國立清華大學 電機工程學系
實作專題研究成果報告

Performance Optimization of Stable Diffusion on Tenstorrent Wormhole N150s

基於 Tenstorrent Wormhole N150s 之 Stable Diffusion 生成性能優化

專題領域： 系統領域

組　　別：B608
指導教授：呂仁碩
組員姓名：劉彥廷、宋律德、蕭楷桐、林士鈞
研究期間：114 年 3 月 1 日至 114 年 11 月 19 日止，共 8 個月

Abstract

This project aims to explore the computational performance and optimization potential of the AI image generation model (Stable Diffusion) on the dedicated acceleration platform Tenstorrent Wormhole N150s. As generative models continue to scale up, their computational and memory bandwidth requirements during the inference stage have increased dramatically, making performance improvement under fixed hardware resources a critical issue. To address this, this study conducts an in-depth analysis of the Tenstorrent accelerator's hardware architecture, core L1 SRAM configuration, and dataflow characteristics, and performs core kernel-level performance optimization for model convolution operations.

The analysis reveals that during the Block-Sharded Conv2D computation stage of Stable Diffusion's UNet module, the excessively large circular buffers for activations and weights create L1 memory pressure and reduce data overlap efficiency, thereby limiting the application of double-buffering and pipelining. To resolve this bottleneck, this study introduces activation matrix slicing along the kernel height direction (slicing-by-kernel-height) based on methods proposed in official forums, enabling Conv2D to perform block computation and data scheduling at a finer granularity. This improvement reduces L1 memory footprint and allows more parallel execution of computation and data transfer.

Experimental results show that under the same model and generation settings (512×512, 50 steps), the single image generation time decreased from 5.9606 seconds to 5.5778 seconds, achieving a performance improvement of approximately 6.4%. Significant inference time reduction was achieved without sacrificing image quality. The research findings not only demonstrate the sustainable optimization potential of AI models on heterogeneous acceleration platforms but also provide practical experience in kernel partitioning and memory scheduling design, which can serve as a reference for future AI accelerator performance optimization.

摘要

本專題旨在探討 **AI 圖片生成模型（Stable Diffusion）在專用加速平台 Tenstorrent Wormhole N150s 上之運算效能與最佳化潛力**。隨著生成式模型規模不斷擴大，其在推論階段對運算與記憶體頻寬的需求急遽上升，如何在固定硬體資源下提升效能成為關鍵議題。為此，本研究深入分析 Tenstorrent 加速器之**硬體架構、核心 L1 SRAM 配置及 dataflow 特性**，並針對模型運算卷積進行**核心 kernel 層級的效能優化**。

在分析中發現，Stable Diffusion 之 UNet 模組於 Block-Sharded Conv2D 運算階段中，因 activation 與 weight 的 circular buffer 過於龐大，造成 L1 記憶體壓力與資料重疊效率下降，進而限制了雙緩衝（double-buffering）與 pipeline 的應用。為解決此瓶頸，本研究依據官方討論區提出之方法，**引入 activation matrix 沿 kernel 高度方向的切片(slicing-by-kernel-height)**，使 Conv2D 能以更細粒度進行區塊計算與資料排程。此改進降低 L1 記憶體占用，並允許更多運算與傳輸並行執行。

實驗結果顯示，於相同模型與生成設定（512×512、50 steps）下，單張影像生成時間由 5.9606 秒降至 5.5778 秒，效能提升 約 6.4%。在無犧牲影像品質的前提下，達成了顯著的推論時間縮減。研究成果不僅展現了 AI 模型在異質加速平台上之可持續優化潛力，亦提供了 kernel 切分與 memory 排程設計的實務經驗，可作為後續 AI 加速卡效能優化之參考。

1. Introduction

生成式人工智慧（Generative AI）近年快速發展，其中以 Stable Diffusion 為代表的潛在擴散模型（Latent Diffusion Models, LDMs）成為影像生成主流。此類模型在推論階段包含**大量卷積運算**、attention 與頻繁的 feature map 傳輸，對運算與記憶體頻寬造成巨大負擔。雖然 GPU 仍主導生成式模型推論，但專為 AI 設計的異質加速平台（如 Tenstorrent Wormhole N150s）具備更高能源效率與可擴展性。然而，由於其記憶體架構與 dataflow 與 GPU 大異其趣，如何在此平台上發揮模型效能成為具實務價值的研究挑戰。

