuiteSparse), từ đó đo đạc hiệu quả nén, chi phí bộ nhớ, độ trễ xử lý, và khả năng tích hợp vào hệ thống nhúng.
5. Đề xuất kiến trúc phần cứng tích hợp CSR phù hợp với môi trường thực tế (như trên FPGA), cùng với các kỹ thuật tối ưu hóa pipeline, truy cập bộ nhớ và tính toán song song, nhằm tối đa hoá hiệu quả xử lý trong ứng dụng SpMV trên hệ thống nhúng.

Đề tài này tập trung vào việc tích hợp kỹ thuật nén dữ liệu Compressed Sparse Row (CSR) vào quá trình tăng tốc phép nhân ma trận thưa (SpMV), với kỳ vọng giảm băng thông bộ nhớ và nâng cao hiệu suất phần cứng trên các nền tảng nhúng như FPGA hoặc GPU. Khác với các nghiên cứu trước đây vốn chủ yếu tập trung vào song song hóa thuật toán (Ngô Thị Nhạn), tối ưu bit-serial (Matthew Denton), hoặc tăng cường xử lý trên GPU (Trần Như Ý, Huỳnh Trung Hiếu, Phạm Thế Bảo), đề tài này đề xuất một hướng tiếp cận mới – kết hợp cơ chế nén dữ liệu ngay trong quá trình thiết kế phần cứng tăng tốc. Cách tiếp cận này được kỳ vọng sẽ tận dụng tốt hơn tài nguyên hệ thống, đặc biệt trong các môi trường có giới hạn về bộ nhớ và năng lượng. Ngoài ra, đề tài cũng hướng tới khả năng ứng dụng trong các mô hình học sâu thực tế, nhằm đánh giá mức độ phù hợp và mở rộng của phương pháp đề xuất.

## Phương pháp nghiên cứu

Để nâng cao hiệu quả xử lý ma trận thưa, nhóm nghiên cứu áp dụng phương pháp tiếp cận đa chiều, các phương pháp cụ thể bao gồm:

1. Phương pháp nghiên cứu lý thuyết: Nhóm sẽ khảo sát và phân tích các định dạng lưu trữ ma trận thưa phổ biến như CSR, CSC và COO để làm rõ đặc trưng dữ liệu và yêu cầu xử lý. Trên cơ sở đó, nhóm tập trung nghiên cứu kỹ thuật nén Compressed Sparse Row (CSR), đánh giá tiềm năng áp dụng cho dữ liệu ma trận thưa, từ góc độ khả năng nén, độ phức tạp thuật toán và mức độ phù hợp khi hiện thực bằng phần cứng. Ngoài ra, nhóm sẽ hệ thống hóa các kiến trúc tăng tốc SpMV hiện có để xác định điểm mạnh – điểm yếu làm nền cho đề xuất của nhóm.
2. Phương pháp thiết kế hệ thống: Dựa trên kết quả phân tích lý thuyết, nhóm sẽ đề xuất thiết kế một mô-đun mã hóa và giải mã CSR dành riêng cho dữ liệu ma trận thưa, hướng tới tính hiệu quả khi triển khai bằng phần cứng. Thiết kế hệ thống sẽ bao gồm mảng xử lý song song (Processing Element Array), hỗ trợ thao tác trên các định dạng dữ liệu đã nén. Nhóm cũng sẽ xây dựng sơ bộ bộ quản lý truy cập bộ nhớ nhằm định tuyến dữ liệu phù hợp giữa các khối xử lý.
3. Phương pháp mô phỏng và đánh giá: Hệ thống sẽ được mô phỏng bằng SystemVerilog để kiểm thử tính đúng đắn về mặt chức năng. Các chỉ số như độ trễ, thông lượng, và tài nguyên FPGA sử dụng sẽ được thu thập nhằm đánh giá hiệu quả sơ bộ của phương pháp đề xuất. Việc phân tích hiệu năng sẽ được thực hiện ở nhiều cấu hình ma trận đầu vào khác nhau nhằm đánh giá tính ổn định và khả năng thích nghi của hệ thống.
4. Phương pháp triển khai thực tế: Sau giai đoạn mô phỏng, thiết kế sẽ được tổng hợp và triển khai trên board FPGA, làm bước trung gian trước khi cân nhắc tích hợp vào các hệ thống nhúng AI. Thông qua việc kiểm thử trên phần cứng thực tế, nhóm sẽ tiếp tục điều chỉnh và tối ưu hóa hệ thống nhằm nâng cao hiệu quả và giảm chi phí tài nguyên.

## 1.5. Giới hạn của đề tài và phương pháp đánh giá hệ thống

Phạm vi nghiên cứu của đề tài giới hạn trong bài toán nhân ma trận thưa với vector (SpMV) – đây là phép toán xuất hiện thường xuyên trong các ứng dụng học sâu và mô phỏng vật lý. Tuy nhiên, đề tài không bao quát các dạng ma trận phức tạp hơn như ma trận thưa đối xứng, ma trận có mẫu phân bố không đều hoặc ma trận siêu thưa với độ phân tán cao.

Việc đánh giá hệ thống sẽ được tiến hành thông qua hai giai đoạn chính: mô phỏng phần mềm và triển khai thực tế trên FPGA. Ở giai đoạn đầu, nhóm sẽ sử dụng mô phỏng chức năng bằng ngôn ngữ SystemVerilog để kiểm tra độ chính xác của thiết kế, đảm bảo rằng quá trình xử lý dữ liệu đầu vào cho kết quả đầu ra đúng như kỳ vọng. Sau khi đảm bảo tính chính xác chức năng, nhóm sẽ đánh giá tốc độ xử lý của hệ thống thông qua các chỉ số định lượng như thông lượng (tính bằng GFLOPS) và độ trễ (thời gian xử lý cho một tác vụ hoàn chỉnh). Các chỉ số này sẽ được đo trực tiếp từ hệ thống sau khi tổng hợp và triển khai trên FPGA, nhằm phản ánh hiệu suất thực tế trong điều kiện phần cứng. Thông qua quá trình đánh giá này, đề tài không chỉ kiểm chứng tính khả thi của thiết kế mà còn làm rõ các điểm mạnh, điểm hạn chế của đề tài.

# KIẾN TRÚC BỘ TĂNG TỐC

## 2.1. Định luật Moore

Theo định luật Moore, số lượng bóng bán dẫn trên chip vi xử lý tăng gấp đôi mỗi hai năm, từ đó giúp tăng hiệu suất và giảm chi phí thiết bị điện tử.

Về các yếu tố ảnh hưởng đến năng lượng tiêu thụ, có thể kể đến hệ số hoạt động, điện dung tải, điện áp nguồn và tần số hoạt động. Hệ số hoạt động phản ánh tỷ lệ thời gian mà các bóng bán dẫn ở trạng thái hoạt động so với trạng thái nghỉ.

A graph with a line

AI-generated content may be incorrect.

Hình 2.1: Định Luật Moore

## 2.2. Định luật Dennard Scaling

Theo định luật Dennard Scaling, khi kích thước bóng bán dẫn giảm, mật độ tích hợp trên chip sẽ tăng lên, cho phép nhiều bóng bán dẫn hơn được đặt trong cùng một diện tích. Đồng thời, công suất tiêu thụ trên mỗi bóng bán dẫn giảm xuống do điện áp hoạt động và dòng rò giảm theo tỷ lệ thuận với quá trình thu nhỏ kích thước. Ngoài ra, tần số hoạt động của vi mạch cũng tăng lên nhờ thời gian chuyển mạch nhanh hơn, giúp cải thiện hiệu suất tổng thể của bộ vi xử lý.

### 2.2.1. Giới hạn và Thay đổi

Đến khoảng năm 2005, định luật Dennard Scaling bắt đầu mất hiệu lực do những hạn chế vật lý không thể khắc phục. Kích thước transistor thu nhỏ đến mức điện áp không thể giảm thêm mà vẫn đảm bảo hoạt động ổn định, đồng thời dòng điện rò tăng đáng kể gây lãng phí năng lượng. Điều này dẫn đến "Power Wall" - rào cản nhiệt và công suất ngăn cản việc tiếp tục tăng tần số hoạt động.

### 2.2.2. Giải pháp

Để duy trì đà tăng hiệu suất khi không thể tiếp tục thu nhỏ transistor, ngành công nghiệp bán dẫn đã chuyển hướng sang các giải pháp mới. Kiến trúc đa lõi trở thành xu hướng chủ đạo, cho phép tăng hiệu năng thông qua xử lý song song thay vì phụ thuộc vào tăng xung nhịp đơn lõi. Bên cạnh đó, các bộ xử lý ngày càng được chuyên môn hóa với những thành phần như bộ tăng tốc AI, GPU và TPU, tối ưu hóa cho các tác vụ cụ thể.

A graph showing different colored lines

AI-generated content may be incorrect.

Hình 2.2: Định Luật Scaling

2.3. Bộ gia tốc (accelerator)

Bộ gia tốc là phần cứng chuyên dụng giúp tăng tốc tính toán nặng như GPU, TPU, FPGA, ASIC. Khi thiết kế, cần xem xét 4 yếu tố:

1. Trạng thái:

Hệ thống có thể hoạt động ở hai chế độ: cục bộ và toàn cục. Ở chế độ cục bộ, dữ liệu được xử lý độc lập giúp giảm thiểu độ trễ. Trong khi đó, chế độ toàn cục cho phép chia sẻ bộ nhớ với CPU, thuận tiện cho giao tiếp nhưng có thể ảnh hưởng đến hiệu suất tổng thể.

2. Thao tác dữ liệu:

Các thiết bị phần cứng thường tích hợp những lệnh đặc biệt để tối ưu hóa xử lý. Ví dụ điển hình bao gồm lệnh nhân ma trận trên TPU hay các thao tác SIMD trên GPU, giúp tăng tốc độ tính toán đáng kể cho các tác vụ chuyên sâu.

3. Truy cập bộ nhớ:

Việc truy cập bộ nhớ chung đòi hỏi cơ chế đồng bộ hóa chặt chẽ. Để tối ưu hiệu suất, hệ thống thường sử dụng kết hợp bộ đệm và bộ nhớ cục bộ, giảm thiểu thời gian chờ truy xuất dữ liệu.

4. Luồng điều khiển:

Các thiết bị có thể hoạt động độc lập hoặc chịu sự điều khiển từ CPU. Trong khi GPU xử lý song song các luồng dữ liệu, một số bộ gia tốc chuyên dụng như xử lý video lại cần CPU điều phối hoạt động.

5. Phân loại hoạt động:

Hệ thống được chia thành hai loại chính: đồng bộ và không đồng bộ. Các thiết bị đồng bộ được CPU trực tiếp điều khiển và tích hợp chặt chẽ, trong khi thiết bị không đồng bộ chạy độc lập với sự can thiệp tối thiểu từ CPU. Việc lựa chọn phương thức phụ thuộc nhiều yếu tố như yêu cầu độ trễ, kích thước dữ liệu và khả năng song song hóa.

## 2.4. Trình tự hoạt động (Operation Sequencing)

1. Kiến trúc temporal (điều khiển theo thời gian):

Kiến trúc temporal sử dụng luồng lệnh động và bộ nhớ chia sẻ như trong CPU và GPU, phụ thuộc vào hệ thống phân cấp bộ nhớ để xử lý đa dạng các tác vụ khác nhau. Cách tiếp cận này linh hoạt nhưng có thể gặp hạn chế về hiệu suất do phụ thuộc vào bộ nhớ chung.

2. Kiến trúc spatial (điều khiển theo không gian):

Kiến trúc spatial bao gồm nhiều phần tử xử lý (PE) nhỏ kết nối trực tiếp với nhau mà không cần bộ nhớ chia sẻ, thường thấy trong FPGA và ASIC. Thiết kế này tối ưu cho thông lượng cao và giảm thiểu truy cập bộ nhớ nhờ khả năng xử lý song song và định tuyến dữ liệu trực tiếp giữa các PE.

|  |  |
| --- | --- |
| Hình 2.3: Kiến trúc Temporal | Hình 2.4: Kiến trúc Spatial |

## 2.5. Kiến Trúc Spatial cho Deep Learning

Kiến trúc không gian gồm ba thành phần cốt lõi: Các phần tử xử lý (PE) - đơn vị tính toán cơ bản với bộ thanh ghi cục bộ dung lượng nhỏ (0.5-1.0 kB) cho phép xử lý song song hiệu quả. Bộ đệm toàn cục (Global Buffer) có dung lượng lớn hơn (100-500 kB) đóng vai trò lưu trữ dữ liệu dùng chung và Mạng liên kết trực tiếp giữa các PE giúp giảm thiểu độ trễ truyền dữ liệu.

Kiến trúc này mang lại ưu điểm vượt trội về hiệu suất, đặc biệt trong việc giảm 50-90% số lần truy cập bộ nhớ so với kiến trúc thời gian (temporal), nhờ khả năng tái sử dụng dữ liệu cục bộ và cơ chế truyền dữ liệu trực tiếp giữa các PE mà không phải thông qua bộ nhớ chung.

A diagram of a computer

AI-generated content may be incorrect.

Hình 2.5: Kiến trúc Spatial dành cho CNN

### 2.5.1. Các khối logic cấu hình (Configurable Logic Blocks - CLB)

CLB (Configurable Logic Block) là đơn vị xử lý cơ bản trong kiến trúc FPGA, có khả năng triển khai logic tổ hợp lẫn logic tuần tự. Mỗi CLB bao gồm các thành phần chính:

1. BLE (Basic Logic Elements): Đơn vị logic gồm.
2. LUT (Look-Up Table): Thực hiện các hàm logic thông qua cơ chế tra cứu bảng giá trị đã được lập trình sẵn.
3. Thanh ghi (Registers): Lưu trữ trạng thái tạm thời phục vụ các mạch tuần tự.
4. Bộ chọn kênh (Multiplexer): Điều khiển tín hiệu và lựa chọn đầu ra.

Cấu trúc module hóa này cho phép CLB linh hoạt thực hiện đa dạng các chức năng logic, từ các phép toán đơn giản đến những mạch điều khiển phức tạp, tạo nên tính chất lập trình được của FPGA. Các CLB kết hợp với nhau thông qua mạng liên kết có thể cấu hình tạo thành hệ thống xử lý hoàn chỉnh.

### 2.5.2. Ứng dụng

Configurable Logic Blocks (CLB) giúp FPGA linh hoạt trong thiết kế phần cứng, ứng dụng trong xử lý tín hiệu số, hệ thống nhúng và các giải pháp tùy chỉnh.

A diagram of a clock

Description automatically generated

Hình 2.6: Khối logic có thể cấu hình

## 2.6. Khối không đồng nhất (Heterogeneous Blocks) trong FPGA

FPGA hiện đại tích hợp các khối chức năng chuyên biệt nhằm tối ưu hiệu năng hệ thống. Cấu trúc này bao gồm: Specific Purpose Logic cho các tác vụ đặc thù; Soft fabric gồm LUT, flip-flop và các phép toán cơ bản; Khối bộ nhớ BRAM với khả năng cấu hình linh hoạt kích thước word và địa chỉ; Bộ nhân/MAC cho xử lý tín hiệu số; và CPU cùng các phần tử xử lý tích hợp.

Kiến trúc không đồng nhất mang lại bốn ưu điểm chính: Tối ưu hiệu năng theo từng ứng dụng cụ thể; Tiết kiệm diện tích chip và năng lượng tiêu thụ; Nâng cao tính linh hoạt trong thiết kế hệ thống; Đồng thời cải thiện đáng kể hiệu suất tổng thể. Cách tiếp cận cho phép FPGA cân bằng giữa hiệu năng chuyên biệt và tính lập trình đa dụng.

A diagram of different colors of memory blocks

Description automatically generated with medium confidence

Hình 2.7: Khối không đồng nhất

## 2.7. Accelerator Taxonomy

Các hệ thống gia tốc tính toán hiện đại có thể được phân thành hai nhóm chính dựa trên phương thức tổ chức và vận hành.

Kiến trúc lập trình theo thời gian (Temporally Programmed) như CPU và GPU hoạt động dựa trên nguyên lý thực thi lệnh tuần tự. Các hệ thống này sử dụng bộ nhớ chia sẻ và tập trung vào khả năng xử lý đa dạng các loại tác vụ khác nhau. Ưu điểm chính của phương pháp này nằm ở tính linh hoạt cao, cho phép thích ứng với nhiều loại ứng dụng.

Kiến trúc lập trình theo không gian (Spatially Programmed) được chia thành hai phân nhóm nhỏ hơn. Phân nhóm Fine Grained tiêu biểu là FPGA với các khối logic cỡ nhỏ (LUT) có thể lập trình chi tiết đến từng bit, mang lại độ tùy biến cực cao. Phân nhóm Coarse Grained bao gồm các kiến trúc như RAW, AsAP và PicoChip sử dụng các phần tử xử lý lớn hơn với cơ chế triggered instructions, cân bằng tốt giữa hiệu suất tính toán và mức tiêu thụ năng lượng.

Sự khác biệt chính giữa hai phương pháp này thể hiện ở khả năng tối ưu hóa hiệu suất. Các kiến trúc không gian thường đạt hiệu suất cao hơn nhờ khả năng song song hóa mạnh mẽ và giảm thiểu đáng kể các thao tác truy cập bộ nhớ. Trong khi đó, kiến trúc thời gian duy trì ưu thế về tính linh hoạt trong lập trình và khả năng thích ứng. Việc lựa chọn kiến trúc phù hợp cần dựa trên các yêu cầu cụ thể về hiệu năng, độ trễ và tính linh hoạt của từng ứng dụng cụ thể.

### 2.7.1. Coarse (ALU) Grained

Kiến trúc coarse-grained sử dụng các đơn vị tính toán lớn (ALU) để đạt hiệu suất cao và tiết kiệm năng lượng. Có thể phân loại thành 4 nhóm chính:

1. Fixed-operation: Bao gồm các bộ gia tốc chuyên dụng như TPU và NVDLA, được thiết kế cứng cho các tác vụ cụ thể (nhân ma trận, tích chập), mang lại hiệu suất tối ưu nhưng kém linh hoạt.
2. Configured-operation: Gồm các kiến trúc như WARP, DySER, TRIPs, WaveScalar và TTA, cho phép cấu hình lại chức năng ALU theo yêu cầu ứng dụng, cân bằng giữa hiệu suất và tính linh hoạt.
3. PC-based: Sử dụng bộ đếm chương trình (PC) để điều khiển luồng lệnh, đại diện bởi Wave, RAW, AsAP và PicoChip. Phù hợp cho ứng dụng cần linh hoạt cao.
4. Triggered Operations: Dựa trên cơ chế kích hoạt lệnh (Triggered Instructions), tối ưu cho các tác vụ xử lý luồng dữ liệu với độ trễ thấp.

# DATAFLOW

## 3.1 Kiến trúc tính toán không gian và phân cấp bộ nhớ

### 3.1.1 Mô hình tính toán không gian (Spatial Compute Paradigm)

Mô hình tính toán không gian (Spatial Compute Paradigm) là một kiến trúc máy tính tiên tiến được thiết kế để tối ưu hóa việc xử lý song song và quản lý luồng dữ liệu hiệu quả. Kiến trúc này bao gồm ba thành phần chính: phân cấp bộ nhớ (Memory Hierarchy), ma trận các đơn vị logic số học (ALU - Arithmetic Logic Units), và luồng dữ liệu (Dataflow).

****

Hình 3.1: Tổ chức ma trận ALU và phân cấp bộ nhớ

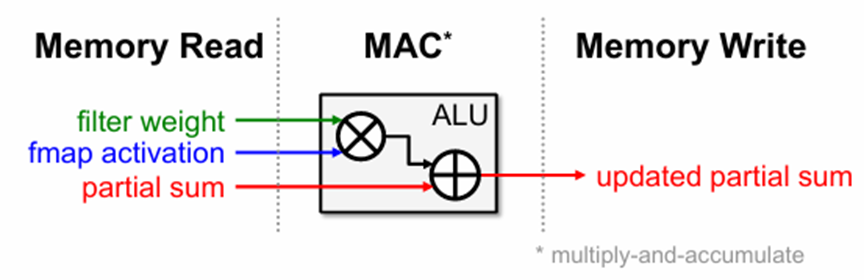
Phân cấp bộ nhớ được kết nối với tất cả các ALU trong hàng đầu tiên thông qua các mũi tên hai chiều, cho phép truy cập dữ liệu hai chiều giữa bộ nhớ và các đơn vị xử lý. Ma trận ALU được sắp xếp theo lưới 4x4, bao gồm 16 đơn vị tính toán, với mỗi ALU được kết nối với các ALU lân cận theo chiều dọc. Thiết kế này mang lại những ưu điểm quan trọng như khả năng xử lý song song nhiều phép tính, truyền dữ liệu hiệu quả giữa các ALU, và tối ưu hóa luồng dữ liệu trong ma trận.

Luồng dữ liệu trong mô hình này cho phép dữ liệu di chuyển theo chiều dọc giữa các ALU, giữa bộ nhớ và hàng ALU đầu tiên. Kiến trúc này đặc biệt phù hợp cho thuật toán xử lý ma trận, tính toán song song, và các ứng dụng xử lý dữ liệu quy mô lớn. Các lĩnh vực như xử lý hình ảnh, học máy, và các phép tính khoa học sẽ được hưởng lợi đáng kể từ mô hình tính toán không gian này, nhờ khả năng xử lý song song và di chuyển dữ liệu linh hoạt.

Trọng tâm của mô hình là tận dụng tối đa tính song song và tối ưu hóa luồng dữ liệu, góp phần giải quyết những thách thức trong việc xử lý các tập dữ liệu lớn và phức tạp. Bằng cách tổ chức các đơn vị tính toán và bộ nhớ theo một cấu trúc linh hoạt và hiệu quả, mô hình tính toán không gian mở ra những khả năng mới trong việc tăng tốc độ và hiệu quả của các hệ thống máy tính hiện đại.

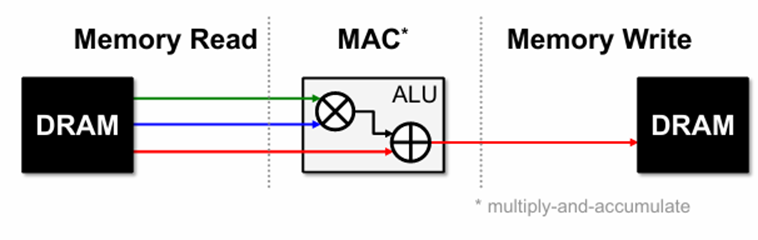
### 3.1.2. Truy xuất bộ nhớ là nút thắt cổ chai

Trong các hệ thống tính toán hiện đại, đặc biệt là các hệ thống xử lý neural networks, việc truy xuất bộ nhớ được chứng minh là một nút thắt cổ chai quan trọng ảnh hưởng đến hiệu năng tổng thể. Để minh họa vấn đề này, chúng ta xem xét hoạt động của các phép tính MAC (Multiply-Accumulate), một thao tác cơ bản trong các mạng neural.



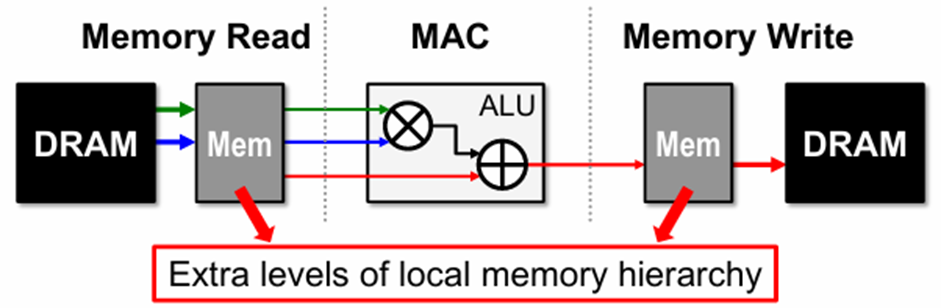
Hình 3.2: Sơ đồ khối phép tính MAC

Một phép MAC điển hình bao gồm ba đầu vào chính: trọng số bộ lọc (filter weight), dữ liệu kích hoạt (fmap activation), và tổng từng phần (partial sum). Đơn vị Arithmetic Logic Unit (ALU) sẽ thực hiện phép nhân (⊗) và phép cộng (⊕) để tạo ra một giá trị tổng từng phần được cập nhật. Tuy nhiên, vấn đề nằm ở chỗ mỗi lần thực hiện phép tính đều yêu cầu truy cập vào bộ nhớ Dynamic Random-Access Memory (DRAM), và chi phí truy cập này là rất lớn.



Hình 3.3: Sơ đồ Truy xuất bộ nhớ và phép tính MAC

Để minh họa mức độ nghiêm trọng, hãy xem xét mạng AlexNet - một mạng neural nổi tiếng. Với 724 triệu phép MAC, hệ thống phải thực hiện gần 2.896 tỷ lần truy cập DRAM. Mỗi lần truy cập DRAM không chỉ tốn thời gian mà còn tiêu tốn một lượng năng lượng đáng kể. Trong trường hợp xấu nhất, mọi thao tác đọc/ghi đều phải qua DRAM, gây ra sự chậm chạp và lãng phí tài nguyên tính toán.



Hình 3.4: Sơ đồ phân cấp Bộ nhớ Cục bộ Bổ sung

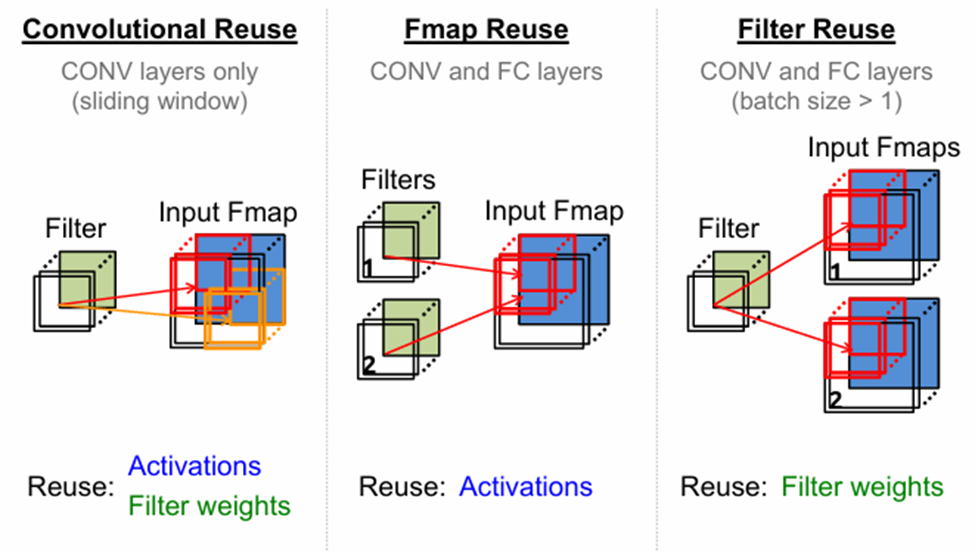
Giải pháp được đề xuất là sử dụng một hệ thống bộ nhớ phân cấp, bằng cách thêm các mức bộ nhớ cục bộ (local memory) giữa DRAM và các đơn vị ALU. Phương pháp này đặc biệt hiệu quả khi cường độ tính toán lớn hơn 1, cho phép giảm số lần truy cập trực tiếp vào DRAM thông qua việc tái sử dụng dữ liệu (data reuse).

Ý tưởng chính của phương pháp này là lưu trữ các dữ liệu đã đọc từ DRAM trong bộ nhớ cục bộ để tái sử dụng, từ đó giảm thiểu số lần truy cập DRAM. Bằng cách này, hệ thống có thể tối ưu hóa việc sử dụng bộ nhớ, giảm thiểu chi phí năng lượng và tăng tốc độ xử lý, đặc biệt là trong các ứng dụng neural networks với số lượng phép tính MAC rất lớn.

Tóm lại, vấn đề "truy xuất bộ nhớ là nút thắt cổ chai" không chỉ là một thách thức kỹ thuật đơn thuần, mà còn là một điểm then chốt ảnh hưởng trực tiếp đến hiệu năng của các hệ thống tính toán hiện đại. Việc hiểu và giải quyết vấn đề này là quan trọng trong quá trình thiết kế và tối ưu hóa các hệ thống xử lý dữ liệu phức tạp.

### 3.1.3. Các loại tái sử dụng dữ liệu trong mạng neural sâu (DNN)

Trong các mạng neural sâu, việc tối ưu hóa sử dụng dữ liệu là một yếu tố then chốt để nâng cao hiệu suất tính toán và giảm thiểu chi phí năng lượng. Nghiên cứu đã xác định ba chiến lược tái sử dụng dữ liệu chính: Tái sử dụng Tích chập (Convolutional Reuse), Tái sử dụng Feature Map (Fmap Reuse), và Tái sử dụng Bộ lọc (Filter Reuse).



Hình 3.5: Sơ đồ minh họa các phương pháp tái sử dụng dữ liệu

Tái sử dụng Tích chập (Convolutional Reuse) là phương pháp được áp dụng riêng cho các lớp tích chập (CONV), sử dụng kỹ thuật cửa sổ trượt (Sliding Window). Phương pháp này tái sử dụng đồng thời hai thành phần: activation (kích hoạt) và filter weights (trọng số bộ lọc). Cụ thể, một bộ lọc được áp dụng nhiều lần trên các vùng khác nhau của Feature Map đầu vào, giúp tối ưu hóa việc sử dụng cả dữ liệu kích hoạt và trọng số.

Tái sử dụng Feature Map (Fmap Reuse) có phạm vi áp dụng rộng hơn, bao gồm cả các lớp tích chập (CONV) và lớp kết nối đầy đủ (Fully Connected - FC). Phương pháp này chỉ tập trung vào việc tái sử dụng các activation. Nhiều bộ lọc khác nhau có thể cùng sử dụng một vùng Feature Map đầu vào, giúp tận dụng tối đa dữ liệu đầu vào cho nhiều phép tính khác nhau.

Tái sử dụng Bộ lọc (Filter Reuse) cũng được áp dụng cho cả lớp CONV và FC, với điều kiện tiên quyết là kích thước batch phải lớn hơn 1. Phương pháp này chỉ tái sử dụng trọng số bộ lọc (filter weights), trong đó cùng một bộ lọc được áp dụng cho nhiều Feature Map đầu vào khác nhau. Điều này giúp tối ưu hóa việc sử dụng bộ lọc trên nhiều mẫu dữ liệu.

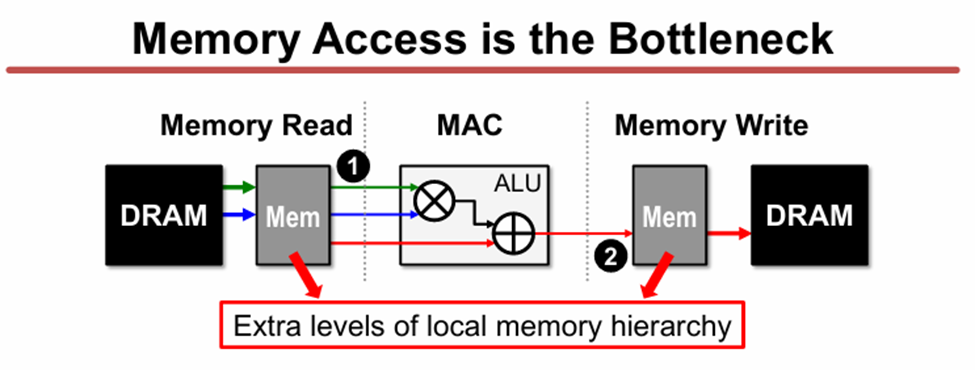
Những phương pháp tái sử dụng dữ liệu này mang lại nhiều lợi ích quan trọng. Thứ nhất, chúng giảm đáng kể số lần truy cập bộ nhớ, giúp tiết kiệm năng lượng và tăng tốc độ tính toán. Thứ hai, chúng tăng hiệu quả sử dụng dữ liệu, cho phép tận dụng tối đa các giá trị đã được nạp vào hệ thống. Thứ ba, chúng cải thiện hiệu năng tính toán bằng cách giảm thiểu việc di chuyển dữ liệu giữa các module.

Mỗi phương pháp tái sử dụng đều phù hợp với các tình huống khác nhau và có thể được kết hợp để tối ưu hóa hiệu suất của mạng neural. Việc lựa chọn và áp dụng phương pháp phù hợp phụ thuộc vào kiến trúc mạng, kích thước dữ liệu, và các ràng buộc về năng lượng và hiệu năng của hệ thống.

Trong bối cảnh phát triển không ngừng của các hệ thống học sâu, các kỹ thuật tái sử dụng dữ liệu đóng vai trò ngày càng quan trọng. Chúng không chỉ là giải pháp kỹ thuật, mà còn là yếu tố then chốt trong việc phát triển các hệ thống tính toán hiệu quả, tiết kiệm năng lượng và có khả năng mở rộng.

### 3.1.4. Phương pháp cải thiện hiệu năng truy cập DRAM

Quay trở lại vấn đề nghẽn cổ chai trong truy xuất bộ nhớ, các nhà nghiên cứu đã phát triển hai giải pháp then chốt để giải quyết thách thức hiệu năng: tái sử dụng dữ liệu (Data Reuse) và tích lũy cục bộ (Local Accumulation).



Hình 3.6: Sơ đồ minh họa nút thắt cổ chai trong truy cập bộ nhớ

Giải pháp đầu tiên - Tái sử dụng dữ liệu - tập trung vào việc giảm thiểu số lần đọc từ DRAM. Nghiên cứu trên mạng AlexNet đã chứng minh khả năng giảm số lần đọc filter/feature map lên đến 500 lần. Phương pháp này sử dụng bộ nhớ cục bộ (Mem) như một bộ đệm giữa DRAM và ALU, cho phép lưu trữ và tái sử dụng dữ liệu đã được đọc.

Giải pháp thứ hai - Tích lũy cục bộ - mở rộng ý tưởng trên bằng cách quản lý các tổng từng phần (partial sums) một cách thông minh. Các partial sum được tích lũy trực tiếp trong bộ nhớ cục bộ, loại bỏ nhu cầu truy cập liên tục DRAM cho quá trình tích lũy. Kết quả là việc giảm số lần truy cập DRAM trở nên đáng kể.

Kết quả ấn tượng được ghi nhận từ việc áp dụng hai giải pháp này trên mạng AlexNet: từ ban đầu phải thực hiện 2.896 triệu lần truy cập DRAM, hệ thống đã giảm xuống còn chỉ 61 triệu lần truy cập trong trường hợp tốt nhất. Con số này không chỉ thể hiện sự cải thiện về hiệu năng tính toán mà còn mang lại những lợi ích đáng kể về mặt năng lượng.

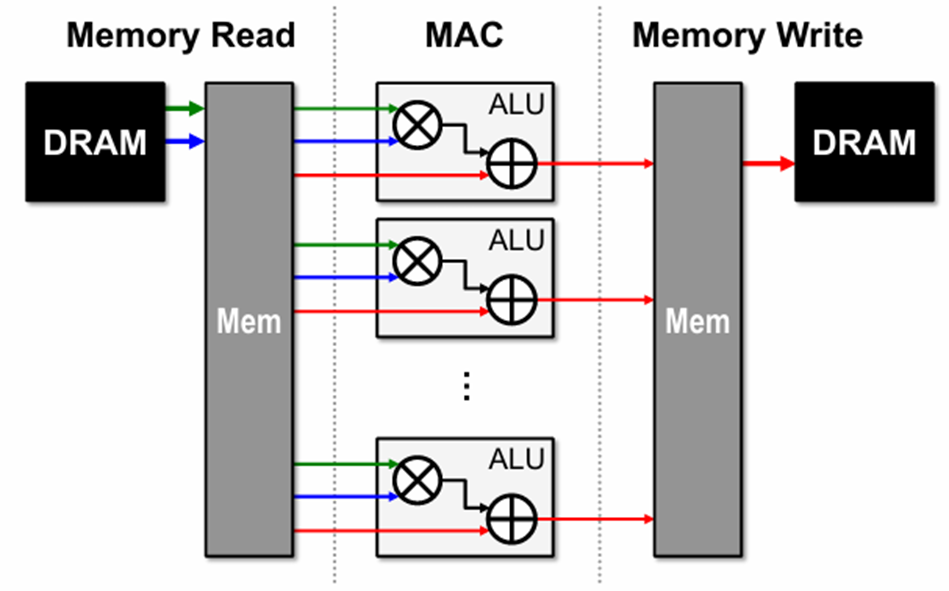
Cải thiện hiệu suất đạt được thông qua ba yếu tố chính: tái sử dụng dữ liệu hiệu quả, tích lũy cục bộ thông minh và một hệ thống phân cấp bộ nhớ được thiết kế hợp lý. Các phương pháp này không chỉ giải quyết vấn đề nghẽn cổ chai trong truy cập DRAM mà còn mở ra hướng tiếp cận mới trong tối ưu hóa kiến trúc máy tính cho các ứng dụng neural networks.

Ý nghĩa sâu xa của nghiên cứu này nằm ở chỗ: không phải lúc nào việc nâng cấp phần cứng cũng là giải pháp duy nhất. Thay vào đó, việc tinh chỉnh các thuật toán, tối ưu hóa luồng dữ liệu và sử dụng thông minh các tài nguyên sẵn có có thể mang lại những cải tiến về hiệu năng và hiệu quả năng lượng.

### 3.1.5 Leverage Parallelism - Tận dụng song song

#### 3.1.5.1. Higher performance - Cải thiện hiệu suất

Việc tận dụng sự song song (Leverage Parallelism) được coi là một trong những chiến lược quan trọng nhất để nâng cao hiệu suất tính toán trong các hệ thống xử lý hiện đại. Phương pháp này tập trung vào việc phân tán và song song hóa các hoạt động tính toán, từ đó tối đa hóa năng lực xử lý của hệ thống.



Hình 3.7: Sơ đồ tận dụng song song các Module ALU và bộ nhớ

Trong mô hình kiến trúc máy tính tiên tiến, việc song song hóa được thể hiện qua việc sắp xếp các module bộ nhớ cục bộ (Mem) và các đơn vị tính toán ALU theo cấu trúc ngang hàng. Ở giai đoạn đọc bộ nhớ (Memory Read), các module Mem và ALU được bố trí song song, cho phép thực hiện nhiều phép tính đồng thời trên các module ALU khác nhau.

Điều này có nghĩa là thay vì xử lý tuần tự, hệ thống có thể thực hiện nhiều phép tính độc lập cùng một lúc. Ưu điểm của phương pháp này là tận dụng tối đa sức mạnh song song của hệ thống, giảm thiểu thời gian chờ đợi và tăng tốc độ xử lý dữ liệu. Tương tự, trong giai đoạn ghi bộ nhớ (Memory Write), các module Mem và DRAM cũng được sắp xếp ngang hàng, cho phép đồng thời ghi dữ liệu vào nhiều vị trí DRAM.

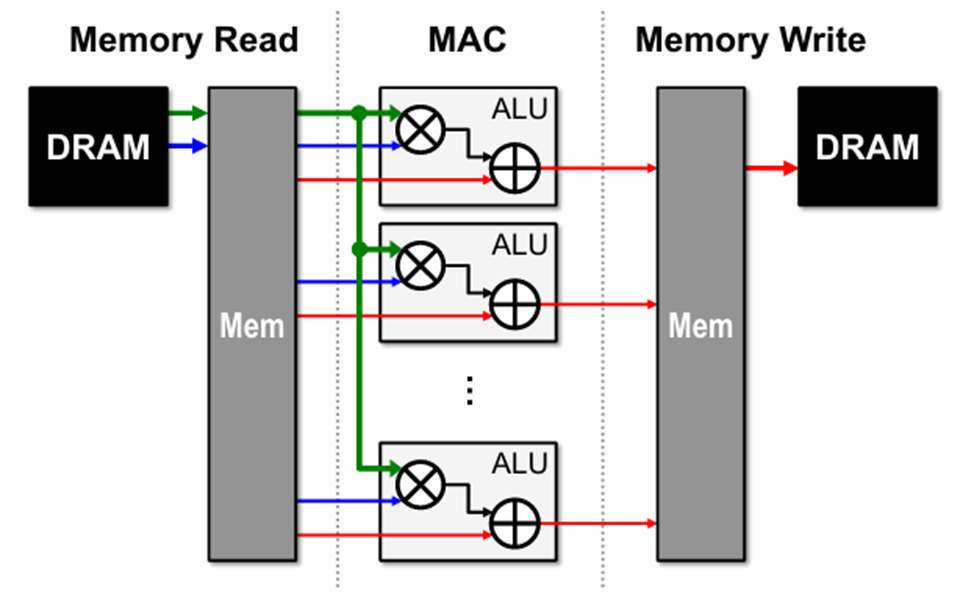
Trọng tâm của phương pháp này không chỉ đơn thuần là tăng tốc độ xử lý, mà còn là tối đa hóa lợi ích của việc tính toán song song, bất kể dữ liệu đầu vào hay đầu ra. Bằng cách phân tán các tác vụ tính toán, hệ thống có thể xử lý khối lượng lớn dữ liệu một cách hiệu quả hơn, đặc biệt là trong các ứng dụng đòi hỏi xử lý song song như học sâu, xử lý hình ảnh và các mô hình tính toán phức tạp.

Việc áp dụng song song hóa đã và đang revolutionize các hệ thống tính toán, cho phép chúng vượt qua những giới hạn của kiến trúc tuần tự truyền thống. Nó không chỉ là một kỹ thuật tối ưu hóa đơn thuần, mà còn là một phương thức tư duy mới trong việc thiết kế và triển khai các hệ thống máy tính hiện đại.

Tuy nhiên, việc triển khai song song hóa không phải là không có thách thức. Các nhà thiết kế hệ thống phải giải quyết các vấn đề phức tạp như đồng bộ hóa dữ liệu, quản lý tài nguyên và giảm thiểu các điểm nghẽn. Nhưng với sự phát triển không ngừng của công nghệ, những thách thức này đang được giải quyết ngày càng hiệu quả.

#### 3.1.5.2. Spatial Data Reuse - Tối ưu tái dữ liệu không gian

Spatial Data Reuse là một phương pháp tiên tiến nhằm tối ưu hóa việc sử dụng dữ liệu trong các hệ thống tính toán hiện đại, đặc biệt là trong các kiến trúc xử lý song song. Phương pháp này tập trung vào việc tận dụng tối đa các nguồn dữ liệu đầu vào và các giá trị tích lũy một cách hiệu quả nhất trên các module tính toán.



Hình 3.8: Sơ đồ tối ưu hóa tái sử dụng dữ liệu không gian

Trong kiến trúc tiên tiến, các module ALU được sắp xếp ngang hàng, tạo ra một môi trường thuận lợi cho việc thực hiện nhiều phép tính song song. Điều quan trọng là các module này không chỉ đơn thuần thực hiện các phép tính độc lập, mà còn có khả năng chia sẻ và tái sử dụng dữ liệu một cách thông minh.

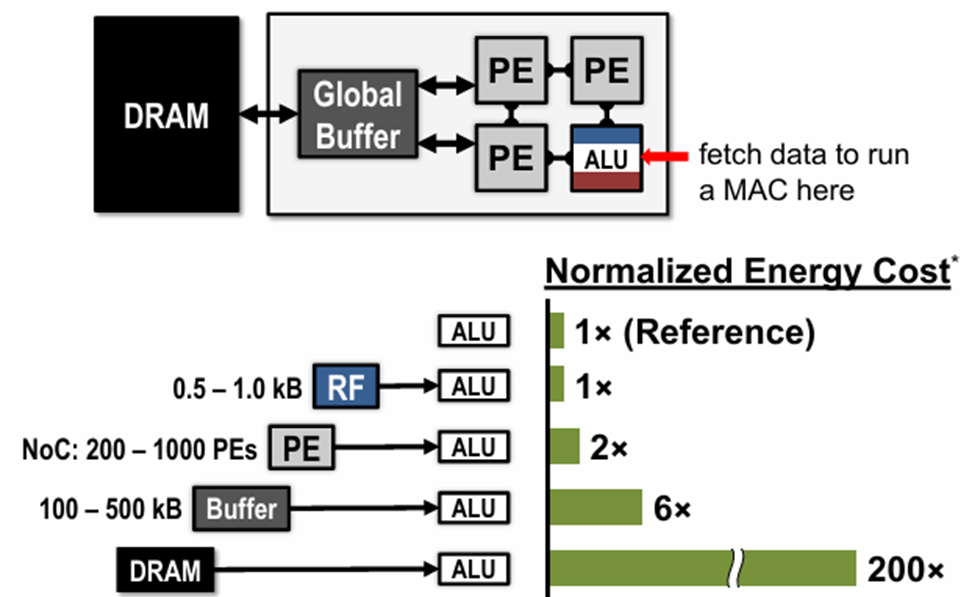
Cụ thể, phương pháp này cho phép tận dụng tối đa việc tái sử dụng dữ liệu đầu vào (input) và các giá trị tích lũy từng phần (partial sum) trên toàn bộ mảng các module ALU. Điều này có nghĩa là một khối dữ liệu đầu vào có thể được sử dụng cho nhiều phép tính khác nhau, giảm thiểu truy cập bộ nhớ liên tục.

Ưu điểm chính của Spatial Data Reuse là khả năng giảm số lần truy cập DRAM, qua đó cải thiện đáng kể hiệu suất tính toán. Bằng cách tối ưu hóa việc di chuyển và sử dụng lại dữ liệu, hệ thống có thể thực hiện nhiều phép tính phức tạp với chi phí năng lượng và thời gian thấp hơn.

Trọng tâm của phương pháp này là tối ưu hóa việc tái sử dụng dữ liệu trong không gian, đảm bảo rằng mỗi bit dữ liệu được sử dụng một cách hiệu quả nhất. Điều này đặc biệt quan trọng trong các ứng dụng đòi hỏi xử lý khối lượng lớn dữ liệu như học sâu, xử lý hình ảnh và các mô hình tính toán phức tạp.

**3.1.6. Truy cập dữ liệu cục bộ giá thấp**

Trong bối cảnh phát triển của các hệ thống tính toán hiện đại, việc truy cập dữ liệu hiệu quả và tiết kiệm năng lượng trở thành một yếu tố then chốt. Nghiên cứu về truy cập dữ liệu cục bộ giá thấp tập trung vào việc tối ưu hóa các bộ đệm (buffer) nằm giữa DRAM và các đơn vị xử lý ALU, nhằm giảm thiểu chi phí năng lượng và tăng hiệu suất tính toán.



Hình 3.9: Phân tích chi phí năng lượng theo cấu trúc bộ nhớ

Bộ đệm toàn cục (Global Buffer) được thiết kế như một giải pháp trung gian thông minh, cho phép lấy dữ liệu cho các phép tính MAC (Multiply-and-Accumulate) từ một nguồn gần hơn và tiết kiệm năng lượng hơn so với việc truy cập trực tiếp DRAM. So sánh năng lượng cho thấy những lợi ích đáng kể từ việc sử dụng bộ đệm cục bộ: chi phí năng lượng có thể giảm tới 6 lần so với truy cập trực tiếp, và trong một số trường hợp, việc tránh truy cập DRAM có thể tiết kiệm năng lượng lên đến 200 lần.

Câu hỏi then chốt được đặt ra là: Làm thế nào để khai thác tối đa việc tái sử dụng dữ liệu và tích lũy cục bộ với các bộ lưu trữ cục bộ chi phí thấp? Chìa khóa nằm ở việc phát triển một luồng xử lý dữ liệu đặc biệt, cho phép tận dụng tối đa các lợi ích từ tái sử dụng dữ liệu và tích lũy cục bộ.