本專題旨在探討 Stable Diffusion—AI 模型在專用 AI 加速器上的效能限制與最佳化策略。我們首先分析 Tenstorrent Wormhole N150s 之硬體架構，包括其獨特的 Tensix core、每核心獨立之 L1 SRAM、資料交換路徑以及 ttnn / tt-metal 的軟體執行模式。透過 profiling 工具及 kernel-level trace，我們發現 UNet 模組中的 Block-Sharded Conv2D 是運算時間占比最高的運算，而其主要瓶頸來自：

- activation 與 weight buffer 對 L1 SRAM 造成較高壓力，限制資料重疊與雙緩衝（double buffering）的使用。
- Conv2D 內部 dataflow 需分配到多個 circular buffer，使得大型 activation matrix 無法在單一核心內高效排程。
- Block-Sharded 演算法在固定切分策略下，欠缺對 L1 容量的彈性調整空間。

為解決上述問題，本專題採用並進一步驗證 Tenstorrent 官方 PR #25805 所提出的改進方法，即將 Block-Sharded Conv2D 的 activation matrix 依據 kernel height 進行切片（slicing-by-kernel-height）。此方法將原本的 activation block 拆分成更細粒度的片段，以提升 compute 與 資料傳輸的效率。我們在此基礎上進行效能實測，並研究此 slicing 策略對計算正確性與影像品質的影響。

為確實呈現研究貢獻，本專題在此明確區分自行實作項目與外部參考內容：

自行執行部分：

- Profiling Stable Diffusion 在 Wormhole N150s 上的推論流程
- 實作大量實測，包括執行時間、生成品質驗證
- 評估 PR #25805 slicing 方法對實際推論效率的提升
- 整合結果、整理資料、撰寫效能分析報告

參考 Tenstorrent 與網路來源部分：

- Tenstorrent 官方程式庫 ttnn / tt-metal 核心程式碼架構

- GitHub 上 PR #25805 提出的 slicing-by-kernel-height 演算法
- Stable Diffusion 相關模型文獻與論文（如 LDM 原始論文）
- Wormhole N150s 之硬體規格文件與公開技術簡介

比較結果顯示，透過此 slicing 優化策略，Stable Diffusion（SDXL Base, 512×512, 50 steps）**單張影像的生成時間從 5.9606 秒降低至 5.5778 秒，效能提升達 6.4%，且生成品質一致，證實該方法能優化運算瓶頸並提升整體執行效率。**此成果展現了 AI 模型在異質加速平台上仍具有可觀的最佳化空間，且 kernel-level 的資料流切分策略可作為未來 AI 加速器開發、排程演算法設計與軟硬體協同最佳化的重要參考。

2. Research Methodology

2-1. System Overview

A. Wormhole N150s Hardware Architecture

Wormhole 主要由網格化的計算單元（Tensix core）、DRAM 記憶體單元（GDDR6）和互聯單元（ETHERNET）三個重要部分組成。下為單一 Tensix core 的內部架構。