Biểu đồ năng lượng không chỉ đơn thuần là những con số thống kê, mà còn là minh chứng sống động cho tiềm năng của việc tối ưu hóa truy cập bộ nhớ. Từ việc sử dụng các thanh ghi (Register File) đến việc quản lý các đơn vị xử lý (Processing Elements), mỗi cấp độ đều mang lại những cải thiện đáng kể về năng lượng và hiệu suất.

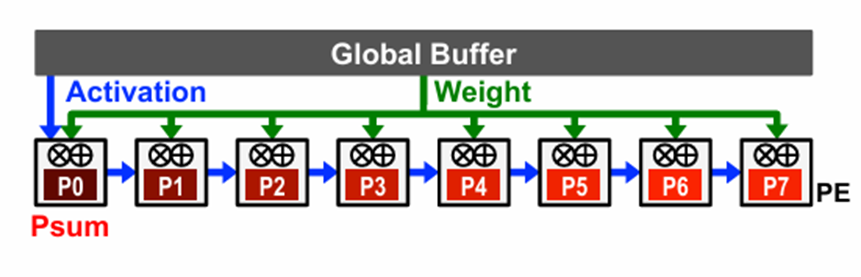
Các kỹ thuật kiến trúc và thuật toán tinh vi đã chứng minh khả năng giảm đáng kể chi phí năng lượng và cải thiện hiệu suất của việc truy cập bộ nhớ - một nút thắt cổ chai quan trọng trong các hệ thống tính toán. Việc tối ưu hóa tái sử dụng dữ liệu và tích lũy cục bộ được xem là những phương pháp có tác động lớn nhất.

Ý nghĩa sâu xa của nghiên cứu này nằm ở chỗ: công nghệ tiên tiến không chỉ đơn thuần là việc tăng cường năng lực phần cứng, mà còn là khả năng sử dụng thông minh và hiệu quả các nguồn lực sẵn có. Trong bối cảnh xu hướng điện toán xanh và hiệu quả năng lượng, những nghiên cứu như vậy đóng vai trò then chốt trong việc định hình tương lai của các hệ thống máy tính.

## 3.2 Các chiến lược luồng dữ liệu trong tính toán

### 3.2.1. Output Stationary (OS)

Output Stationary (OS) là một chiến lược dataflow tiên tiến được phát triển để tối ưu hóa việc thực hiện các phép tích chập (convolution) trên các kiến trúc phần cứng chuyên dụng cho học sâu. Chiến lược này được thiết kế nhằm nâng cao hiệu quả tính toán và tiết kiệm năng lượng trong quá trình xử lý các mạng neural.



Hình 3.10: Kiến trúc Output Stationary

Kiến trúc cơ bản của Output Stationary bao gồm ba thành phần chính: Bộ đệm toàn cục (Global Buffer), Các phần tử xử lý (Processing Elements - PE), và các giá trị tổng riêng phần (Partial Sums). Bộ đệm toàn cục đóng vai trò quan trọng trong việc lưu trữ và phân phối dữ liệu activation và weight tới các PE.

Nguyên lý hoạt động của Output Stationary rất độc đáo. Mỗi PE được gán trách nhiệm tính toán một phần tử đầu ra cụ thể. Các weights và activations được chuyển từ global buffer tới các PE, và điều đặc biệt là mỗi PE duy trì và tích lũy giá trị đầu ra của mình tại chỗ - do đó có tên gọi "stationary" (ổn định).

Ưu điểm chính của chiến lược này nằm ở hai khía cạnh. Thứ nhất, nó giảm thiểu tiêu thụ năng lượng cho việc đọc/ghi các giá trị tổng riêng phần bằng cách tối đa hóa việc tích lũy cục bộ. Các tổng riêng phần được tính toán và tích lũy ngay trong PE, loại bỏ nhu cầu di chuyển dữ liệu giữa các bộ nhớ.

Thứ hai, Output Stationary tận dụng hiệu quả kỹ thuật broadcast/multicast. Các trọng số bộ lọc được phát sóng tới nhiều PE, còn các giá trị activation được tái sử dụng không gian trên toàn bộ mảng PE thông qua Global Buffer.

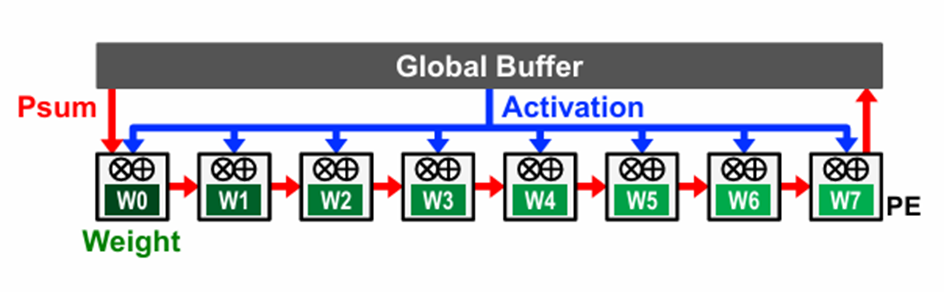
Đặc điểm "stationary" thể hiện ở chỗ giá trị đầu ra được giữ cố định tại một PE cụ thể. Mỗi PE chịu trách nhiệm tính toán hoàn chỉnh một phần tử đầu ra, trong khi các giá trị đầu vào và trọng số di chuyển, các giá trị tổng riêng phần được tích lũy tại chỗ.

Output Stationary đặc biệt hiệu quả trong một số trường hợp cụ thể: các lớp tích chập có kích thước filter lớn, hệ thống có giới hạn băng thông bộ nhớ, kiến trúc phần cứng có nhiều PE song song, và khi chi phí di chuyển các giá trị tổng riêng phần cao hơn so với việc di chuyển weights và activations.

Các ví dụ thực tế như kiến trúc ShiDianNao và nghiên cứu tại Đại học KU Leuven đã chứng minh tính ưu việt của chiến lược này trong việc thiết kế các bộ tăng tốc tích chập hiệu quả về năng lượng cho ứng dụng trí tuệ nhân tạo và học sâu.

### 3.2.2. Weight Stationary (WS)

Weight Stationary (WS) là một chiến lược dataflow tiên tiến khác trong việc tối ưu hóa xử lý các mạng neural, tập trung vào việc giảm thiểu chi phí năng lượng và tăng hiệu quả tính toán thông qua việc giữ cố định các trọng số trong quá trình xử lý.



Hình 3.11: Kiến trúc Weight Stationary

Kiến trúc của Weight Stationary bao gồm ba thành phần chính: Bộ đệm toàn cục (Global Buffer), Mảng các phần tử xử lý (PE Array), và các trọng số được cố định. Bộ đệm toàn cục đóng vai trò then chốt trong việc lưu trữ và quản lý dữ liệu activation và các giá trị tổng riêng phần (partial sums).

Nguyên lý hoạt động của Weight Stationary rất độc đáo. Mỗi phần tử xử lý (PE) được gán một trọng số filter cụ thể và giữ nguyên trọng số này trong suốt quá trình tính toán. Các giá trị activation được truyền qua mảng PE, trong khi các giá trị tổng riêng phần được tích lũy và di chuyển qua PE, cuối cùng quay trở lại global buffer.

Ưu điểm chính của chiến lược này nằm ở hai khía cạnh quan trọng. Thứ nhất, nó giảm thiểu năng lượng đọc trọng số bằng cách tối đa hóa việc tái sử dụng trọng số trong các phép tính chập. Mỗi trọng số được sử dụng nhiều lần, giảm thiểu số lần truy cập bộ nhớ.

Thứ hai, Weight Stationary tối ưu hóa việc phân phối dữ liệu. Các giá trị activation được phát sóng đến nhiều PE cùng lúc, trong khi các giá trị tổng riêng phần được tích lũy không gian trên toàn bộ mảng PE. Điều này giúp tận dụng tối đa cấu trúc song song của hệ thống.

Chiến lược này đặc biệt phù hợp với các mạng neural có các bộ lọc nhỏ được sử dụng nhiều lần. Nó đặc biệt hiệu quả trong các bộ tăng tốc có bộ nhớ on-chip giới hạn, và thường được sử dụng trong các kiến trúc như Tensor Processing Unit (TPU) và các hệ thống FPGA.

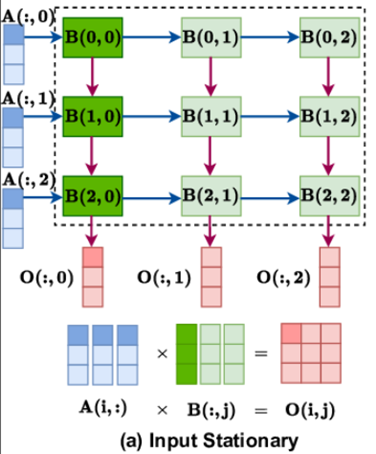
Việc áp dụng Weight Stationary mang lại những lợi ích đáng kể trong việc thiết kế các bộ tăng tốc neural network hiệu quả về năng lượng. Đặc biệt, trong các ứng dụng có giới hạn về băng thông bộ nhớ dành cho việc truy cập trọng số, chiến lược này cho thấy tiềm năng to lớn trong việc cải thiện hiệu suất tính toán.

Tuy nhiên, cũng cần lưu ý rằng hiệu quả của Weight Stationary phụ thuộc vào kiến trúc cụ thể của mạng neural và các ràng buộc về phần cứng. Không phải mọi kiến trúc đều phù hợp với chiến lược này, do đó việc lựa chọn và tinh chỉnh là rất quan trọng.

Trong bối cảnh phát triển không ngừng của các hệ thống xử lý trí tuệ nhân tạo, Weight Stationary được coi là một trong những chiến lược quan trọng giúp cầu nối giữa hiệu quả tính toán và hiệu quả năng lượng.

### 3.2.3 Input Stationary (IS)

Input Stationary là một chiến lược dataflow tiên tiến trong kiến trúc tăng tốc mạng neural tích chập, với đặc điểm độc đáo là giữ cố định dữ liệu đầu vào (input activations) trong quá trình tính toán, trong khi các trọng số và giá trị tổng riêng phần có thể di chuyển.



Hình 3.12: Kiến trúc Input Stationary

Đặc điểm chính của Input Stationary nằm ở việc duy trì và tái sử dụng các giá trị kích hoạt đầu vào, thay vì cố định đầu ra hoặc trọng số như các chiến lược khác. Phương pháp này được phát triển và nghiên cứu đặc biệt hiệu quả trong các mạng neural tích chập thưa (sparse Convolutional Neural Networks).

Lý do Input Stationary đặc biệt hiệu quả với các mạng CNN thưa là do đặc tính của chúng. Trong các mạng thưa, nhiều trọng số có giá trị bằng không, tạo ra những lợi thế tính toán độc đáo. Khi đầu vào thường lớn hơn trọng số và yêu cầu nhiều bộ nhớ hơn, việc giữ cố định input activation có thể giảm đáng kể số lần đọc từ bộ nhớ.

Nguyên lý hoạt động của Input Stationary khá đơn giản nhưng hiệu quả. Các đơn vị xử lý (Processing Elements - PE) sẽ nạp và giữ các giá trị đầu vào cố định. Các trọng số được đưa vào các PE để thực hiện các phép tính, và các giá trị tổng riêng phần (partial sums) được tạo ra và di chuyển qua các PE.

Ưu điểm chính của chiến lược này là khả năng giảm đáng kể năng lượng truy cập bộ nhớ, đặc biệt là khi kích thước đầu vào lớn hơn nhiều so với trọng số. Điều này đặc biệt hiệu quả trong các hệ thống có bộ nhớ đầu vào lớn (off-chip) và tiêu tốn nhiều năng lượng để truy cập.

Tuy nhiên, Input Stationary không phải là giải pháp vạn năng. Phương pháp này ít được nghiên cứu và áp dụng cho các mạng neural dày đặc (dense networks). Hiệu quả của nó phụ thuộc nhiều vào cấu trúc và đặc điểm của mạng neural cụ thể.

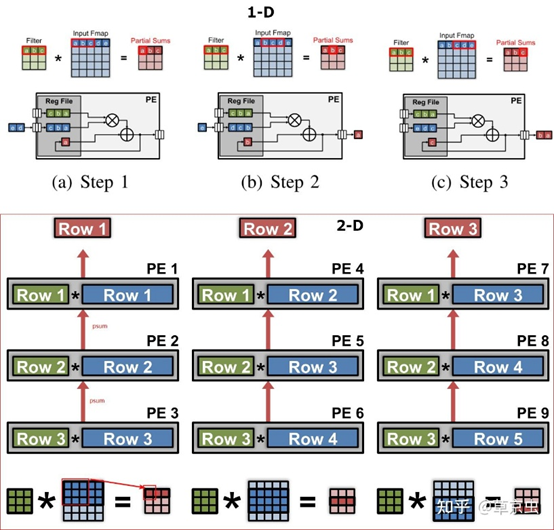
Nghiên cứu của Parashar et al. về SCNN (Sparse Convolutional Neural Network) tại hội nghị ISCA 2017 đã là một trong những công trình tiên phong chứng minh tiềm năng của Input Stationary. Nghiên cứu này đã mở ra những khả năng mới trong việc tối ưu hóa xử lý các mạng neural với cấu trúc thưa.

Trong bối cảnh không ngừng phát triển của các kiến trúc phần cứng cho trí tuệ nhân tạo, Input Stationary được coi là một trong những phương pháp đầy hứa hẹn. Nó không chỉ là một kỹ thuật tối ưu hóa đơn thuần, mà còn là một bước tiến quan trọng trong việc thiết kế các hệ thống tính toán hiệu quả và tiết kiệm năng lượng.

Tuy nhiên, việc áp dụng Input Stationary vẫn đòi hỏi những nghiên cứu sâu rộng hơn để khám phá toàn bộ tiềm năng của phương pháp này trong các kiến trúc máy học và xử lý song song hiện đại.

### 3.2.4. Luồng dữ liệu tiết kiệm năng lượng: Row Stationary (RS)

Row Stationary (RS) là một chiến lược dataflow tiên tiến được phát triển nhằm tối ưu hóa hiệu suất năng lượng tổng thể trong các bộ tăng tốc neural network. Khác biệt với các phương pháp truyền thống, RS không chỉ tập trung tối ưu hóa một loại dữ liệu cụ thể, mà hướng tới việc cải thiện toàn diện hiệu quả năng lượng của hệ thống.



Hình 3.13: Kiến trúc Row Stationary

Các đặc điểm chính của Row Stationary bao gồm hai nguyên tắc cơ bản. Thứ nhất, phương pháp ưu tiên tối đa hóa việc tái sử dụng và tích lũy dữ liệu tại các thanh ghi (Register File - RF). Đây là cấp bộ nhớ thấp nhất với chi phí năng lượng truy cập thấp nhất. Thiết kế của RS nhằm lưu trữ và tái sử dụng dữ liệu nhiều lần tại các register file.

Thứ hai, Row Stationary hướng tới tối ưu hóa hiệu quả năng lượng tổng thể bằng cách xem xét chi phí năng lượng cho tất cả các loại dữ liệu: trọng số (weights), đầu vào (inputs) và các giá trị tổng riêng phần (partial sums). Thay vì tối ưu hóa riêng lẻ từng loại dữ liệu như các phương pháp trước đây, RS đánh giá tổng chi phí năng lượng.

Nguyên lý hoạt động của Row Stationary được tổ chức theo "hàng" (rows) rất độc đáo. Cụ thể, một hàng của trọng số bộ lọc được giữ cố định trong register file. Tương tự, một hàng của feature map đầu vào (ifmap) cũng được duy trì ổn định. Các phép tính được thực hiện bằng cách tính toán các giá trị tổng riêng phần dọc theo một hàng của feature map đầu ra (ofmap).

Ưu điểm nổi bật của Row Stationary nằm ở tính linh hoạt và khả năng tối ưu hóa toàn diện. Trong khi các phương pháp trước đây thường tối ưu hóa riêng lẻ cho đầu ra, trọng số hay đầu vào, Row Stationary tạo ra một cách tiếp cận toàn diện hơn. Phương pháp này đặc biệt hiệu quả với các mạng neural đa dạng và các kiến trúc phần cứng cần song song hóa.

Trong bối cảnh phát triển không ngừng của các hệ thống học sâu, Row Stationary được coi là một trong những chiến lược đầy triển vọng. Nó không chỉ là một kỹ thuật tối ưu hóa, mà còn là một cách tiếp cận mới trong việc thiết kế các hệ thống tính toán hiệu quả, tiết kiệm năng lượng và có khả năng mở rộng.

### 3.2.5. So sánh với các luồng dữ liệu khác

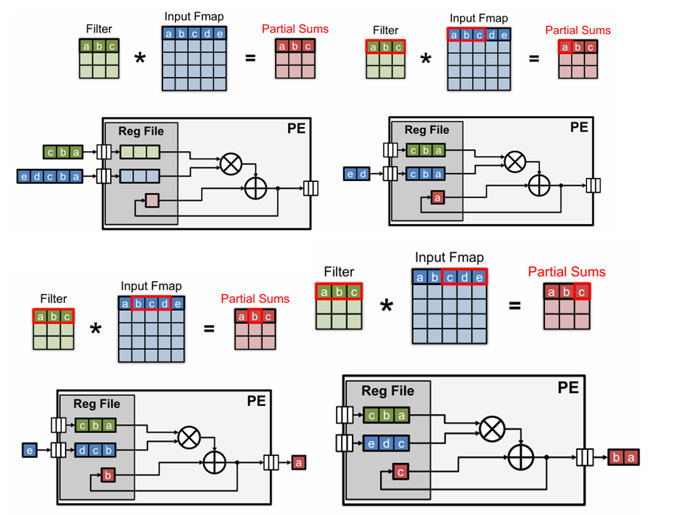
Bảng 3.1: Tiêu chí đánh giá các luồng dữ liệu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tiêu chí** | **Output  Stationary (OS)** | **Weight  Stationary (WS)** | **Input  Stationary (IS)** | **Row  Stationary (RS)** |
| **Định nghĩa** | Giữ kết quả đầu ra (output) cố định trong bộ nhớ, tính toán từng phần tử output từ input và weight. | Giữ trọng số (weight) cố định trong bộ nhớ, tái sử dụng weight cho nhiều phép tính. | Giữ đầu vào (input) cố định trong bộ nhớ, tái sử dụng input cho nhiều phép tính. | Tối ưu hóa theo hàng (row) của ma trận, mỗi hàng được xử lý song song và cục bộ. |
| **Cách tổ chức dữ liệu** | Output được lưu trữ lâu dài, input và weight di chuyển qua đơn vị tính toán. | Weight được lưu trữ lâu dài, input và output di chuyển qua đơn vị tính toán. | Input được lưu trữ lâu dài, weight và output di chuyển qua đơn vị tính toán. | Dữ liệu được chia theo hàng, tối ưu hóa tái sử dụng dữ liệu cục bộ. |
| **Ưu điểm** | - Giảm truy cập bộ nhớ cho output.  - Phù hợp với mạng tích chập (CNN). | - Tái sử dụng weight hiệu quả.  - Phù hợp với lớp fully-connected. | - Tái sử dụng input hiệu quả.  - Ít truy cập bộ nhớ cho input lớn. | - Tối ưu hóa song song hóa.  - Linh hoạt với nhiều loại mạng nơ-ron. |
| **Nhược điểm** | - Có thể tăng truy cập bộ nhớ cho input/weight.  - Phức tạp khi scale. | - Tăng truy cập bộ nhớ cho input/output.  - Hạn chế với input lớn. | - Tăng truy cập bộ nhớ cho weight/output.  - Ít hiệu quả với weight lớn. | - Phức tạp trong quản lý dữ liệu.  - Có thể không tối ưu với một số mạng. |
| **Ứng dụng điển hình** | - CNN với output kích thước lớn.  - Thiết bị cần giảm lưu trữ output. | - Lớp fully-connected trong DNN.  - Thiết bị có bộ nhớ weight lớn. | - CNN với input kích thước lớn.  - Hệ thống tối ưu băng thông input. | - Kiến trúc phần cứng như TPU.  - Mạng đa dạng cần song song hóa. |
| **Tái sử dụng dữ liệu** | Cao cho output, thấp cho input/weight. | Cao cho weight, thấp cho input/output. | Cao cho input, thấp cho weight/output. | Cao cho dữ liệu cục bộ theo hàng. |
| **Hiệu quả năng lượng** | Tốt khi output lớn, kém khi input/weight lớn. | Tốt khi weight lớn, kém khi input/output lớn. | Tốt khi input lớn, kém khi weight/output lớn. | Tốt với song song hóa, phụ thuộc vào thiết kế. |

## 3.3. Quá trình tính toán tích chập

### 3.3.1 Tính toán 1D theo phương pháp Row Stationary

Phương pháp tính toán tích chập 1D theo Row Stationary là một kỹ thuật tiên tiến trong xử lý dữ liệu neural, mang lại hiệu quả và tối ưu trong việc thực hiện các phép tính giữa bộ lọc và dữ liệu đầu vào. Phương pháp này được thiết kế để tận dụng tối đa việc tái sử dụng dữ liệu và tối ưu hóa quá trình tính toán trong một đơn vị xử lý (Processing Element - PE).



Hình 3.14: Quá trình tính toán tích chập 1D theo phương pháp Row Stationary

Cấu trúc của phương pháp bao gồm hai phần chính: phần trên minh họa tổng quan về phép tích chập 1D giữa bộ lọc (filter) và Feature Map (Fmap) đầu vào, còn phần dưới chi tiết hoạt động bên trong PE với Register File và quá trình tính toán.

Quá trình tính toán diễn ra theo các bước chính: khởi tạo, nạp dữ liệu, thực hiện các phép nhân và cộng, cập nhật window đầu vào và filter, đồng thời sinh ra các giá trị tổng riêng phần (partial sums) tương ứng.

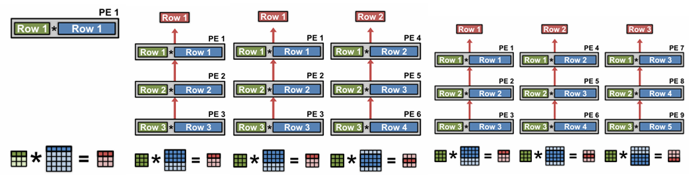
Đặc điểm nổi bật của phương pháp Row Stationary là khả năng giữ hàng filter cố định, xử lý hiệu quả dữ liệu đầu vào theo từng cửa sổ trượt. Điều này giúp giảm thiểu số lần truy cập bộ nhớ và tối ưu hóa quá trình xử lý.

Kỹ thuật này tạo ra các partial sums cho quá trình tích chập 1D một cách năng động và hiệu quả, thông qua việc duy trì các giá trị filter và window đầu vào, đồng thời liên tục cập nhật và xuất ra các giá trị tổng riêng phần.

Trong bối cảnh phát triển các hệ thống xử lý neural, phương pháp tính toán tích chập 1D theo Row Stationary được coi là một bước tiến quan trọng, mở ra những khả năng mới trong việc tối ưu hóa tính toán và sử dụng năng lượng.

### 3.3.2 Tính toán tích chập 2D trong mảng PE

Trong xử lý tín hiệu và học máy, việc tính toán tích chập hai chiều (2D convolution) là một thao tác quan trọng, đặc biệt là trong các ứng dụng như xử lý hình ảnh và mạng nơ-ron tích chập (CNN). Phương pháp Row Stationary được mở rộng để xử lý các phép tính tích chập 2D một cách hiệu quả, tận dụng tối đa tính song song của hệ thống.



Hình 3.15: Quá trình tính toán tích chập 2D theo phương pháp Row Stationary

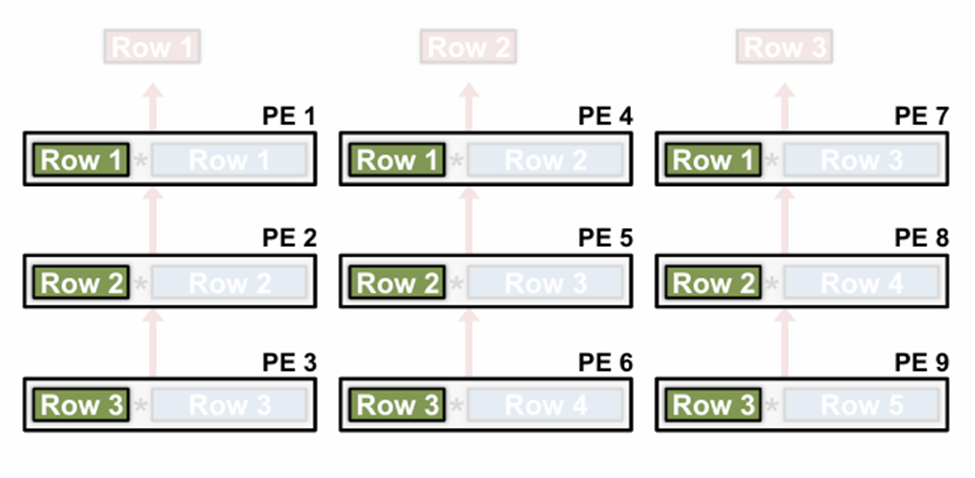
Việc mở rộng tính toán tích chập từ 1 chiều sang 2 chiều được thực hiện bằng cách tăng dần số lượng đơn vị xử lý (PE) và tổ chức chúng theo các cột và hàng. Quá trình này bắt đầu từ một PE đơn lẻ và dần mở rộng lên 3 PE, 6 PE, và cuối cùng là 9 PE, cho phép tính toán song song các phần tử của ma trận kết quả.

Mỗi giai đoạn mở rộng đều tuân theo nguyên lý Row Stationary, trong đó các hàng filter và dữ liệu đầu vào được giữ cố định, và việc tính toán được thực hiện một cách hiệu quả và song song. Điều này giúp tăng tốc độ xử lý, đặc biệt là trong các ứng dụng học sâu như mạng nơ-ron tích chập (CNN).

### 3.3.3 Tái sử dụng dữ liệu trong tính toán tích chập

#### 3.3.3.1 Tái sử dụng hàng bộ lọc theo chiều ngang giữa các PE

Trong kiến trúc tích chập sử dụng Row Stationary, việc tái sử dụng hàng bộ lọc theo chiều ngang là một kỹ thuật quan trọng nhằm tối ưu hóa hiệu quả tính toán. Mỗi hàng của bộ lọc (Filter) được sử dụng đồng thời trên nhiều đơn vị xử lý (PE) trong cùng một hàng.



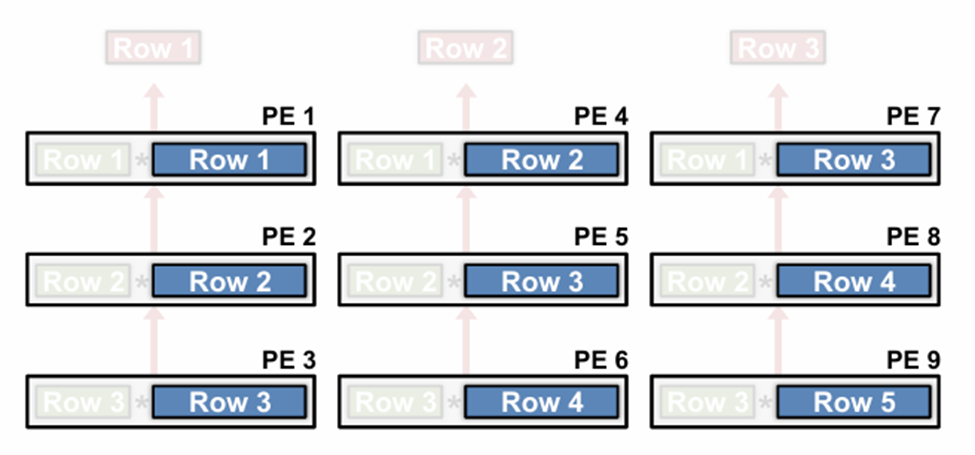
Hình 3.16: Phân phối hàng bộ lọc trên các PE theo chiều ngang

Ví dụ, hàng thứ nhất của bộ lọc sẽ được sử dụng song song trên các PE như PE1, PE4 và PE7. Tương tự, hàng thứ hai và thứ ba của bộ lọc cũng được phân phối và sử dụng trên các PE tương ứng. Phương pháp này giúp giảm đáng kể số lần truy cập bộ nhớ, bởi cùng một dữ liệu bộ lọc chỉ được đọc một lần và sau đó được phân phối cho nhiều đơn vị xử lý.

Việc tái sử dụng hàng bộ lọc theo chiều ngàng giúp 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.

#### 3.3.3.2 Tái sử dụng hàng đặc trưng đầu vào theo đường chéo

Trong kiến trúc tích chập, việc tái sử dụng hàng đặc trưng đầu vào (input feature map) được thực hiện theo một mô hình đường chéo độc đáo. Phương pháp này tận dụng tính địa phương của các phép tích chập, trong đó các phần tử đầu ra liền kề thường cần truy cập các hàng đầu vào chồng lấn.



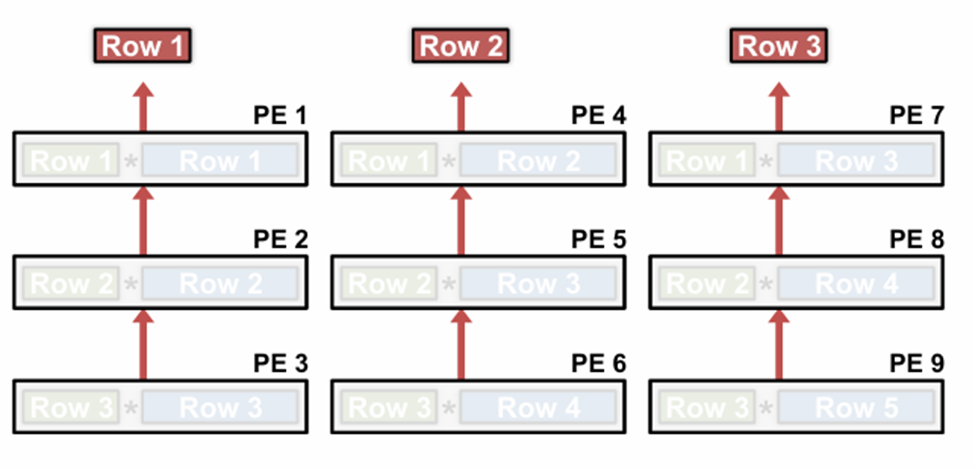
Hình 3.17: Phân phối hàng bộ lọc trên các PE theo đường chéo

Cụ thể, các hàng dữ liệu đầu vào được sắp xếp và phân bổ theo một mẫu đường chéo giữa các đơn vị xử lý (PE). Ví dụ, hàng 1 sẽ được gán cho PE1, PE4 và PE7; hàng 2 sẽ được gán cho PE2, PE5 và PE8; và cứ như vậy. Thiết kế này đảm bảo rằng dữ liệu đầu vào di chuyển và được sử dụng hiệu quả nhất trong toàn bộ mảng PE.

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.

#### 3.3.3.3. Tích lũy tổng một phần (partial sums) theo chiều dọc

Trong kiến trúc tích chập sử dụng Row Stationary, việc tích lũy các tổng một phần (partial sums) được thực hiện theo chiều dọc trên mảng các đơn vị xử lý (PE). Mỗi cột PE (ví dụ PE1-PE3, PE4-PE6, PE7-PE9) được giao nhiệm vụ tính toán và tích lũy một phần tử đầu ra cụ thể.



Hình 3.18: Phân phối hàng bộ lọc trên các PE theo chiều dọc

Các kết quả trung gian từ mỗi PE trong cột được tích lũy theo chiều dọc, sau đó mỗi cột PE cuối cùng sẽ tạo ra một phần tử trong hàng đầu ra. Phương pháp này tối ưu hóa luồng dữ liệu, cho phép các tổng một phần "chảy" một cách tự nhiên theo chiều dọc, thay vì phải di chuyển dữ liệu một cách không hiệu quả qua toàn bộ mảng.

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.

Khi kết hợp cả ba chiến lược - tái sử dụng theo chiều ngang để giảm lượng truy cập bộ lọc, tái sử dụng theo đường chéo nhằm tối ưu hóa việc di chuyển dữ liệu đầu vào, và tích lũy theo chiều dọc để giảm thiểu di chuyển dữ liệu trung gian - ta có một mô hình dataflow có hiệu quả cao trong tính toán tích chập. Phương pháp này không chỉ tối ưu hóa việc sử dụng tài nguyên phần cứng mà còn nâng cao đáng kể hiệu suất tính toán trong các ứng dụng học máy và xử lý tín hiệu.

# ÁNH XẠ TỚI PHẦN CỨNG

## 4.1. Điều phối dữ liệu

Điều phối dữ liệu tập trung vào quản lý di chuyển và lưu trữ dữ liệu để tối ưu hiệu suất khi ánh xạ phép tính của mạng nơ-ron sâu (DNN) lên phần cứng.

Các hệ thống tính toán hiện đại áp dụng những nguyên tắc cốt lõi nhằm tối ưu hiệu suất toàn diện. Việc tái sử dụng tài nguyên hiệu quả được thực hiện thông qua bố trí bộ nhớ gần đơn vị xử lý, giảm thiểu thời gian truy xuất dữ liệu. Nguyên tắc đồng bộ chính xác đảm bảo hệ thống chỉ xử lý những dữ liệu thực sự cần thiết, loại bỏ thời gian chờ đợi không cần thiết.

Kỹ thuật chồng chéo tải và sử dụng khéo léo kết hợp với bộ đệm kép giúp duy trì nguồn cung dữ liệu liên tục, trong khi nguyên tắc hiệu quả bộ nhớ tập trung vào việc loại bỏ trạng thái nhàn rỗi và giảm thiểu độ trễ. Việc tối ưu băng thông được thực hiện thông qua kiểm soát chặt chẽ số lượng yêu cầu truyền tải, cùng với cơ chế chia sẻ dữ liệu thông minh giữa các đơn vị xử lý giúp giảm đáng kể chi phí truy cập.

Những nguyên tắc này kết hợp hài hòa mang lại nhiều lợi ích thiết thực: nâng cao hiệu năng hệ thống, giảm thiểu độ trễ xử lý, tiết kiệm năng lượng tiêu thụ, tối ưu hóa việc sử dụng tài nguyên phần cứng và đảm bảo luồng dữ liệu được xử lý trơn tru, hiệu quả. Chúng tạo thành nền tảng quan trọng cho các thiết kế hệ thống tính toán hiệu suất cao ngày nay.

A group of rectangular colored buttons with white text

AI-generated content may be incorrect.

Hình 4.1: Nguyên tắc hướng dẫn sắp xếp dữ liệu

## 4.2. Các phương pháp điều phối dữ liệu

Có nhiều cách tiếp cận khác nhau, bao gồm:

1. Phương pháp ẩn (Implicit): Hệ thống tự động quản lý dữ liệu, dễ triển khai nhưng khó tối ưu.
2. Phương pháp tường minh (Explicit): Lập trình viên kiểm soát trực tiếp luồng dữ liệu, cho hiệu suất cao hơn nhưng phức tạp hơn.
3. Phương pháp kết hợp (Coupled): Tính toán và truy xuất dữ liệu diễn ra đồng bộ, đơn giản nhưng dễ gây nghẽn.
4. Phương pháp tách rời (Decoupled): Tách biệt hai quá trình này, tăng hiệu suất nhưng khó triển khai.
5. Phương pháp Explicit Decoupled kết hợp ưu điểm của cả hai, vừa cho phép kiểm soát chi tiết vừa đạt hiệu suất cao, là giải pháp tối ưu cho các hệ thống hiện đại.

## 4.3. EDDO Strategies

### 4.3.1. Buffets

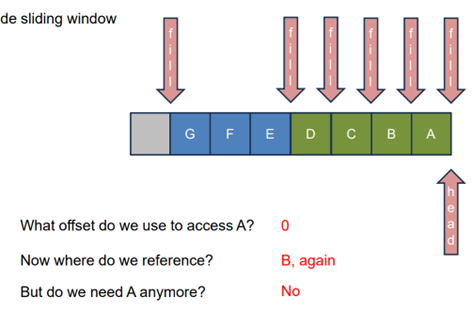
Buffets là mô hình lưu trữ mở rộng, cho phép truy cập dữ liệu ngẫu nhiên trong vùng nhớ đang hoạt động mà không cần tải lại toàn bộ dữ liệu.

1. FIFO: Buffets hỗ trợ truy cập ngẫu nhiên thay vì chỉ theo thứ tự vào-ra.
2. Scratchpad: Buffets có cơ chế kiểm soát đồng bộ (scoreboarding) để theo dõi trạng thái dữ liệu.
3. Cache: Buffets giảm yêu cầu lưu trữ thông tin thẻ (tag store) và sử dụng mô hình push thay vì pull để tối ưu băng thông.

### 4.3.2. Các Đặc Tính Hành Vi của Buffet (Buffet Behavioral Attributes)

Hành vi của buffet được xác định dựa trên trình tự địa chỉ liên quan đến việc nạp dữ liệu (fill), đọc dữ liệu (read), và cập nhật dữ liệu (update).

1. Nạp Dữ liệu (Fill): Bộ đệm lấy dữ liệu từ nguồn nạp đến khi đầy.
2. Đọc Dữ liệu (Read): Bộ đệm chuyển dữ liệu đến kênh đọc (nếu đã nạp). Địa chỉ đọc có thể yêu cầu xóa dữ liệu để tạo không gian (qua cổng điều khiển thu nhỏ).
3. Cập Nhật Dữ liệu (Update): Bộ đệm lấy giá trị mới từ kênh cập nhật.
4. Hỗ trợ nhiều bộ đệm logic: Một bộ đệm vật lý có thể chứa nhiều bộ đệm logic, tối ưu hóa lưu trữ và xử lý.



Hình 4.2: Ví dụ về điều khiển Buffer

## 4.4. Ánh xạ đầu ra cố định sang phần cứng (Mapping Output Stationary (OS) to Hardware)

Output Stationary (OS) là một phương pháp ánh xạ dữ liệu lên phần cứng, giữ các giá trị đầu ra cố định trong Processing Elements (PEs) để giảm thiểu việc đọc/ghi dữ liệu, từ đó tối ưu hóa hiệu suất và năng lượng.

### 4.4.1. Tích chập 1D (1-D Convolution)

Tích chập 1D thực hiện phép nhân giữa trọng số và dữ liệu đầu vào, sau đó cộng dồn kết quả để tạo đầu ra. Kỹ thuật này có hai ưu điểm chính: (1) Giảm đáng kể lượng tính toán nhờ cơ chế cửa sổ trượt, (2) Tối ưu hóa truy xuất bộ nhớ bằng cách tái sử dụng dữ liệu đầu qua các lần tính toán liên tiếp.

Các hệ thống gia tốc phần cứng thường khai thác triệt để hai đặc điểm này để đạt hiệu suất cao, đặc biệt trong các ứng dụng xử lý tín hiệu và mạng neural, nơi yêu cầu về tốc độ xử lý và tiết kiệm năng lượng luôn được đặt lên hàng đầu.

### 4.4.2. Luồng dữ liệu trong một PE ((Single PE Output Stationary Flow)

Kiến trúc PE (Processing Element) tĩnh đầu ra tối ưu hóa tính toán bằng cách giữ cố định các tổng tích lũy tại PE trong khi truyền dữ liệu và trọng số qua nó theo chu kỳ. Cách tiếp cận này loại bỏ thao tác tính toán địa chỉ phức tạp và áp dụng mô hình systolic đồng bộ, giúp tối ưu đường truyền dữ liệu.

Ưu điểm chính là giảm đáng kể thời gian truy cập bộ nhớ và tăng hiệu suất pipeline, đặc biệt phù hợp cho các phép tích chập trong mạng neural khi cần tính toán lặp trên cùng bộ trọng số. Kiến trúc này cân bằng tốt giữa hiệu năng và tiêu thụ năng lượng.

### 4.4.3. Mô hình tham chiếu (Reference Pattern)

Mô hình tham chiếu tối ưu hóa tính toán bằng cách tận dụng tính chất tái sử dụng của dữ liệu trong các phép tích chập. Cụ thể, các giá trị đầu ra và trọng số được sử dụng liên tục nhiều lần, trong khi dữ liệu đầu vào được quản lý hiệu quả thông qua cơ chế cửa sổ trượt.

Kỹ thuật sliding window cho phép hệ thống chỉ lưu trữ các phần dữ liệu đầu vào cần thiết tại từng thời điểm, giảm đáng kể yêu cầu về bộ nhớ. Đồng thời, việc tái sử dụng liên tục các trọng số và kết quả trung gian giúp tối ưu hóa hiệu suất tính toán tổng thể, đặc biệt hiệu quả khi triển khai trên các kiến trúc phần cứng chuyên dụng.

### 4.4.4. Truy cập dữ liệu cấp L1 (L1 Data Accesses)

Trong các phép tích chập, số lần truy cập bộ nhớ L1 được xác định như sau: Mỗi phép nhân-tích lũy (MAC) yêu cầu một lần đọc trọng số và một lần đọc dữ liệu đầu vào, với tổng số Q × S lần đọc cho mỗi loại (trọng số và đầu vào). Riêng dữ liệu đầu ra chỉ cần ghi một lần với tổng số Q lần ghi, không yêu cầu đọc đầu ra trong quá trình tính toán.

Các thông số này giúp đánh giá hiệu quả sử dụng bộ nhớ đệm L1 khi triển khai các thuật toán tích chập trên phần cứng.

### 4.4.5. Bộ nhớ đệm trung gian (Intermediate Buffering)

Bộ nhớ đệm trung gian đóng vai trò quan trọng trong việc tối ưu hiệu suất hệ thống bằng cách giảm thiểu số lần truy xuất dữ liệu từ bộ nhớ chính. Cơ chế này hoạt động thông qua việc bổ sung các tầng bộ nhớ đệm giữa bộ nhớ chính và các đơn vị xử lý (PE), tạo ra hệ thống phân cấp bộ nhớ hiệu quả.

Để tận dụng tối đa bộ đệm trung gian, kiến trúc hệ thống thường được thiết kế với các vòng lặp bổ sung trong cấu trúc lồng nhau (loop nest). Các vòng lặp này cho phép dữ liệu được tái sử dụng nhiều lần tại bộ đệm trước khi cần truy xuất lại từ bộ nhớ chính, qua đó cải thiện đáng kể hiệu suất tổng thể và giảm tiêu thụ năng lượng của hệ thống.

### 4.4.6. Ánh xạ không gian (Spatial Mapping)

Ánh xạ không gian là phương pháp phân phối phép tính trên nhiều đơn vị xử lý (PE) để tăng tốc độ xử lý thông qua tính toán song song. Khi triển khai, các PE cần giao tiếp hiệu quả với nhau để trao đổi dữ liệu trung gian và đồng bộ kết quả. Đồng thời, hệ thống phải tích hợp cơ chế tổng hợp không gian để kết hợp chính xác các kết quả cục bộ từ từng PE thành đầu ra cuối cùng. Kỹ thuật này đặc biệt quan trọng trong các ứng dụng xử lý song song khối lượng lớn như mạng neural, nơi yêu cầu cao về hiệu suất tính toán và khả năng mở rộng. Cách tiếp cận này không chỉ tăng tốc độ xử lý mà còn tối ưu hóa việc sử dụng tài nguyên phần cứng thông qua phân bổ khối lượng công việc hợp lý giữa các PE.

### 4.4.7. Trọng số cố định trong không gian (Spatial Weight Stationary - Skewed)

Kỹ thuật này tối ưu hóa hiệu suất tính toán bằng cách giữ cố định trọng số trong từng PE trong thời gian dài, nhằm tăng cường khả năng tái sử dụng dữ liệu. Hệ thống hỗ trợ truyền dữ liệu đa hướng (multicast) giữa các PE, cho phép giao tiếp hiệu quả và đồng bộ trong quá trình xử lý song song.

Quá trình ánh xạ đóng vai trò quan trọng khi triển khai mạng nơ-ron sâu (DNN) lên phần cứng gia tốc, bao gồm việc phân bổ tác vụ tính toán và lập lịch truy cập dữ liệu sao cho phù hợp với giới hạn tài nguyên của hệ thống. Cách tiếp cận này đảm bảo sự cân bằng giữa việc sử dụng bộ nhớ và khả năng xử lý, đồng thời tối ưu hóa hiệu suất tổng thể thông qua việc quản lý chặt chẽ luồng dữ liệu và phép tính trên kiến trúc phần cứng đa PE.

### 4.4.8. Ánh xạ ( Mapping)

Việc thiết kế hệ thống tính toán hiệu quả đòi hỏi quy trình tối ưu hóa qua 4 bước chính. Đầu tiên là lựa chọn luồng dữ liệu phù hợp bằng cách xác định thứ tự vòng lặp tính toán, giúp kiểm soát tốt cách thức truy xuất và di chuyển dữ liệu trong hệ thống bộ nhớ.

Tiếp theo, quá trình chia nhỏ dữ liệu được thực hiện để phân tách bài toán thành các phần nhỏ hơn, xác định rõ giới hạn không gian và thời gian cho từng vòng lặp, tạo điều kiện xử lý song song trên nhiều phần tử xử lý (PEs).

Ở mức độ chi tiết hơn, hệ thống cần được tinh chỉnh bằng cách lựa chọn các thông số hành vi cụ thể như việc bỏ qua một số cấp bộ nhớ không cần thiết để tối ưu hóa hiệu suất truy xuất.

Cuối cùng, bước gắn kết phép tính đảm bảo mỗi thao tác tính toán được ánh xạ chính xác tới đơn vị phần cứng phù hợp nhất, cân bằng tải và tận dụng tối đa tài nguyên hệ thống. Quy trình này đặc biệt quan trọng khi triển khai các thuật toán phức tạp như mạng neural trên kiến trúc đa lõi.

### 4.4.9. Mô hình tính toán CPU vs. DNN (CPU vs. DNN Compute Models)

Mô hình tính toán CPU hoạt động theo nguyên lý truyền thống, bắt đầu từ bộ xử lý trung tâm và trình biên dịch. Hệ thống thực thi các chương trình nhị phân để xử lý dữ liệu đầu vào, sau đó xuất kết quả đã xử lý cùng các thống kê về hành vi hệ thống. Cách tiếp cận này phù hợp cho các tác vụ đa dụng với luồng điều khiển phức tạp.

Trong khi đó, mô hình tính toán DNN tập trung vào xử lý mạng nơ-ron sâu, bắt đầu từ việc xác định đặc tả bài toán và hình dạng dữ liệu đầu vào. Quá trình bao gồm xác định luồng dữ liệu tối ưu cùng các ràng buộc phần cứng cụ thể, sau đó thực hiện ánh xạ các đầu vào và trọng số để tạo ra kết quả cuối cùng. Mô hình này đặc biệt hiệu quả cho các tác vụ xử lý song song khối lượng lớn như nhận dạng hình ảnh hay xử lý ngôn ngữ tự nhiên.

Bảng 4.1: So sánh giữa mô hình CPU và CNN

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Mô hình CPU** | **Mô hình DNN** |
| Bắt đầu | Bộ xử lý và trình biên dịch | Đặc tả bài toán và dữ liệu |
| Cách xử lý | Chạy chương trình nhị phân | Xây dựng mô hình dựa trên dữ liệu và ràng buộc  Tổ chức luồng dữ liệu và ánh xạ các phép toán  Kết quả dự đoán từ mô hình DNN |
| Định hướng dữ liệu | Xử lý dữ liệu theo từng lệnh cụ thể | Tổ chức luồng dữ liệu và ánh xạ các phép toán |
| Đầu ra | Dữ liệu đã xử lý và thống kê hành vi | Kết quả dự đoán từ mô hình DNN |

### 4.4.10. Công cụ hỗ trợ ánh xạ (Mapping Tools)

Timeloop là công cụ chuyên dụng cho việc đánh giá và khám phá không gian thiết kế kiến trúc các bộ gia tốc DNN. Nó cung cấp khả năng mô phỏng đa dạng các mô hình mạng nơ-ron sâu khác nhau và có thể tích hợp với nhiều kiến trúc phần cứng, giúp các nhà thiết kế tìm ra giải pháp tối ưu cho hệ thống của mình.

Accelergy đóng vai trò bổ trợ quan trọng bằng cách cung cấp ước lượng chính xác mức tiêu thụ năng lượng dựa trên mô tả kiến trúc và phương án triển khai cụ thể. Công cụ này cho phép đánh giá hiệu quả năng lượng ngay từ giai đoạn thiết kế, giúp tối ưu hóa kiến trúc để đạt được cân bằng giữa hiệu suất tính toán và tiêu thụ năng lượng. Cả hai công cụ kết hợp tạo thành bộ giải pháp mạnh mẽ cho quy trình thiết kế bộ gia tốc DNN hiện đại.

### 4.4.11. Các đặc tả cần thiết cho ánh xạ (Specifications for Mapping)

Timeloop sử dụng ba thành phần đặc tả chính để phân tích và tối ưu hóa kiến trúc gia tốc DNN. Đầu tiên là đặc tả khối lượng công việc, cung cấp thông tin chi tiết về không gian phép toán và không gian dữ liệu cần xử lý. Thành phần này giúp xác định phạm vi bài toán cần giải quyết.

Đặc tả kiến trúc đóng vai trò quan trọng trong việc xác định các thông số vận hành hệ thống. Nó bao gồm: các ánh xạ hợp lệ giữa phép tính và phần cứng, thứ tự vòng lặp xử lý dữ liệu, giới hạn thực thi của các vòng lặp. Đặc tả này còn tích hợp các cơ chế tối ưu như khả năng tái sử dụng dữ liệu theo thời gian và không gian, cùng các ràng buộc về thứ tự thực thi chỉ số.

Thành phần cuối cùng là đặc tả triển khai, tập trung vào các thông số hiệu năng cụ thể. Nó xác định độ trễ hệ thống và mức tiêu thụ năng lượng cho từng ánh xạ, đồng thời cung cấp thông tin chi tiết về: băng thông bộ đệm, kiến trúc cổng truy cập bộ đệm, mức độ song song của các PE, và độ trễ mạng liên kết. Những thông số này giúp đánh giá chính xác hiệu suất thực tế của kiến trúc đề xuất.

# CO-DESIGN OF DNN MODELS AND HARDWARE: SPARSITY

## 5.1. Nguồn gốc của độ thưa

Độ thưa trong mạng neural xuất hiện từ hai nguồn chính với các đặc điểm khác nhau. Đối với activation, độ thưa chủ yếu sinh ra từ ba cơ chế: tính chất của hàm kích hoạt ReLU khi chuyển đổi tất cả giá trị âm thành 0, các mẫu tương quan tồn tại trong chính dữ liệu đầu vào, cùng với cấu trúc biểu diễn đặc thù của dữ liệu như trong trường hợp biểu diễn đồ thị.

Mặt khác, độ thưa của trọng số được tạo ra một cách chủ động thông qua các kỹ thuật tối ưu mô hình. Quá trình này bao gồm việc sắp xếp lại ma trận trọng số theo những mẫu hình nhất định, cơ chế tái sử dụng trọng số thông minh và đặc biệt là kỹ thuật pruning (cắt tỉa mạng) nhằm loại bỏ các kết nối không quan trọng. Cả hai dạng độ thưa này đều đóng vai trò quan trọng trong việc tối ưu hóa hiệu suất tính toán và tiết kiệm tài nguyên hệ thống.

## 5.2. Khai thác tính thưa thớt

Tính chất thưa thớt trong dữ liệu và các phép tính toán mang lại những lợi ích đáng kể cho việc tối ưu hóa hệ thống. Bằng cách loại bỏ các giá trị bằng không không cần thiết, chúng ta có thể tiết kiệm đáng kể không gian lưu trữ và giảm năng lượng tiêu thụ. Đồng thời, việc tránh thực hiện các phép tính với giá trị không cũng giúp tăng tốc độ xử lý tổng thể.

Những kỹ thuật này kết hợp với nhau tạo thành cơ chế tối ưu mạnh mẽ, cho phép giảm thiểu cả ba yếu tố quan trọng: không gian lưu trữ, thời gian tính toán và năng lượng tiêu thụ, từ đó nâng cao hiệu quả tổng thể của hệ thống xử lý dữ liệu thưa thớt.

A diagram of a graph

AI-generated content may be incorrect.

Hình 5.1: Sự thưa thớt giữa có và không có hàm kích hoạt

## 5.3. Các kỹ thuật cụ thể

### 5.3.1. Nén dữ liệu thưa

Việc nén dữ liệu thưa thớt đem lại những lợi ích đáng kể trong việc giảm thiểu chi phí hệ thống. Quan trọng nhất là giảm đáng kể chi phí di chuyển dữ liệu qua các bus bộ nhớ và tiết kiệm không gian lưu trữ. Kỹ thuật này đặc biệt hiệu quả với các ma trận thưa thường xuất hiện trong các mô hình học sâu.

Để đảm bảo hiệu quả thực tế, quá trình nén cần đáp ứng ba yêu cầu cốt lõi: khả năng giải mã duy nhất để đảm bảo tính toàn vẹn dữ liệu, thuật toán đủ nhẹ để không tạo thêm gánh nặng tính toán, và quan trọng nhất là không được làm mất mát dữ liệu để bảo toàn độ chính xác của các phép tính. Các giải pháp nén thường sử dụng phương pháp mã hóa đơn nhất (uniquely decodable) kết hợp với thuật toán tối giản, cân bằng giữa hiệu quả nén và chi phí giải mã.

### 5.3.2. Cnvlutin

Cnvlutin là kiến trúc phần cứng chuyên xử lý tích chập, phát triển từ DaDianNao với chi phí chip chỉ 4.49%. Hệ thống tự động bỏ qua phép tính với activation bằng 0, giúp tăng tốc độ xử lý 1.37-1.52 lần so với phương pháp thông thường. Đặc biệt hiệu quả với mạng neural sử dụng ReLU, Cnvlutin mang lại giải pháp tối ưu cả về hiệu suất và năng lượng cho các ứng dụng AI biên.

A diagram of a graph

AI-generated content may be incorrect.

Hình 5.2: Cnvlutin

### 5.3.3. Pruning activation

Pruning activation là kỹ thuật loại bỏ các giá trị kích hoạt (activation values) nhỏ không đáng kể trong mạng neural, giúp giảm độ phức tạp tính toán. Tuy nhiên, việc này cần được thực hiện cẩn trọng để không ảnh hưởng đáng kể đến độ chính xác của mô hình.

Kết quả thực nghiệm cho thấy:

1. Trên tập dữ liệu ImageNet, kỹ thuật này giúp tăng tốc độ xử lý ~11%
2. Trên tập dữ liệu MNIST, hiệu quả còn ấn tượng hơn với khả năng giảm một nửa chi phí tính toán

Phương pháp này đặc biệt hữu ích cho các ứng dụng cần tốc độ xử lý nhanh với độ chính xác có thể chấp nhận được, giúp cân bằng giữa hiệu suất và chất lượng mô hình.

A graph of different types of data

AI-generated content may be incorrect.

Hình 5.3: Biểu đồ của hàm kích hoạt cắt tỉa

### 5.3.4. Khai thác ReLU

Hàm kích hoạt ReLU (Rectified Linear Unit) tạo ra tính thưa thớt tự nhiên bằng cách chuyển đổi tất cả giá trị âm thành 0. Đặc tính này cho phép tối ưu hóa hiệu suất hệ thống thông qua hai cơ chế chính:

1. Giảm phép tính thừa: Khi có thể dự đoán trước kết quả âm (sau ReLU sẽ thành 0), hệ thống có thể bỏ qua các phép tính không cần thiết, tiết kiệm đáng kể tài nguyên tính toán.
2. Yêu cầu phần cứng hỗ trợ: Cần bổ sung các mạch kiểm tra để xác định thời điểm dừng tính toán sớm khi dự đoán được kết quả âm. Dù làm tăng độ phức tạp thiết kế nhưng mang lại hiệu quả tổng thể cao.

Cách tiếp cận này đặc biệt hiệu quả với các mô hình deep learning sử dụng ReLU phổ biến, giúp cân bằng giữa tăng tốc độ xử lý và chi phí triển khai phần cứng.

A diagram of a function

AI-generated content may be incorrect.

Hình 5.4: Khai thác ReLU

### 5.3.5. Tương quan không gian/thời gian

Tận dụng tương quan không gian-thời gian giúp giảm phép tính thừa trong xử lý dữ liệu. Các activation lân cận thường tương đồng, cho phép áp dụng chung kết quả tính toán. Trong xử lý video, khung hình liên tiếp có độ tương quan cao, chỉ cần xử lý phần thay đổi. Phương pháp này tối ưu hiệu quả tính toán mà vẫn đảm bảo độ chính xác.

**A comparison of a infrared image

AI-generated content may be incorrect.**

Hình 5.5: Khai thác tương quan không gian đầu vào

## 5.4. Graph Neural Networks (GNN)

GNN xử lý dữ liệu đồ thị thông qua ma trận liền kề thưa thớt, nơi mỗi nút được biểu diễn bằng vector đặc trưng và toàn bộ đồ thị bằng ma trận đặc trưng. Quá trình tính toán gồm hai bước chính:

1. Aggregate: Tổng hợp thông tin từ các nút lân cận để tạo đặc trưng trung bình
2. Combine: Kết hợp đặc trưng tổng hợp trọng số để tạo đầu vào cho lớp tiếp theo

Phương pháp này tối ưu hóa việc xử lý đồ thị thưa bằng cách chỉ tính toán trên các kết nối tồn tại, giảm đáng kể chi phí tính toán so với ma trận đầy đủ.

A diagram of a diagram of a diagram

AI-generated content may be incorrect.

Hình 5.6: Mạng nơ-ron đồ thị

## 5.5. Khai thác trọng số dư thừa

Quá trình khai thác trọng số dư thừa được thực hiện qua hai bước chính. Đầu tiên, hệ thống tiến hành tiền xử lý để sắp xếp và tổ chức lại các trọng số theo một cấu trúc tối ưu. Sau đó, áp dụng kỹ thuật thực hiện phép cộng trước phép nhân (multiply-accumulate) giúp giảm đáng kể số lượng phép nhân cần thực hiện và số lần đọc trọng số từ bộ nhớ.

Cách tiếp cận này mang lại hiệu quả kép: vừa giảm số phép tính toán cần thực hiện, vừa tối ưu hóa việc truy xuất bộ nhớ, từ đó cải thiện đáng kể hiệu suất tổng thể của hệ thống mà không làm ảnh hưởng đến độ chính xác của mô hình. Đặc biệt phù hợp với các hệ thống cần xử lý lượng lớn các phép tính ma trận trong thời gian thực.

A black text with red text

AI-generated content may be incorrect.

Hình 5.7: Khai thác trọng số dư thừa

### 5.5.1. Pruning – Make Weights Sparse

Loại bỏ trọng số dựa trên độ lớn của trọng số. Cách loại bỏ này bao gồm:

1. Chọn kiến trúc mạng hợp lý: Xác định số lượng lớp, số lượng nơ-ron trong mỗi lớp, v.v.
2. Huấn luyện mạng nơ-ron đến khi đạt kết quả hợp lý: Đây là bước học ban đầu để có một mô hình hoạt động tốt.
3. Tính đạo hàm bậc hai (second derivative) của từng trọng số: Đạo hàm bậc hai giúp xác định mức độ ảnh hưởng của trọng số đến hàm lỗi.
4. Tính "saliency" (tầm quan trọng) của từng trọng số: Saliency thể hiện tác động của trọng số đối với lỗi huấn luyện. Trọng số có giá trị saliency thấp là ít quan trọng.
5. Sắp xếp trọng số theo saliency và xóa đi những trọng số ít quan trọng: Những trọng số có ít ảnh hưởng sẽ bị loại bỏ để giảm số lượng tham số.
6. Quay lại bước 2 và lặp lại quá trình: Mô hình sẽ tiếp tục được huấn luyện để điều chỉnh lại các trọng số còn lại.

|  |  |
| --- | --- |
| A graph showing the fine tuning  AI-generated content may be incorrect.  Hình 5.8: Biểu đồ khi áp dụng phương pháp cắt tỉa | A diagram of a training scheme  AI-generated content may be incorrect.  Hình 5.9: Làm cho trọng lượng thưa thớt  bằng phương pháp cắt tỉa |

A diagram of a weight removal program

AI-generated content may be incorrect.

Hình 5.10: Quy trình áp dụng phương pháp cắt tỉa

### 5.5.2. Loại bỏ trọng số: Chấm điểm

Các kỹ thuật cắt tỉa mô hình thường áp dụng phương pháp chấm điểm để đánh giá và lựa chọn trọng số cần loại bỏ. Cách tiếp cận này gán điểm số cụ thể cho từng trọng số hoặc nhóm trọng số dựa trên mức độ ảnh hưởng của chúng đến các tiêu chí quan trọng như độ chính xác của mô hình. Trong đó, phương pháp phổ biến nhất là cắt tỉa dựa trên độ lớn của trọng số, thường ưu tiên loại bỏ các giá trị có độ lớn nhỏ vì ít ảnh hưởng đến đầu ra.

Một biến thể khác là cắt tỉa dựa trên đặc trưng, tập trung vào việc phân tích tầm quan trọng của từng đặc trưng trong mô hình. Đặc biệt, các phương pháp hiện đại còn tích hợp đồng thời việc đánh giá tác động của trọng số đến cả hiệu quả năng lượng lẫn thông lượng xử lý, giúp tối ưu hóa toàn diện hiệu suất mô hình trên các hệ thống phần cứng cụ thể. Cách tiếp cận đa tiêu chí này cho phép cân bằng giữa độ chính xác và hiệu suất vận hành của mô hình sau khi cắt tỉa.

### 5.5.3. Gom nhóm

Tăng mức độ thô (cấu trúc dễ hơn cho phần cứng), ít báo hiệu hơn cho vị trí của các số không (nén tốt hơn).

A diagram of a diagram

AI-generated content may be incorrect.

Hình 5.11: Ví dụ về loại bỏ gom nhóm

### 5.5.4. Các loại cắt tỉa khác

1. Coarse-grained pruning: Tạo cấu trúc thưa dễ khai thác trên phần cứng (ví dụ: Scalpel, tốc độ tăng 1.92x).
2. Pattern-based pruninh: Cắt tỉa theo mẫu thay vì hàng.
3. Ranking và Fine-tuning: Xếp hạng trọng số/group để loại bỏ, sau đó tinh chỉnh để khôi phục độ chính xác.

## 5.6. Tinh chỉnh và Lập lịch trình

Cập nhật các giá trị của các trọng số còn lại để khôi phục độ chính xác, xác định số lượng trọng số cần tỉa bớt trong mỗi lần lặp.

## 5.7. Hiệu quả và tradeoff

Pruning giảm trọng số (9x) và MAC (3x) trên AlexNet, nhưng hiệu quả khác nhau giữa các tầng (CONV vs FC).

Các mô hình DNN hiện đại hiệu quả hơn có thể bị giảm độ chính xác nhanh hơn khi pruning.

Sử dụng direct metrics (năng lượng, độ trễ) thay vì proxy mang lại tradeoff tốt hơn.

A graph of different layers

AI-generated content may be incorrect.

Hình 5.12: Cắt tỉa và các loại lớp

## 5.8. Phần cứng hỗ trợ

Các hệ thống phần cứng hiện đại tích hợp nhiều cơ chế thông minh để nâng cao hiệu quả hoạt động. Kỹ thuật gating cho phép tự động vô hiệu hóa các phép tính toán hoặc truy xuất dữ liệu khi phát hiện giá trị 0, giúp giảm thiểu đáng kể năng lượng tiêu thụ. Cùng với đó, cơ chế skipping tối ưu hóa thời gian xử lý bằng cách bỏ qua các chu kỳ tính toán không cần thiết, vừa tiết kiệm năng lượng vừa tăng tốc độ tổng thể.

Một yếu tố quan trọng khác là việc lựa chọn định dạng biểu diễn tensor phù hợp. Bằng cách tối ưu hóa cách thức lưu trữ và truyền tải dữ liệu tensor, hệ thống có thể giảm thiểu dung lượng bộ nhớ sử dụng đồng thời tăng hiệu suất truy xuất. Các kỹ thuật này kết hợp với nhau tạo thành giải pháp tổng thể giúp nâng cao hiệu quả năng lượng và tốc độ xử lý của hệ thống AI chuyên sâu.

## 5.9. Kết luận

Độ thưa đạt 30-70%, cần phần cứng chuyên dụng để khai thác (khác với thư viện phần mềm yêu cầu >99%).

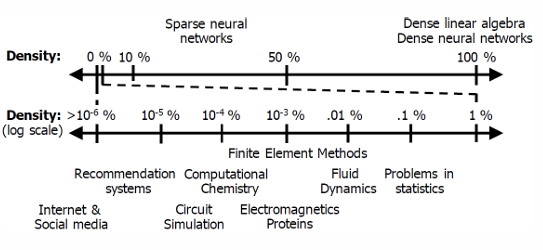
Pruning thô và fine-tuning tăng cường hiệu quả, nhưng cần cân nhắc tradeoff giữa tốc độ, kích thước và độ chính xác.

# SPARSE ARCHITECTURES

## 6.1 Tổng quan về Sparse Tensor

Sparse Tensors là những tensor (cấu trúc dữ liệu đa chiều, tương tự như mảng đa chiều) đặc biệt ở chỗ phần lớn các phần tử có giá trị bằng 0, chỉ một số ít phần tử mang giá trị khác 0. Đây là một khái niệm quan trọng trong xử lý dữ liệu hiện đại, mang lại nhiều lợi ích đáng kể.

Việc sử dụng Sparse Architectures mang lại ba ưu điểm chính. Thứ nhất, về mặt lưu trữ, sparse tensor có thể giảm dung lượng bộ nhớ tới hàng nghìn lần đối với các tensor cực kỳ thưa bằng cách chỉ lưu trữ các phần tử khác 0. Thứ hai, về tính toán, các thuật toán có thể bỏ qua các phép tính liên quan đến giá trị 0, từ đó làm tăng đáng kể tốc độ xử lý. Thứ ba, sparse tensor cho phép xử lý các bài toán với dữ liệu có kích thước rất lớn mà các tensor truyền thống không thể giải quyết do giới hạn bộ nhớ.



Hình 6.1: Thang đo mật độ của Sparse Tensor trên các ứng dụng khác nhau

Các thang đo mật độ (density) của sparse tensor rất đa dạng, từ cực kỳ thưa (10^-6 %) cho tới tương đối dày đặc (1%). Mức độ sparse khác nhau phù hợp với các ứng dụng khác nhau. Ví dụ, các mạng xã hội và internet thuộc phân khúc cực kỳ thưa, trong khi các bài toán về protein, động lực học chất lỏng và thống kê nằm ở phân khúc ít thưa hơn.

Đối với các kiến trúc máy học, sparse tensor mang lại những lợi ích cụ thể. Về mặt tài nguyên, chúng cho phép lưu trữ và xử lý dữ liệu lớn với bộ nhớ ít hơn nhiều. Các phép toán trên sparse tensor có thể nhanh hơn đáng kể vì có thể bỏ qua các phép tính với số 0. Nhiều kiến trúc mạng nơ-ron hiện đại còn sử dụng tính thưa để giảm số lượng tham số mà không làm giảm đáng kể hiệu suất. Quan trọng hơn, sparse tensor tương thích với nhiều miền ứng dụng thực tế, từ xử lý ảnh đến mô phỏng vật lý.

Hiểu được bản chất và ưu điểm của sparse tensor là bước đầu tiên quan trọng trong việc áp dụng các kiến trúc tiên tiến này vào các bài toán thực tế, mở ra những khả năng mới trong xử lý dữ liệu và học máy.

### 6.1.1 Các loại Tensor

Tensor được phân loại theo bậc (rank), mỗi bậc biểu diễn một chiều không gian khác nhau và mang những đặc điểm riêng biệt. Hiểu được các bậc tensor là cơ sở để nắm bắt cấu trúc dữ liệu đa chiều này.

Rank-0 - Scalar (Vô hướng) là đơn vị cơ bản nhất, được biểu diễn bằng một khối đơn lẻ. Đây là một giá trị đơn không có chiều, chẳng hạn như số 5 hay số 3.14. Scalar tượng trưng cho điểm khởi đầu của không gian tensor, nơi mà mọi thứ còn đơn giản nhất.

Rank-1 - Vector (Vectơ) nâng cấp lên một chiều, được thể hiện như một hàng các khối. Một vector là một mảng một chiều chứa các giá trị, ví dụ như dãy số [1, 2, 3, 4, 5, 6]. Tại đây, mỗi phần tử đã có vị trí và ý nghĩa riêng trong không gian một chiều.

Rank-2 - Matrix (Ma trận) mở rộng sang hai chiều, được biểu diễn như một lưới 2D gồm các khối. Ma trận là một mảng hai chiều các giá trị, như [[1, 2, 3], [4, 5, 6], [7, 8, 9]]. Tại bậc này, ta đã có khái niệm về hàng và cột, tạo nên một không gian phức tạp hơn.

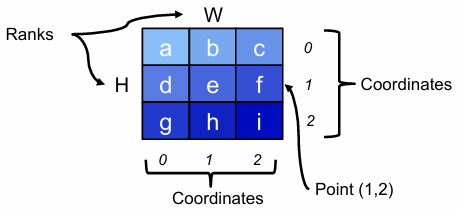
Rank-3 - Cube (Khối lập phương) là bước nhảy vọt tiếp theo, biểu diễn một mảng ba chiều. Ví dụ như [[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[10, 11, 12], [13, 14, 15], [16, 17, 18]], [[19, 20, 21], [22, 23, 24], [25, 26, 27]]]. Tại đây, ta đã có một không gian ba chiều hoàn chỉnh, cho phép biểu diễn các dữ liệu phức tạp hơn.

Bậc (rank) của tensor cho biết số lượng chỉ số cần thiết để xác định một phần tử cụ thể trong tensor đó. Mỗi bậc gia tăng lại mở ra một không gian mới, phức tạp hơn và giàu thông tin hơn. Từ scalar đơn giản cho đến khối lập phương đa chiều, tensor cung cấp một công cụ mạnh mẽ để biểu diễn và xử lý dữ liệu trong nhiều lĩnh vực từ toán học đến khoa học máy tính.

Sự phát triển từ rank-0 đến rank-3 không chỉ là sự gia tăng về số chiều, mà còn là sự gia tăng về khả năng biểu diễn và xử lý thông tin phức tạp. Mỗi bậc tensor đều có ý nghĩa và ứng dụng riêng, tạo nên một hệ thống linh hoạt và mạnh mẽ để làm việc với dữ liệu đa chiều.

### 6.1.2 Thuật ngữ dữ liệu Tensor

Trong thế giới của tensor, mỗi phần tử được xác định một cách chính xác thông qua hệ thống tọa độ phức tạp và có ý nghĩa. Hai khái niệm cốt lõi chi phối việc hiểu và thao tác với tensor là: "Các phần tử của mỗi bậc (chiều) được xác định bởi tọa độ của chúng" và "Mỗi phần tử của tensor được xác định bởi bộ tọa độ từ mỗi bậc của nó".

****

Hình 6.2: Biểu diễn tọa độ và Ranks trong ma trận

Hãy lấy một ma trận mẫu để minh họa: một tensor bậc 2 chứa các phần tử từ a đến i, được tổ chức theo hai chiều chính - chiều H (Height/Chiều cao) và chiều W (Width/Chiều rộng). Mỗi chiều có các tọa độ riêng, cụ thể là 0, 1, 2. Điều này có nghĩa là bạn có thể xác định chính xác vị trí của bất kỳ phần tử nào bằng cách sử dụng một cặp tọa độ (hàng, cột).

Tọa độ là khái niệm then chốt trong việc định vị các phần tử. Ví dụ, điểm (1,2) chính là vị trí của phần tử "f" - nằm ở hàng 1, cột 2 của ma trận. Mỗi chiều (rank) sở hữu không gian tọa độ riêng của mình, cho phép xác định vị trí chính xác của từng phần tử.

Tính độc đáo của tensor nằm ở khả năng biểu diễn dữ liệu đa chiều một cách chính xác và linh hoạt. Bằng cách sử dụng hệ tọa độ phức tạp này, chúng ta có thể mô tả các cấu trúc dữ liệu phức tạp từ các miền ứng dụng khác nhau - từ xử lý ảnh đến học máy.

Hệ thống tọa độ này không chỉ là một cách thức kỹ thuật để định vị dữ liệu, mà còn là nền tảng cho việc thực hiện các phép toán phức tạp trên tensor. Mỗi tọa độ mang trong mình một ý nghĩa không gian, cho phép chúng ta thao tác, biến đổi và phân tích dữ liệu một cách chính xác và sâu sắc.

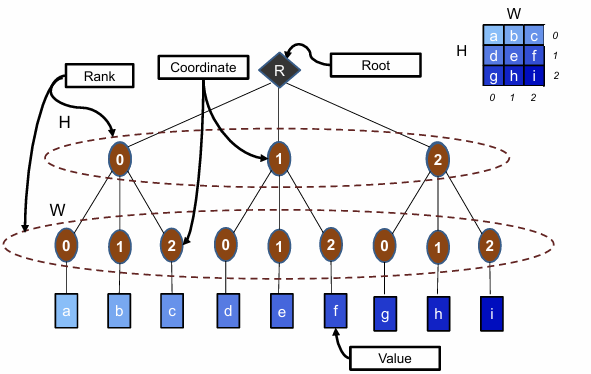
Hiểu được cách thức hoạt động của tọa độ trong tensor là chìa khóa để nắm bắt sức mạnh của các cấu trúc dữ liệu đa chiều này. Đó không chỉ là việc lưu trữ thông tin, mà còn là cách biểu diễn và xử lý thông tin một cách tinh vi và hiệu quả.

### 6.1.3 Biểu diễn Tensor dựa trên cây và sợi (Tree-based and Fiber)

Biểu diễn tensor thông qua cấu trúc cây và khái niệm sợi (fiber) mở ra những góc nhìn mới về cách tổ chức và truy cập dữ liệu đa chiều. Đây là những phương pháp tiên tiến giúp tối ưu hóa lưu trữ và xử lý thông tin trong các tensor phức tạp.

1. Biểu diễn Tensor dựa trên Cây

Cấu trúc cây cung cấp một phương thức trừu tượng và linh hoạt để biểu diễn tensor. Mỗi đường dẫn từ gốc đến một lá đại diện cho một phần tử cụ thể trong tensor. Ví dụ, để tìm phần tử "f", ta sẽ đi từ gốc R, di chuyển đến chiều H với tọa độ 1, sau đó sang chiều W với tọa độ 2, cuối cùng đến giá trị "f".

****

Hình 6.3: Cấu trúc cây Tensor

Ý nghĩa sâu xa của biểu diễn này nằm ở việc thể hiện mối quan hệ phân cấp giữa các chiều và tọa độ. Mỗi nhánh của cây không chỉ là một con đường định vị, mà còn là một cách thức tổ chức dữ liệu một cách logic và có cấu trúc. Điều này tạo nên nền tảng cho các phương pháp lưu trữ và truy cập hiệu quả.

2. Tree-based Tensor Abstraction (with Fibers)

Khái niệm "fiber" (sợi) mở rộng và làm sâu sắc thêm cách nhìn về tensor. Một fiber được định nghĩa là tập hợp các phần tử tensor có chung tọa độ trên một số chiều nhất định. Trong biểu diễn cây, các sợi được thể hiện bằng các đường tròn đứt khối ở các tầng khác nhau.

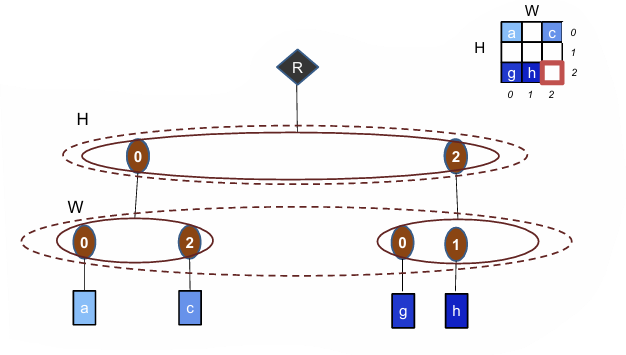
****

Hình 6.4: Biểu diễn Fiber trong cấu trúc Tensor

Mỗi sợi tương ứng với một đường đi một phần trong cây, cho phép tham chiếu hiệu quả đến các nhóm phần tử. Ví dụ, một fiber có thể là tất cả các phần tử của một hàng cụ thể trong ma trận, hoặc tất cả các giá trị tại một tọa độ cụ thể trong một tensor nhiều chiều.

Tính quan trọng của fiber trong xử lý sparse tensor (tensor thưa) là vô cùng lớn. Nó cho phép:

1. Biểu diễn hiệu quả các tensor thưa mà không cần lưu trữ các giá trị 0
2. Truy vấn nhanh chóng phần tử hoặc nhóm phần tử thông qua việc duyệt cây
3. Thực hiện các thao tác trên sparse tensor ở cấp độ sợi, giảm đáng kể độ phức tạp tính toán
4. Nén dữ liệu bằng cách chỉ lưu trữ các nhánh có giá trị khác không

****

Hình 6.5: Biểu diễn Tensor thưa - Loại bỏ các Fiber bằng không

Khi tensor trở nên thưa thớt, việc loại bỏ các phần tử bằng 0 trở nên cực kỳ hiệu quả. Nếu một fiber (như một hàng H=1) chỉ chứa các giá trị 0, toàn bộ fiber có thể được loại bỏ. Điều này không chỉ tiết kiệm bộ nhớ mà còn tối ưu hóa tính toán, cho phép các thuật toán bỏ qua hoàn toàn các fiber bằng 0 thay vì kiểm tra từng phần tử.

## 6.2. Phương pháp biểu diễn tọa độ ngầm

Implicit Coordinates (tọa độ ngầm) là một kỹ thuật tinh vi trong xử lý dữ liệu thưa, cho phép biểu diễn và nén dữ liệu mà không cần lưu trữ rõ ràng các tọa độ của các phần tử khác 0. Thay vì sử dụng phương pháp truyền thống yêu cầu lưu trữ chi tiết vị trí, phương pháp này tập trung vào việc nén dữ liệu một cách thông minh và hiệu quả.

Bản chất của phương pháp này nằm ở việc loại bỏ thông tin dư thừa. Thay vì lưu trữ toàn bộ không gian tensor, chỉ những phần tử mang thông tin (các giá trị khác 0) mới được quan tâm. Điều này không chỉ tiết kiệm bộ nhớ mà còn tối ưu hóa các thao tác tính toán, đặc biệt là với các tập dữ liệu có mức độ thưa cao.

### 6.2.1. Run-Length Encoding (RLE)

Run-Length Encoding (RLE) là một kỹ thuật nén dữ liệu đơn giản, thường được sử dụng cho dữ liệu thưa (sparse data). Ý tưởng chính là thay vì gửi các giá trị 0 (các phần tử không quan trọng trong dữ liệu thưa), RLE sẽ gửi **số lượng** (run length) các số 0 liên tiếp, sau đó gửi giá trị khác 0 tiếp theo.

RLE mang lại ưu điểm chính là tiết kiệm bộ nhớ khi dữ liệu có nhiều chuỗi số 0 dài và không cần lưu trữ tọa độ rõ ràng. Tuy nhiên, phương pháp này cũng tồn tại một số hạn chế như kém hiệu quả khi dữ liệu không quá thưa và khó thực hiện truy cập ngẫu nhiên.

Ứng dụng của RLE rất rộng rãi, phù hợp với các hệ thống có dữ liệu thưa như xử lý ảnh, mạng xã hội và các ứng dụng xử lý thông tin có tính chất lặp lại cao. Sự đơn giản và hiệu quả của RLE làm cho nó trở thành một công cụ quan trọng trong việc nén dữ liệu và tối ưu hóa lưu trữ thông tin.

### 6.2.2. Significance Map Coding (Bitmask Coding)

Significance Map Coding, còn được gọi là bitmask coding, là một phương pháp tiên tiến trong biểu diễn dữ liệu thưa. Kỹ thuật này sử dụng một bitmask (dãy nhị phân) để chỉ ra vị trí của các giá trị khác 0, sau đó truyền tải các giá trị đó một cách hiệu quả.

Phương pháp hoạt động dựa trên việc tạo ra một bitmask, trong đó mỗi bit biểu thị việc một vị trí có giá trị khác 0 hay không. Cụ thể, bit 1 cho biết vị trí đó có giá trị khác 0, còn bit 0 biểu thị giá trị bằng 0. Sau khi xác định các vị trí này, các giá trị khác 0 tương ứng sẽ được truyền tải.

So với Run-Length Encoding, Significance Map Coding mang lại những lợi thế riêng. Phương pháp này cho phép xác định nhanh chóng vị trí các giá trị khác 0 thông qua bitmask và đặc biệt hiệu quả khi các chuỗi số 0 không quá dài. Tuy nhiên, đối với các vector rất dài, bitmask có thể chiếm dung lượng đáng kể do yêu cầu 1 bit cho mỗi phần tử.

Ưu điểm chính của kỹ thuật này là tính linh hoạt và khả năng truy cập nhanh chóng các phần tử khác 0. Nó đặc biệt phù hợp trong các ứng dụng yêu cầu xử lý nhanh dữ liệu thưa, như xử lý ảnh, học máy và các hệ thống xử lý tín hiệu số.

### 6.2.3. Huffman Encoding

Huffman Encoding là một phương pháp nén dữ liệu tiên tiến, được xây dựng dựa trên nguyên lý phân phối xác suất xuất hiện của các giá trị. Kỹ thuật này tối ưu hóa việc biểu diễn dữ liệu bằng cách gán mã (codeword) ngắn hơn cho các giá trị xuất hiện thường xuyên, và mã dài hơn cho những giá trị xuất hiện ít.

Cốt lõi của Huffman Encoding nằm ở việc xây dựng một bảng xác suất cho các "message" (giá trị hoặc run lengths), sau đó sử dụng cây Huffman để gán mã sao cho những giá trị có xác suất cao được mã hóa bằng số bit ít hơn. Điều này giúp giảm thiểu tổng số bit cần thiết để biểu diễn dữ liệu.

Ưu điểm của phương pháp này là khả năng nén dữ liệu một cách tối ưu, đặc biệt là với các tập dữ liệu có phân phối xác suất không đều. Tuy nhiên, Huffman Encoding cũng có nhược điểm là độ phức tạp trong việc triển khai, đòi hỏi các thuật toán xây dựng cây mã và giải mã tương đối phức tạp.

Trong lĩnh vực xử lý dữ liệu thưa, Huffman Encoding mang lại những lợi ích đáng kể như giảm không gian lưu trữ, tăng tốc độ truyền tải dữ liệu và hỗ trợ tốt cho việc triển khai trên các thiết bị có tài nguyên hạn chế. Đây là một công cụ quan trọng trong việc nén dữ liệu và tối ưu hóa lưu trữ thông tin.

### 6.2.4. So sánh các phương pháp

Mỗi phương pháp nén dữ liệu thưa - Run-Length Encoding (RLE), Significance Map Coding và Huffman Encoding - đều sở hữu những đặc điểm và ưu nhược điểm riêng, phù hợp với các tình huống và loại dữ liệu khác nhau.

Run-Length Encoding là phương pháp đơn giản nhất, đặc biệt hiệu quả khi dữ liệu có nhiều chuỗi số 0 liên tiếp dài. Tuy nhiên, hiệu quả của nó giảm đáng kể khi dữ liệu không quá thưa. Significance Map Coding mang lại giải pháp linh hoạt hơn, cho phép truy cập nhanh các vị trí có giá trị khác 0 thông qua bitmask. Huffman Encoding được coi là phương pháp tối ưu nhất về mặt lý thuyết, nhưng lại phức tạp hơn trong việc triển khai thực tế.

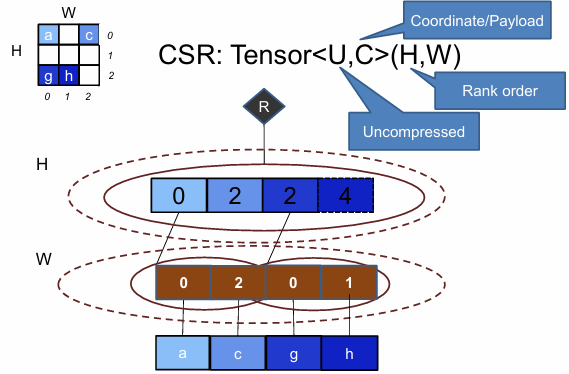
Việc lựa chọn phương pháp phù hợp phụ thuộc vào nhiều yếu tố. Độ thưa của dữ liệu là tiêu chí quan trọng hàng đầu - các dữ liệu càng thưa, RLE hoặc Huffman càng hiệu quả. Các yêu cầu tính toán quyết định hướng lựa chọn: nếu cần truy cập nhanh, Significance Map là lựa chọn tối ưu; nếu ưu tiên nén tối đa, Huffman sẽ là giải pháp lý tưởng.

Trong lĩnh vực máy học và mạng nơ-ron sâu, Significance Map Coding (Bitmask Coding) được ưa chuộng nhất. Phương pháp này không chỉ hiệu quả về mặt phần cứng mà còn dễ dàng triển khai trên các thiết bị chuyên dụng, đặc biệt là các hệ thống FPGA và ASIC. Khả năng xử lý nhanh các phép toán bit và hỗ trợ tốt cho các mô hình mạng nơ-ron thưa là những ưu điểm vượt trội của phương pháp này.

### 6.2.5 Ứng dụng các phương pháp nén trong thực tế

1. Notation for CSR (Compressed Sparse Row)

Compressed Sparse Row (CSR) là một phương pháp tiên tiến trong biểu diễn ma trận thưa, được thiết kế để tối ưu hóa việc lưu trữ và xử lý dữ liệu với số lượng phần tử 0 lớn. Thay vì lưu trữ toàn bộ không gian ma trận, CSR chỉ tập trung vào các phần tử mang giá trị khác 0, kèm theo thông tin cần thiết để truy xuất chúng.



Hình 6.6: Biểu diễn CSR

Cấu trúc CSR bao gồm ba mảng chính: Segment Array (hay còn gọi là row pointers), Coordinate Array và Value Array. Segment Array giúp xác định vị trí bắt đầu và kết thúc của mỗi hàng, Coordinate Array lưu trữ các chỉ số cột, còn Value Array chứa các giá trị khác 0 tương ứng.

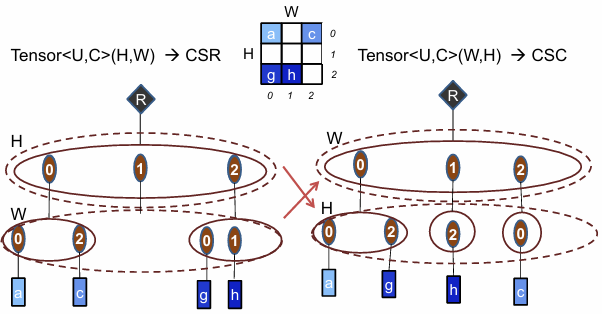
Ưu điểm của CSR nằm ở khả năng tiết kiệm bộ nhớ đáng kể và hiệu quả trong các phép toán ma trận thưa, như nhân ma trận-véc-tơ. Tuy nhiên, phương pháp này cũng có hạn chế nhất định, đặc biệt là trong việc truy cập ngẫu nhiên và xử lý các phép toán theo cột.

CSR được ứng dụng rộng rãi trong nhiều lĩnh vực, từ mạng nơ-ron sâu - nơi được sử dụng để biểu diễn các ma trận trọng số sau khi áp dụng kỹ thuật pruning, cho đến xử lý đồ thị - với việc biểu diễn ma trận kề. Trong các thư viện tính toán khoa học như SciPy, CSR là một định dạng chuẩn cho việc lưu trữ và xử lý ma trận thưa.

2. Representation of Order of Ranks

Compressed Sparse Row (CSR) và Compressed Sparse Column (CSC) là hai phương pháp biểu diễn ma trận thưa với cách tiếp cận khác nhau, mỗi phương pháp phù hợp với những loại phép toán và yêu cầu xử lý riêng biệt.

CSR được thiết kế để lưu trữ và xử lý ma trận theo hàng (row-major), trong khi CSC tập trung vào biểu diễn theo cột (column-major). Điều quan trọng là cả hai phương pháp đều làm việc với cùng một tập dữ liệu, nhưng khác nhau ở thứ tự rank và cách duyệt dữ liệu.



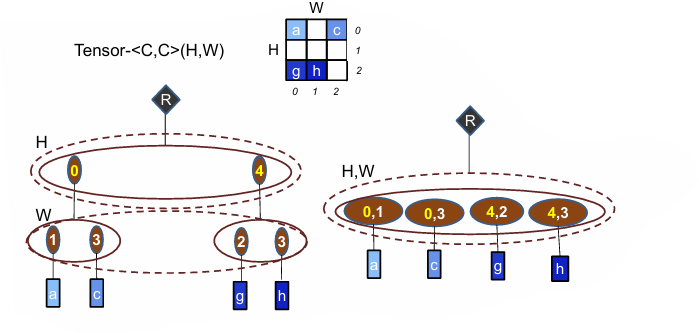
Hình 6.7: Biểu diễn CSR và CSC

Quá trình chuyển đổi giữa CSR và CSC được thực hiện thông qua một phép hoán đổi rank đơn giản. Việc này không làm thay đổi nội dung dữ liệu, mà chỉ điều chỉnh cách tổ chức và truy cập thông tin trong bộ nhớ. Mỗi phương pháp sẽ có ưu thế riêng: CSR hiệu quả cho các phép toán theo hàng, còn CSC phù hợp với các thao tác truy cập cột nhanh chóng.

Sự lựa chọn giữa CSR và CSC phụ thuộc vào loại phép toán cụ thể muốn thực hiện. Hiểu rõ đặc điểm của từng phương pháp giúp các nhà phát triển tối ưu hóa hiệu suất xử lý dữ liệu thưa trong các ứng dụng máy học và xử lý số.

3. Merging Ranks

Merging Ranks là một kỹ thuật tiên tiến trong xử lý tensor thưa, nhằm tối ưu hóa cấu trúc dữ liệu và nâng cao hiệu suất truy cập. Phương pháp này tập trung vào việc gộp các rank (chiều) của tensor theo những cách khác nhau để giảm thiểu độ phức tạp và tăng cường khả năng quản lý bộ nhớ.



Hình 6.8: Biểu diễn Merging Ranks

Quá trình hợp nhất ranks diễn ra thông qua nhiều biến thể khác nhau, mỗi biến thể phục vụ những mục đích cụ thể. Từ việc chuyển đổi cấu trúc tensor thành các biểu diễn khác nhau như Tensor-C², Tensor-H², Tensor-U² hay Tensor-R², mục tiêu luôn là tối ưu hóa không gian lưu trữ và tăng tốc độ xử lý.

Ý nghĩa sâu xa của kỹ thuật này nằm ở khả năng linh hoạt trong việc tái cấu trúc dữ liệu. Bằng cách gộp các rank, các nhà nghiên cứu có thể giảm thiểu yêu cầu về bộ nhớ, cải thiện lokalitas dữ liệu và tạo ra những biểu diễn hiệu quả hơn cho các tensor phức tạp.

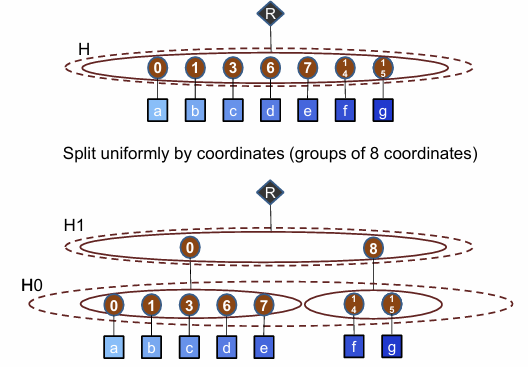
Merging Ranks không chỉ là một kỹ thuật thuần túy lý thuyết, mà còn là công cụ quan trọng trong việc tối ưu hóa các hệ thống xử lý dữ liệu lớn, từ học máy đến xử lý ảnh và khoa học dữ liệu.

4. Splitting Fibers

* Coordinate Space

Coordinate Space là một kỹ thuật tiên tiến trong xử lý sparse tensor, nhằm tối ưu hóa cấu trúc dữ liệu và nâng cao hiệu suất truy cập. Phương pháp này tập trung vào việc chia không gian tọa độ của tensor thành các nhóm con nhỏ hơn, tạo ra một cấu trúc phân cấp linh hoạt và hiệu quả.

Mục tiêu chính của Coordinate Space là giảm thiểu độ phức tạp tính toán và tăng cường lokalitas dữ liệu. Bằng cách chia nhỏ không gian tọa độ, kỹ thuật này cho phép xử lý song song và tuần tự các phần của tensor một cách hiệu quả hơn. Điều này đặc biệt quan trọng trong các ứng dụng xử lý dữ liệu lớn và thưa, từ máy học đến xử lý ảnh và tín hiệu.



Hình 6.9: Phân chia Coordinate Space

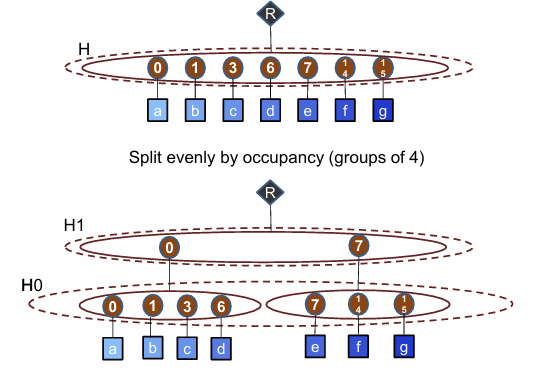
Lợi ích của Coordinate Space không chỉ dừng lại ở việc giảm độ phức tạp. Phương pháp này còn hỗ trợ việc truy cập nhanh chóng các phần tử trong tensor, tối ưu hóa việc sử dụng bộ nhớ và tăng cường hiệu suất xử lý. Bằng cách tạo ra các nhóm con logic, Coordinate Space mở ra những khả năng mới trong việc quản lý và xử lý dữ liệu đa chiều.

Trong bối cảnh các hệ thống tính toán ngày càng phức tạp, Coordinate Space trở thành một công cụ quan trọng giúp các nhà nghiên cứu và kỹ sư vượt qua những thách thức về hiệu suất và bộ nhớ khi làm việc với các tập dữ liệu lớn và thưa.

* Position Space

Position Space là một kỹ thuật tiên tiến trong xử lý sparse tensor, tập trung vào việc chia không gian dựa trên vị trí của các phần tử dữ liệu. Khác với Coordinate Space, phương pháp này chuyển trọng tâm từ tọa độ sang vị trí thực tế của các giá trị trong tensor.

Mục tiêu chính của Position Space là tối ưu hóa cách thức lưu trữ và truy cập dữ liệu thưa. Bằng cách chia không gian theo vị trí, kỹ thuật này giúp tăng hiệu quả truy cập dữ liệu, giảm độ phức tạp tính toán và cải thiện lokalitas bộ nhớ. Điều này đặc biệt quan trọng trong các ứng dụng như xử lý ảnh, tín hiệu và các hệ thống máy học có dữ liệu thưa.



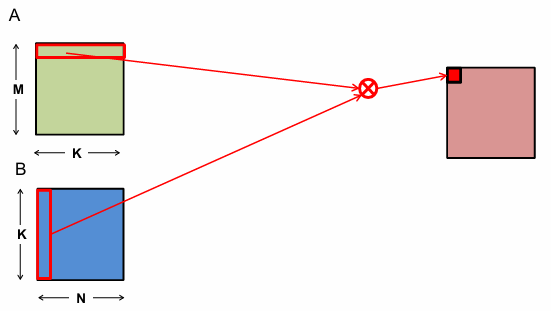
Hình 6.10: Phân chia Position Space

Ưu điểm nổi bật của Position Space bao gồm khả năng nâng cao hiệu suất truy cập, tối ưu hóa việc xử lý song song và cung cấp một cách tiếp cận linh hoạt để quản lý dữ liệu đa chiều. Phương pháp này không chỉ là một kỹ thuật lý thuyết, mà còn là một công cụ thực tế giúp các nhà nghiên cứu và kỹ sư vượt qua những thách thức trong xử lý dữ liệu lớn và phức tạp.

Trong bối cảnh công nghệ ngày càng phát triển, Position Space trở thành một phương pháp quan trọng để giải quyết các vấn đề về hiệu suất và bộ nhớ trong các hệ thống tính toán hiện đại.

## 6.3. Phép nhân ma trận thưa

Phép nhân ma trận thưa là kỹ thuật tối ưu hóa quan trọng trong lĩnh vực máy học, được thiết kế để xử lý ma trận số lượng lớn phần tử bằng không. Kỹ thuật này nhằm giảm thiểu chi phí tính toán và lưu trữ bằng cách chỉ tập trung vào các phần tử mang giá trị khác 0.



Hình 6.11: Minh họa điều kiện nhân ma trận - Chiều chia sẻ K

Để thực hiện phép nhân ma trận, điều kiện tiên quyết là số cột của ma trận đầu tiên phải bằng số hàng của ma trận thứ hai. Chiều này được gọi là "chiều chia sẻ", là nơi diễn ra các phép tính nhân và cộng để tạo ra ma trận kết quả.

Phương pháp Einsum (Einstein Summation) cung cấp một cách tiếp cận tổng quát và linh hoạt cho phép nhân ma trận. Nguyên tắc cơ bản của Einsum bao gồm:

1. Chỉ số chia sẻ (intersection): Các chỉ số xuất hiện cả hai ma trận, thể hiện phép giao.
2. Chỉ số co lại (reduction): Các chỉ số xuất hiện ở cả hai ma trận nhưng không có trong kết quả, thường được thực hiện bằng phép cộng.
3. Chỉ số không co lại: Các chỉ số xác định các điểm của kết quả đầu ra.

Quá trình tính toán diễn ra bằng cách duyệt qua không gian các chỉ số hợp lệ. Tại mỗi điểm, giá trị được tính toán dựa trên các chỉ số được chỉ định cho từng ma trận, sau đó được gán cho kết quả.

Phép nhân ma trận thưa không chỉ là một kỹ thuật toán học thuần túy, mà còn là một công cụ quan trọng trong việc xử lý dữ liệu lớn, đặc biệt là trong các ứng dụng máy học và xử lý ảnh. Bằng cách chỉ tập trung vào các phần tử mang thông tin, kỹ thuật này giúp tiết kiệm tài nguyên tính toán và mở ra khả năng mới trong xử lý dữ liệu phức tạp.

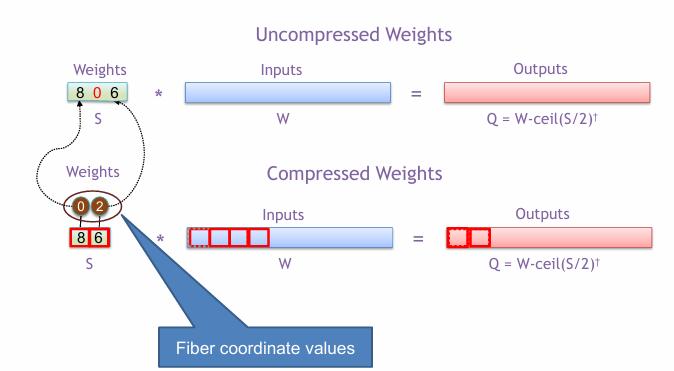
Sự phát triển của các thuật toán và cấu trúc dữ liệu như CSR (Compressed Sparse Row) và các phương pháp nén khác đã làm cho phép nhân ma trận thưa trở nên càng hiệu quả và quan trọng trong các hệ thống tính toán hiện đại.

### 6.3.1. Kiến trúc với trọng số thưa (Sparse Weights)

Sparse Weights là một khái niệm tiên tiến trong lĩnh vực máy học, mô tả các mô hình có phần lớn trọng số bằng 0. Mục tiêu chính của kỹ thuật này là tối ưu hóa hiệu suất tính toán, giảm thiểu yêu cầu bộ nhớ và ngăn chặn hiện tượng overfitting trong các mạng nơ-ron sâu.

1. Output Stationary

Chiến lược Output Stationary là một phương pháp tiên phong trong xử lý trọng số thưa. Phương pháp này tập trung vào việc giữ các giá trị đầu ra cố định trong bộ nhớ trong quá trình tính toán, cho phép xử lý hiệu quả các ma trận có nhiều giá trị bằng không.



Hình 6.12: So sánh trọng số nén và không nén

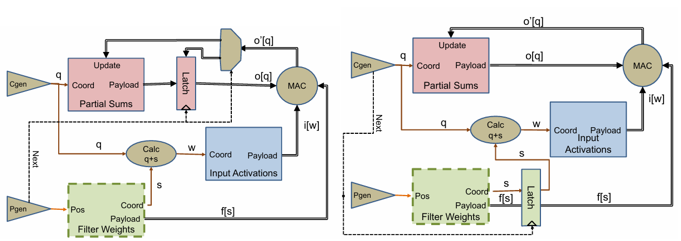
So sánh giữa trọng số không nén và trọng số nén cho thấy những khác biệt đáng chú ý. Trong biểu diễn không nén, toàn bộ ma trận trọng số được sử dụng, bao gồm cả các giá trị bằng không. Ngược lại, trọng số nén chỉ lưu trữ và xử lý các giá trị khác 0, kèm theo thông tin về vị trí của chúng.

Kỹ thuật này không chỉ đơn thuần là một phương pháp nén dữ liệu, mà còn là một chiến lược tính toán thông minh. Bằng cách chỉ tập trung vào các trọng số mang thông tin, Output Stationary giúp giảm đáng kể chi phí tính toán và lưu trữ, đồng thời duy trì hiệu suất của mô hình.

Trong thực tế, việc áp dụng Sparse Weights và Output Stationary đã mở ra những khả năng mới trong việc phát triển các mô hình máy học hiệu quả, đặc biệt là trên các hệ thống có tài nguyên hạn chế. Đây là một bước tiến quan trọng trong việc làm cho các thuật toán học sâu trở nên más tiết kiệm và linh hoạt hơn.

2. Cấu trúc của Output Stationary và Weight Stationary

Mạng nơ-ron thưa yêu cầu các kiến trúc phần cứng chuyên biệt để xử lý hiệu quả. Hai cấu trúc chính được minh họa ở đây bao gồm các thành phần: Filter Weights, Input Activations, khối Calc q+s, Update Partial Sums, Latch, và khối MAC (Multiply-Accumulate).



Hình 6.13: So sánh kiến trúc Weight Stationary   
và Output Stationary - Sparse Weight

Kiến trúc này được tối ưu hóa để:

* Chỉ xử lý các trọng số khác 0
* Tính toán theo mô hình "Output-Stationary" và “Weight-Stationary”
* Tận dụng phép duyệt "concordant" (đồng bộ)
* Tối ưu hóa tính toán

Đây là một thiết kế phần cứng chuyên dụng nhằm tăng tốc độ và hiệu quả năng lượng khi thực hiện các phép tính trên mạng nơ-ron thưa, đặc biệt phù hợp cho các ứng dụng yêu cầu hiệu suất cao hoặc các thiết bị với nguồn tài nguyên hạn chế.

3. Định hướng xử lý trọng số thưa

Việc không tập trung vào Input Stationary xuất phát từ những nhận định sâu sắc về bản chất của dữ liệu trong mạng nơ-ron sâu. Tính thưa chủ yếu nằm ở trọng số (weights), chứ không phải ở đầu vào (inputs). Trong hầu hết các trường hợp, các kích hoạt (activations) sau các lớp như ReLU thường không còn thưa nữa.

Các chiến lược Weight Stationary và Output Stationary được ưu tiên vì chúng tận dụng trực tiếp tính thưa của trọng số. Bằng cách chỉ xử lý các trọng số mang thông tin, các phương pháp này giúp giảm đáng kể chi phí tính toán và lưu trữ. Đầu ra vẫn được tối ưu hóa để tận dụng tính thưa của trọng số, qua đó giảm băng thông bộ nhớ và nâng cao hiệu suất tính toán.

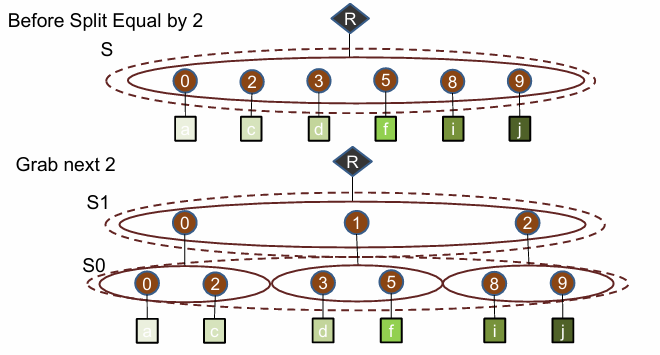
Input Stationary thường chỉ hiệu quả trong những trường hợp đầu vào đã thưa, điển hình như trong xử lý ngôn ngữ tự nhiên với các biểu diễn thưa như bag-of-words. Tuy nhiên, trong các mạng nơ-ron sâu hiện đại, đầu vào thường khá dày đặc, đặc biệt là sau các lớp như ReLU, điều này làm giảm đi tính hiệu quả của Input Stationary.

Các kiến trúc phần cứng hiện đại như NVIDIA GPU hay Google TPU được tối ưu hóa cho Weight Stationary hoặc Output Stationary khi xử lý trọng số thưa. Cách tổ chức này phù hợp hơn với cách dữ liệu được tổ chức và tính toán trong các mạng nơ-ron hiện đại, do đó mang lại hiệu quả cao hơn.

Sự lựa chọn này không chỉ là một quyết định kỹ thuật đơn thuần, mà còn phản ánh sự hiểu biết sâu sắc về cách thức dữ liệu được xử lý trong các hệ thống học sâu. Bằng cách tập trung vào những phương pháp phù hợp nhất với đặc điểm của dữ liệu, các nhà nghiên cứu có thể đạt được hiệu suất tính toán tối ưu.

#### 6.3.1.1. Fiber Splitting trong không gian vị trí

Fiber Splitting trong không gian vị trí là một kỹ thuật tiên tiến nhằm tối ưu hóa việc xử lý dữ liệu thưa bằng cách chia một "fiber" (tập hợp các vị trí và giá trị) thành các phần bằng nhau dựa trên các chỉ số.



Hình 6.14: Minh họa Fiber Splitting

Quá trình này bắt đầu từ một fiber duy nhất S với các vị trí và giá trị ban đầu. Sau khi chia, fiber được tách thành hai phần S0 và S1, mỗi phần chứa một tập con của các vị trí và giá trị. Khối R (Reorder) đóng vai trò quan trọng trong việc sắp xếp lại hoặc tái tổ chức dữ liệu sau khi phân chia.

Mục tiêu chính của kỹ thuật này là giảm độ phức tạp tính toán và tận dụng tối đa tính thưa ma trận. Bằng cách chia nhỏ dữ liệu, các nhà nghiên cứu có thể:

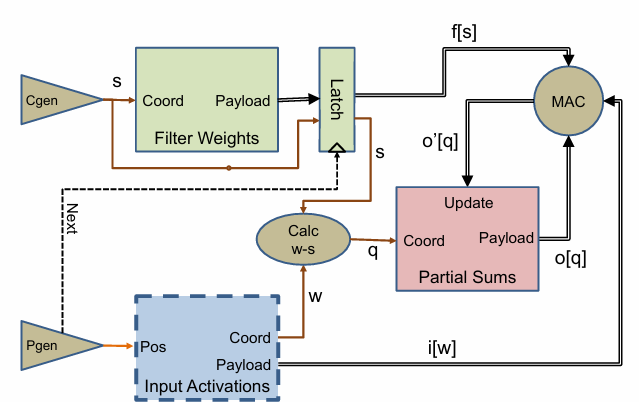
1. Hỗ trợ xử lý song song hiệu quả hơn
2. Tối ưu hóa việc truy cập và xử lý dữ liệu
3. Giảm thiểu chi phí tính toán

Fiber Splitting không chỉ là một kỹ thuật thuần túy lý thuyết, mà còn là một công cụ thực tế giúp cải thiện hiệu suất xử lý trong các hệ thống xử lý dữ liệu thưa, từ máy học đến xử lý ảnh và khoa học dữ liệu.

### 6.3.2 Kiến trúc với đầu vào thưa (Sparse Inputs)

1. Weight Stationary

Kiến trúc Weight Stationary với đầu vào thưa là một phương pháp tiên tiến trong xử lý mạng nơ-ron, tập trung vào việc giữ các trọng số cố định trong khi các đầu vào được truyền qua để tính toán.



Hình 6.15: Kiến trúc Weight Stationary - Sparse Inputs

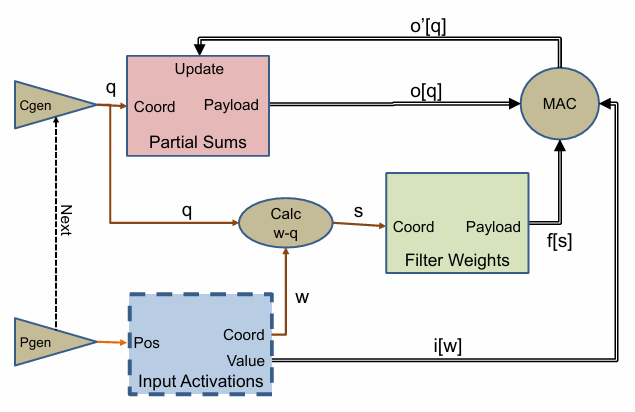
So sánh với Weight Stationary - Sparse Weights, phiên bản này có điểm khác biệt cơ bản ở chỗ tính thưa không còn ở trọng số, mà là ở các giá trị đầu vào. Khối Latch vẫn được sử dụng để giữ các giá trị cố định, nhưng mục tiêu là truyền các đầu vào thưa qua để thực hiện các phép tính.

Điểm chung giữa hai phiên bản là việc sử dụng mô hình "weight stationary" để giữ trọng số cố định, nhưng khác nhau ở bản chất dữ liệu thưa (trọng số hay đầu vào) và vai trò của khối Latch trong quá trình xử lý.

Kiến trúc này cho phép xử lý hiệu quả mạng nơ-ron có đầu vào thưa, tận dụng tối đa các phép tính có ý nghĩa và giảm thiểu chi phí tính toán. Đây là một bước tiến quan trọng trong việc tối ưu hóa các hệ thống học sâu, đặc biệt là trên các thiết bị có tài nguyên hạn chế.

2. Output Stationary

Output Stationary với đầu vào thưa là một phương pháp tiên tiến trong xử lý mạng nơ-ron, tập trung vào việc giữ các giá trị đầu ra cố định trong quá trình tính toán.



Hình 6.16: Kiến trúc Output Stationary – Sparse Inputs

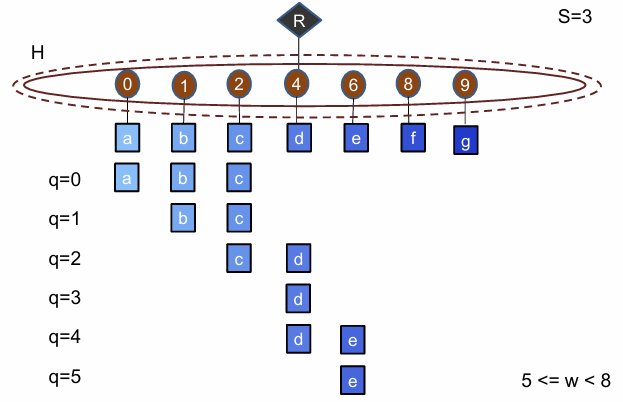
So sánh với Output Stationary - Sparse Weights, phiên bản này chuyển trọng tâm từ tính thưa của trọng số sang tính thưa của đầu vào. Khối đầu ra o[q] vẫn được giữ cố định, nhưng quy trình xử lý được điều chỉnh để phù hợp với đặc điểm của đầu vào thưa.

Điểm chung giữa hai phiên bản là việc sử dụng mô hình "Output Stationary" để giữ đầu ra cố định, nhưng khác nhau ở bản chất dữ liệu thưa (trọng số hay đầu vào). Phương pháp này cho phép xử lý hiệu quả các mạng nơ-ron có đặc điểm đầu vào thưa, tối ưu hóa việc tính toán và lưu trữ.

Ý nghĩa của kiến trúc này nằm ở khả năng thích ứng với các dạng dữ liệu phức tạp, giảm thiểu chi phí tính toán và nâng cao hiệu suất xử lý trong các hệ thống học sâu. Đây là một bước phát triển quan trọng trong việc thiết kế các mô hình máy học ngày càng hiệu quả và tiết kiệm tài nguyên.

#### 6.3.2.1 Sparse Sliding Window

Sparse Sliding Window là một kỹ thuật tiên tiến được sử dụng trong xử lý tín hiệu, thị giác máy tính và học máy để phân tích dữ liệu thưa một cách hiệu quả. Phương pháp này áp dụng nguyên lý cửa sổ trượt (sliding window) nhưng được tối ưu hóa đặc biệt cho các tập dữ liệu có nhiều phần tử bằng không.



Hình 6.17: Minh họa Sparse Sliding Window

Sparse Sliding Window là một Kỹ thuật hoạt động bằng cách di chuyển một cửa sổ có kích thước cố định qua chuỗi dữ liệu thưa, nhưng với một điểm khác biệt quan trọng: chỉ thu nhận các phần tử khác không. Ví dụ, với một cửa sổ kích thước 3, phương pháp này sẽ di chuyển qua dữ liệu, bỏ qua các vị trí trống và chỉ tập trung vào các giá trị mang thông tin.

Khối "R" đóng vai trò then chốt trong quá trình này, thực hiện việc trích xuất và sắp xếp dữ liệu một cách thông minh. Điều này giúp tối ưu hóa việc xử lý, giảm thiểu chi phí tính toán và tăng hiệu suất xử lý dữ liệu thưa.

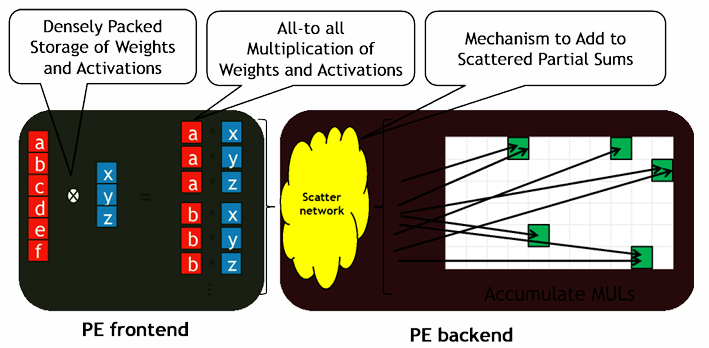
Ý nghĩa sâu xa của Sparse Sliding Window nằm ở khả năng xử lý hiệu quả các tập dữ liệu có cấu trúc phức tạp, nơi mà các phương pháp truyền thống gặp nhiều hạn chế. Kỹ thuật này mở ra những khả năng mới trong việc phân tích và xử lý thông tin, đặc biệt là trong các lĩnh vực yêu cầu độ chính xác và hiệu quả cao.

## 6.4 Mạng CNN thưa (Sparse CNN)

### 6.4.1 Kiến trúc của SCNN

Sparse CNN (Convolutional Neural Network thưa) là một hệ thống tiên tiến trong việc tối ưu hóa xử lý các lớp tích chập của mạng nơ-ron, được thiết kế đặc biệt để xử lý hiệu quả các trường hợp có cả trọng số và kích hoạt đầu vào đều thưa.

Kiến trúc này sử dụng chiến lược "Input Stationary Dataflow" - một phương pháp cách mạng nhằm tối ưu hóa tính toán, giảm thiểu băng thông bộ nhớ và nâng cao hiệu quả năng lượng trong các mạng học sâu.



Hình 6.18: Kiến trúc SCNN

Cấu trúc SCNN bao gồm hai thành phần chính: PE frontend và PE backend. PE frontend được thiết kế để lưu trữ các trọng số và kích hoạt đầu vào ở dạng được nén chặt, cho phép xử lý hiệu quả các giá trị khác không. Quá trình nhân toàn diện giữa trọng số và kích hoạt được thực hiện, tạo ra tất cả các tổ hợp có thể.

PE backend bao gồm mạng scatter để phân phối các kết quả nhân từ PE frontend đến các vị trí phù hợp. Một cơ chế để cộng dồn các tổng riêng được triển khai, cho phép tích lũy các kết quả nhân đã được phân tán về các vị trí đầu ra tương ứng.

Luồng dữ liệu Input Stationary cho phép các kích hoạt đầu vào được giữ cố định trong bộ nhớ, trong khi các trọng số được luân chuyển. Các phép nhân được thực hiện đồng thời, kết quả được phân tán qua mạng scatter và sau đó được cộng dồn tại các vị trí đầu ra.

Ưu điểm chính của Sparse CNN bao gồm:

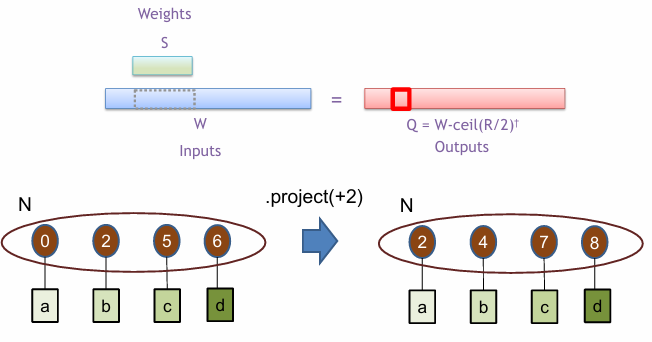
1. Tiết kiệm bộ nhớ thông qua việc nén dữ liệu thưa
2. Giảm thiểu tiêu thụ năng lượng bằng cách chỉ xử lý các giá trị khác không
3. Tăng tốc độ tính toán thông qua kiến trúc được tối ưu hóa cho dữ liệu thưa

Kiến trúc này đặc biệt có giá trị cho các mạng CNN hiện đại, nơi tính thưa thớt xuất hiện một cách tự nhiên hoặc được tạo ra thông qua các kỹ thuật như pruning (tỉa) và lượng tử hóa. SCNN đại diện cho một bước tiến quan trọng trong việc tối ưu hóa các mạng học sâu, mở ra những khả năng mới trong xử lý hiệu quả các mô hình phức tạp.

### 6.4.2 Phép chiếu và giao tọa độ sợi

1. Fiber Coordinate Projection (Phép chiếu tọa độ sợi)

Phép chiếu tọa độ sợi là một kỹ thuật tinh vi trong xử lý dữ liệu thưa, minh họa mối quan hệ phức tạp giữa trọng số, đầu vào và đầu ra trong các mạng nơ-ron. Công thức Q = W-ceil(R/2) cho thấy cách tính toán tinh tế các giá trị đầu ra dựa trên các tọa độ và giá trị ban đầu.

****

Hình 6.19: Minh họa phép chiếu tọa độ Fiber – tịnh tiến vị trí

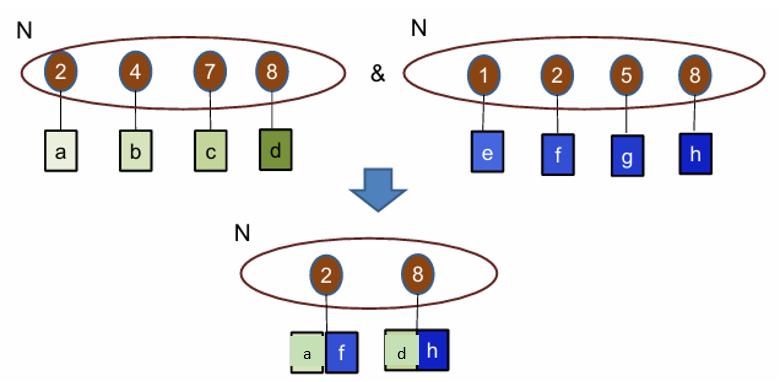
Quá trình chiếu diễn ra bằng cách áp dụng một phép tịnh tiến lên các tọa độ. Ví dụ, với một tập tọa độ ban đầu [0, 2, 5, 6], phép chiếu .project(+2) sẽ chuyển đổi chúng thành [2, 4, 7, 8]. Điều quan trọng là các giá trị gốc vẫn được giữ nguyên, chỉ có vị trí của chúng thay đổi.

Câu hỏi then chốt là: "Liệu phép chiếu có đòi hỏi phần cứng phức tạp không?" Câu trả lời phụ thuộc hoàn toàn vào cách biểu diễn dữ liệu. Tính phức tạp của phần cứng được xác định bởi cấu trúc và phương thức lưu trữ dữ liệu.

Ý nghĩa sâu xa của phép chiếu nằm ở khả năng chuyển đổi linh hoạt các tọa độ mà không cần xây dựng lại toàn bộ cấu trúc dữ liệu. Đây là một kỹ thuật quan trọng trong việc tối ưu hóa xử lý các mạng thưa, cho phép các nhà nghiên cứu thao tác hiệu quả với các cấu trúc dữ liệu phức tạp.

2. Fiber Intersection (Giao của sợi)

Phép giao sợi là một kỹ thuật tiên tiến trong xử lý dữ liệu thưa, tập trung vào việc tìm ra các phần tử chung giữa hai tập hợp khác nhau. Kỹ thuật này đặc biệt hữu ích trong việc kết hợp thông tin từ nhiều nguồn và giảm lượng dữ liệu cần xử lý.



Hình 6.20: Minh họa phép chiếu tọa độ Fiber – Tìm phần tử chung

Với hai tập hợp có các tọa độ và giá trị khác nhau, phép giao sẽ xác định các điểm chung. Ví dụ, với hai tập [2, 4, 7, 8] và [1, 2, 5, 8], phép giao sẽ cho ra [2, 8] - những tọa độ xuất hiện ở cả hai tập.

Một lần nữa, câu hỏi về độ phức tạp phần cứng lại được đặt ra. Và một lần nữa, câu trả lời phụ thuộc hoàn toàn vào cách biểu diễn dữ liệu.

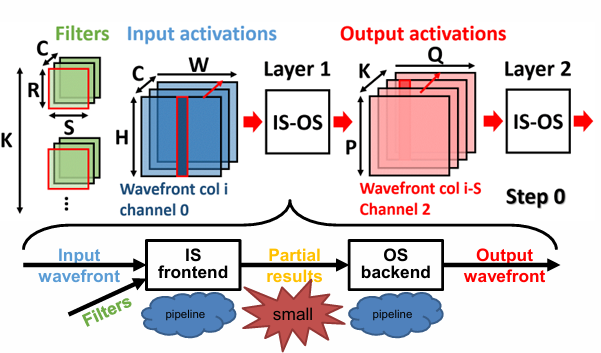
Ý nghĩa của phép giao nằm ở khả năng:

1. Tìm kiếm nhanh chóng các phần tử chung
2. Kết hợp thông tin từ nhiều nguồn
3. Giảm thiểu lượng dữ liệu cần xử lý

Cả hai phép toán - chiếu và giao - đều là những công cụ quan trọng trong xử lý dữ liệu thưa, mở ra khả năng mới trong việc xử lý và phân tích thông tin phức tạp.

### 6.4.3 Mô hình luồng dữ liệu IS-OS

Mô hình luồng dữ liệu Input-Stationary và Output-Stationary (IS-OS) là một kiến trúc tiên tiến trong xử lý mạng nơ-ron, được thiết kế để tối ưu hóa hiệu suất tính toán và sử dụng bộ nhớ.

****

Hình 6.21: Kiến trúc luồng dữ liệu IS-OS

Kiến trúc này bao gồm các thành phần chính: bộ lọc (Filters) với kích thước R×C×K, kích hoạt đầu vào (Input activations) với kích thước C×W×H, và kích hoạt đầu ra (Output activations) với kích thước K×Q×P. Mỗi thành phần đóng vai trò quan trọng trong quá trình xử lý dữ liệu.

Ở Layer 1, mô hình IS-OS xử lý đầu vào dạng "Wavefront col i channel 0" và tạo ra đầu ra dạng "Wavefront col i-S channel 2". Quá trình này diễn ra thông qua hai giai đoạn chính: IS frontend và OS backend.

IS frontend được tối ưu để xử lý mặt sóng đầu vào, giữ các giá trị đầu vào cố định trong bộ nhớ. Điều này giúp tối ưu hóa đường ống xử lý và giảm thiểu chi phí truy cập bộ nhớ. Các kết quả trung gian lưu trữ với dung lượng bộ nhớ tối thiểu.

OS backend tập trung vào việc xử lý các kết quả một phần để tạo ra mặt sóng đầu ra. Bằng cách giữ các giá trị đầu ra cố định, phương pháp này tiếp tục tối ưu hóa đường ống xử lý và giảm thiểu chi phí tính toán.

Ưu điểm chính của kiến trúc IS-OS:

1. Kết hợp các ưu điểm của cả Input-Stationary và Output-Stationary
2. Giảm thiểu chi phí truy cập bộ nhớ
3. Chỉ lưu trữ kết quả một phần, tiết kiệm bộ nhớ
4. Tối ưu hóa xử lý dạng mặt sóng (wavefront)

Kiến trúc này đặc biệt hiệu quả trong việc xử lý các mạng nơ-ron lớn trên phần cứng chuyên dụng. Bằng cách tối ưu hóa cả tốc độ xử lý và tiêu thụ năng lượng, IS-OS mở ra những khả năng mới trong tính toán học sâu, đặc biệt là trên các hệ thống có tài nguyên hạn chế.

Sự phát triển của IS-OS reflect xu hướng không ngừng tìm kiếm các phương pháp tính toán hiệu quả hơn trong lĩnh vực trí tuệ nhân tạo, nơi mà mỗi bit, mỗi chu kỳ xử lý đều có giá trị to lớn

# CHƯƠNG 7. SPARSE MATRIX MULTIPLICATION ACCELERATOR ARCHITECTURE

## 7.1. Fully-Connected (FC) Layer

Tầng fully-connected trong mạng nơ-ron (mọi nơ-ron đều kết nối với nhau) có thể được biểu diễn dưới dạng phép nhân ma trận. Ban đầu, dữ liệu đầu vào (input) và bộ lọc (filters) có nhiều chiều (ví dụ: kênh, chiều cao, chiều rộng). Nhưng bằng cách "làm phẳng" (flatten) chúng, ta biến nó thành một phép nhân ma trận đơn giản: Z=A×B, trong đó là đầu vào, B là trọng số, và Z là đầu ra.

A diagram of different colored squares

AI-generated content may be incorrect.

Hình 7.1: Lớp kết nối đầy đủ

## 7.2. Einsum

Einsum là một cách viết gọn gàng để mô tả phép nhân ma trận hoặc tensor. Ví dụ: Zm,n=Am,k×Bn,k có nghĩa là nhân hai ma trận A và B để ra Z. Nó giúp ta hiểu rõ cách các chiều (indices) tương tác với nhau.

1. Shared indices -> Intersection (&): Các chiều giống nhau giữa A và B (như k) là nơi hai ma trận "gặp nhau". Giao nhau này giống như tìm những phần tử chung để nhân với nhau.
2. Contracted indices -> Reduction (+=): Chiều chung k sẽ được "thu gọn" bằng cách cộng tất cả các kết quả nhân lại với nhau. Ví dụ, tổng trên k k để từ Am,k×Bn,k ra Zm,n
3. Uncontracted indices -> Populate output point (<<): Các chiều không bị thu gọn (như m và n) quyết định vị trí của kết quả trong ma trận đầu ra Z. Chúng "điền" giá trị vào Zm,n.

A diagram of a fiber intersection

AI-generated content may be incorrect.

Hình 7.2: Chỉ số chia sẻ tác động đến giao nhau

A diagram of a machine

AI-generated content may be incorrect.

Hình 7.3: Chỉ số không được thu gọn ảnh hưởng đến đầu ra

Index arithmetic -> Projection: Trong trường hợp phức tạp hơn (như phép cuốn tích), các chỉ số có thể cộng trừ (ví dụ: p+r,q+s) để "chiếu" dữ liệu từ vị trí này sang vị trí khác, giúp ánh xạ đầu vào sang đầu ra.

Phần này biến một tầng FC thành phép nhân ma trận, dùng Einsum để mô tả, và phân tích cách các chiều làm việc cùng nhau: giao nhau để nhân, thu gọn để cộng, điền để tạo đầu ra, và chiếu để xử lý vị trí.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 7.4: Biểu diễn phép nhân ma trận thưa

## 7.3. ExTensor - Tổng quan

ExTensor là một kiến trúc phần cứng được thiết kế để tăng tốc xử lý đại số tensor thưa (sparse tensor algebra), dựa trên bài báo tại MICRO 2019. Nó tập trung vào việc tận dụng tính thưa (nhiều phần tử bằng 0) để giảm tính toán và tăng hiệu quả trong học sâu.

1. Phân cấp lát thưa (Hierarchical Sparse Tiling):

Chia tensor lớn thành các "lát" (tiles) nhỏ hơn để xử lý từng phần, giúp tiết kiệm bộ nhớ và tối ưu truy cập dữ liệu.

2. Giao nhau phân cấp (Hierarchical Intersection):

Tìm các phần tử chung (khác 0) giữa các tensor ở nhiều cấp độ khác nhau, thay vì duyệt toàn bộ, để giảm lãng phí tính toán.

3. Đơn vị giao nhau tối ưu hóa:

Một bộ phận phần cứng đặc biệt được thiết kế để thực hiện phép giao nhau nhanh và hiệu quả.

## 7.4. Quan sát từ hoạt hình hai cấp

Quá trình xử lý dữ liệu trong mô hình này thể hiện sự khác biệt rõ rệt giữa hai tensor đầu vào:

1. Tensor A được duyệt một cách hiệu quả, chỉ cần truy cập mỗi lát dữ liệu một lần duy nhất
2. Tensor B yêu cầu duyệt lặp lại nhiều lần qua cùng các lát dữ liệu

Cả hai tensor đều tuân thủ nguyên tắc xử lý tuần tự từng lát, hoàn thành xong một lát trước khi chuyển sang lát tiếp theo. Trong quá trình tạo đầu ra, các lát kết quả được hình thành tuần tự, tuy nhiên một số vị trí đầu ra (ví dụ như (0,0)) có thể bị thiếu do ảnh hưởng của tính chất thưa trong dữ liệu. Điều này phản ánh đặc điểm quan trọng của mô hình khi xử lý các tensor thưa, nơi không phải tất cả các vị trí đầu ra đều được tạo ra do không có dữ liệu đầu vào tương ứng.

ExTensor giúp xử lý tensor thưa nhanh hơn bằng cách chia nhỏ vấn đề, tập trung vào các phần tử quan trọng (khác 0), và dùng phần cứng chuyên dụng để tối ưu giao nhau. Điều này rất hữu ích cho các ứng dụng học sâu cần tính toán hiệu quả.

## 7.5. OuterSPACE - Tổng quan

OuterSPACE là một kiến trúc phần cứng được giới thiệu trong bài báo tại HPCA 2018, tập trung vào tăng tốc phép nhân ma trận thưa bằng cách sử dụng tích ngoài (outer product) thay vì tích trong thông thường. Nó tận dụng tính thưa để giảm tính toán và tối ưu hiệu suất.

A diagram of a diagram

AI-generated content may be incorrect.

Hình 7.5: Thiết kế OuterSpace

OuterSPACE chia nhỏ phép nhân ma trận thưa thành hai giai đoạn, tận dụng tích ngoài để xử lý dữ liệu thưa hiệu quả, giảm chi phí tính toán và tăng khả năng song song. Tuy nhiên, nó cần quản lý cẩn thận thứ tự duyệt để tránh xung đột bộ nhớ.

## 7.6. Gamma:

### 7.6.1. Tạo Tensor Trung gian T

Tensor trung gian T được tạo thông qua phép toán right(Am,k, Bk,n) sử dụng tích ngoài hoặc giao nhau giữa các phần tử. Quá trình thực hiện theo thứ tự duyệt M (chậm nhất), K, N (nhanh nhất), với khả năng song song hóa chủ yếu trên chiều K (các sợi K) và một phần trên chiều M.

1. Tensor A được duyệt tuần tự một lần theo thứ tự tự nhiên
2. Các cột Bn của tensor B được truy xuất lặp lại nhiều lần
3. Với mỗi giá trị M, xử lý song song trên K tạo ra các sợi Tn đồng bộ

Tensor T[M,K,N] có thể chuyển đổi thứ hạng thành T'[M,N,K] nhờ tính chất đồng bộ của các sợi Tn và xử lý song song trên K, đảm bảo các phần tử cần thiết luôn sẵn sàng cho việc chuyển đổi.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 7.6: Tensor trung gian T

### 7.6.2. Rank Swizzled T

Quá trình chuyển đổi tensor từ T[M,K,N] sang T′[M,N,K] (Rank Swizzle) nhằm tối ưu hóa luồng xử lý. Việc sắp xếp lại thứ tự chiều này mang lại hai lợi ích chính:

Đồng bộ hóa truy xuất: Đảm bảo các bước xử lý tiếp theo có thể duyệt dữ liệu một cách tuần tự và hiệu quả

Tối ưu pipeline: Giảm thiểu tình trạng nghẽn cổ chai và lãng phí bộ nhớ bằng cách sắp xếp lại dữ liệu phù hợp với mô hình tính toán

Kỹ thuật này đặc biệt quan trọng trong các hệ thống xử lý tensor, giúp cân bằng giữa hiệu suất tính toán và yêu cầu về bộ nhớ. Việc chuyển đổi thứ hạng cho phép dữ liệu được tổ chức lại theo cách tối ưu nhất cho các phép toán tiếp theo trong pipeline xử lý.

A diagram of a diagram of a structure

AI-generated content may be incorrect.

Hình 7.7: Rank Swizzled T

### 7.6.3. FiberCache

FiberCache đóng vai trò như bộ đệm sợi, tách biệt quá trình đọc dữ liệu với độ trễ bộ nhớ. Bằng cách sử dụng bộ đệm nhỏ, giải pháp này giảm đáng kể thời gian chờ (roundtrip latency), đặc biệt hiệu quả trong các trường hợp độ trễ bộ nhớ thấp.

Gamma áp dụng thuật toán Gustavson để xử lý ma trận thưa một cách tối ưu, kết hợp khéo léo ba yếu tố: pipeline xử lý, song song hóa và quản lý bộ nhớ thông minh. Kết quả thử nghiệm trên các ma trận thưa phổ biến cho thấy Gamma (với tiền xử lý) đạt hiệu suất vượt trội so với Intel MKL, OuterSPACE và SpArch, chứng minh tính ưu việt của giải pháp này trong xử lý dữ liệu thưa.

## 7.7. Mô hình hóa các luồng dữ liệu thưa thớt – TeAAL

TeAAL giống như một "bản đồ" để hiểu và thử nghiệm các cách tổ chức dữ liệu thưa trong phần cứng.

Nó cho phép người thiết kế "vẽ" trước cách tính toán (Einsum + lịch trình) và suy ra cách thực thi (loop nest), giúp tối ưu hiệu suất và năng lượng.

A diagram of a computer

AI-generated content may be incorrect.

Hình 7.8: Mô hình TeAAL

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 8.1. Quá trình học tập và tìm hiểu

Trong suốt quá trình thực hiện Đồ án 1, nhóm chúng em đã có cơ hội tiếp cận, nghiên cứu và hệ thống lại nhiều kiến thức quan trọng liên quan đến thiết kế phần cứng dành cho học sâu (Deep Learning). Trọng tâm của đồ án là việc thiết kế bộ tăng tốc phần cứng tối ưu cho việc xử lý ma trận thưa, thông qua các kỹ thuật như Run-Length Encoding (RLE) và các chiến lược dataflow tiết kiệm năng lượng.

Nguồn tài liệu chính mà nhóm sử dụng trong quá trình tìm hiểu là cuốn sách chuyên khảo "Hardware Architectures for Deep Learning", từ đó nhóm đã:

1. Học được cách phân tích các đặc điểm cấu trúc phần cứng như spatial vs temporal architecture, memory hierarchy, và data reuse.
2. Hiểu rõ hơn về cách ánh xạ các phép toán học sâu như tích chập (convolution), nhân ma trận lên phần cứng.
3. Nắm được các kỹ thuật tối ưu hóa tính toán như pruning, sparsity-aware scheduling, và encoding.

Ngoài ra, nhóm cũng tham khảo các bài báo khoa học và slide bài giảng được giảng viên cung cấp để làm rõ hơn các nội dung khó.

## 8.2. Đánh giá Đồ án 1

Đồ án 1 không tập trung vào việc xây dựng hệ thống hoàn chỉnh, mà là giai đoạn nghiên cứu và tổng hợp kiến thức lý thuyết chuyên sâu để làm nền tảng cho việc triển khai cụ thể trong Đồ án 2. Qua đồ án này, nhóm đã có nhận thức sâu sắc về:

1. Vai trò của phần cứng chuyên biệt trong các hệ thống trí tuệ nhân tạo hiện đại.
2. Tầm quan trọng của độ thưa (sparsity) trong việc tiết kiệm tài nguyên tính toán và năng lượng.
3. Khả năng ứng dụng của kỹ thuật nén không mất mát như RLE trong việc tối ưu mô hình học sâu.

## 8.3. Hướng phát triển trong Đồ án 2

Từ các kiến thức đã nghiên cứu, trong Đồ án 2, nhóm dự định sẽ:

1. Thiết kế và mô phỏng một mô hình phần cứng thực tế, áp dụng kỹ thuật RLE vào pipeline xử lý ma trận thưa.
2. Ánh xạ một đoạn mã mạng nơ-ron đơn giản (ví dụ: LeNet hoặc MobileNet đã pruning) vào kiến trúc phần cứng đề xuất.
3. So sánh hiệu suất thực thi với các phương pháp không nén để kiểm chứng lợi ích của phương pháp đề xuất.
4. Xây dựng báo cáo chi tiết, có thể sử dụng các công cụ mô phỏng như Vivado, Quartus hoặc tích hợp với môi trường Python (sử dụng PyTorch hoặc TensorFlow Lite).

# CHƯƠNG 9: THIẾT KẾ DATAPATH CHO MÔ HÌNH

## 9.1. Tìm hiểu về thuật toán CSR:

Phương pháp nén CSR (Compressed Sparse Row), hay còn gọi là Compressed Row Storage, là một kỹ thuật lưu trữ ma trận thưa (sparse matrix) nhằm tiết kiệm bộ nhớ bằng cách chỉ lưu các phần tử khác không (non-zero) của ma trận. CSR được sử dụng phổ biến trong các ứng dụng tính toán khoa học, đồ họa máy tính, và học máy, nơi mà ma trận có kích thước lớn nhưng phần lớn các phần tử là số 0.

### 9.1.1. Định nghĩa và cách hoạt động

CSR biểu diễn một ma trận thưa A kích thước m×n (với mhàng và n cột) bằng ba mảng dữ liệu chính:

1. **Mảng giá trị (values)**: Lưu tất cả các phần tử khác không của ma trận, được sắp xếp theo thứ tự duyệt từng hàng (row-major order).
2. **Mảng chỉ số cột (column indices)**: Lưu chỉ số cột tương ứng của mỗi phần tử khác không trong mảng giá trị.
3. **Mảng con trỏ hàng (row pointers)**: Lưu vị trí bắt đầu của mỗi hàng trong mảng giá trị (và mảng chỉ số cột). Mảng này có kích thước m+1, trong đó phần tử cuối cùng chỉ đến vị trí kết thúc của ma trận.

Ví dụ

Xét ma trận thưa 4×5:

A = [5, 0, 0, 8, 0],

[0, 0, 3, 0, 0],

[0, 6, 0, 0, 9],

[0, 0, 0, 0, 0]

Các phần tử khác không: (0,0,5),(0,3,8),(1,2,3),(2,1,6),(2,4,9).

CSR biểu diễn:

* **Values**: [5,8,3,6,9]
* **Col\_indices**: [0,3,2,1,4]
* **Row\_ptr**: [0,2,3,5,5]

**Ưu điểm**

1. **Tiết kiệm bộ nhớ**: Chỉ lưu các phần tử khác không, giảm đáng kể dung lượng so với lưu toàn bộ ma trận.
2. **Hiệu quả tính toán**: Phù hợp cho các phép toán ma trận thưa như nhân ma trận-vectơ, giải hệ phương trình tuyến tính.
3. **Dễ truy cập hàng**: Truy cập các phần tử của một hàng cụ thể nhanh chóng nhờ mảng Row\_ptr.

**Nhược điểm**

1. **Khó sửa đổi ma trận**: Thêm hoặc xóa phần tử khác không yêu cầu tái cấu trúc mảng, làm tăng chi phí tính toán.
2. **Truy cập cột chậm**: Vì dữ liệu được tổ chức theo hàng, truy cập cột không hiệu quả bằng các định dạng khác như CSC (Compressed Sparse Column).

**Ứng dụng**

1. Giải các hệ phương trình tuyến tính trong phương pháp số (ví dụ: phương pháp phần tử hữu hạn).
2. Xử lý đồ thị (ma trận kề thưa).
3. Học máy (ví dụ: lưu trữ đặc trưng thưa trong các mô hình như SVM hoặc hồi quy tuyến tính).

## 9.2. Biểu diễn thuật toán CSR bằng Datapath

# TÀI LIỆU THAM KHẢO

[1] [6.5930/1 Hardware Architecture for Deep Learning - Spring 2024](https://csg.csail.mit.edu/6.5930/lecnotes.html).

[2] Kuon, Ian, Russell Tessier, and Jonathan Rose. ”FPGA architecture: Survey and challenges." Foundations and Trends in Electronic Design Automation 2.2 (2008): 135-253.

[3] Moore, “Cramming more components onto integrated circuits”, Electronics 1965.

[4] Moore, Progress in digital integrated electronics, IEDM 1975.

[5] Leiserson et al., There's Plenty of Room at the Top, Science.

[6] Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974.

[7] Leiserson et al., There's Plenty of Room at the Top, Science.

[8] Parashar, et.al., SCNN, ISCA 2017.

[9] Hegde, et.al., ExTensor, MICRO 2019.