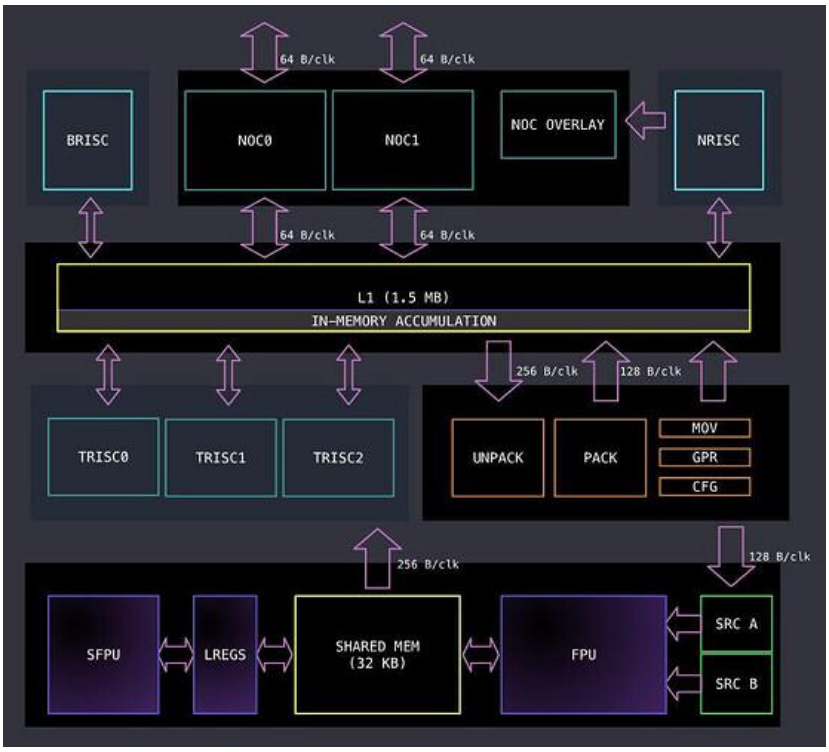

Fig. 2-1 Tensix core 內部架構(來源:Tenstorrent)

Tensix 核心是 Tenstorrent 晶片中最基礎的運算單元，其內部由多個專用處理模組組成，透過軟體協同運作以達成高效率 AI 運算。整體結構可分為以下幾個主要模組：

i. Baby RISC-V 核心（淺藍色區塊）

內含五個輕量級 RISC-V 核心，負責控制流程與部分計算：

- BRISC、NRISC：主要處理資料搬移與系統層級控制。
- TRISC0、TRISC1、TRISC2：負責驅動計算流程（分別負責 UNPACK、MATH、PACK）。

另外，核心內資料移動和計算的最小單位為 tile，1個 tile 大小為32*32。

ii. 資料傳輸模組（淺藍色）

包含 NOC0、NOC1 與 NOC Overlay，負責 Tensix 與主記憶體／其他核心之間的資料搬移，並在 NoC 上協調資料流動。

iii. 記憶體系統（黃色區塊）

Tensix 具備核心記憶體 L1 sram (1.5MB)：

L1 SRAM 是 Tensix 的高速運算緩衝區，負責存放輸入、權重、輸出 tile 與中間累加結果，並支援 UNPACK/MATH/PACK 的 pipeline 流程，是提升 FPU 利用率與降低 DRAM 流量的關鍵記憶體。

其中包含專門給 Baby RISC-V 核心用來在 kernel 間交換資料的 circular buffers

iv. 計算引擎（紫色）

Tensix 由兩大類運算模組組成：

- FPU（Tensor Math Engine）：進行矩陣乘法、**卷積**等密集張量計算，採 tile-based 模式並依賴 SRC A/B 作為來源。
- SFPU（Special Function PU）：處理 exp、sqrt、softmax、top-K 等特殊非線性算子。

Tensix FPU 運算流程

1. UNPACK：從 L1 將輸入 tile 載入 SRC A/B。
2. MATH：FPU 從 SRC 寄存器取值，執行矩陣乘法或張量運算，結果存入 DST。
3. PACK：將 DST 中的運算結果寫回 L1。(DST 寄存器並未畫入途中)

B. ttnn / tt-metal Execution Model

Tenstorrent 的軟體堆疊由四層組成：

TT-Forge（編譯）→ TT-NN（Python/C++ 算子）→ TT-Metal（提供 kernel 管理和資料搬移的底層 API）→ TT-LLK（Tensix 低階 Kernel）。

執行流程（以 Python 呼叫 ttnn.conv2d 為例）：

1. Python 呼叫 ttnn.conv2d；TT-NN 會建立輸入/權重 tensor，根據參數決定卷積的滑動視窗和輸出形狀。
2. TT-NN 根據使用者指定或內建設定決定 sharding 方式、每個 tile 的大小和需要幾個 Tensix core，再為 halo 和複製產生主機端配置。

3. TT-NN 透過 TT-Metal 的 API 建立並排程執行 kernel 程式
4. TT-LLK 在 Tensix 核上執行 kernel（UNPACK→MATH→PACK），以流水線方式處理多個 tile，提高吞吐量。
5. TT-Metal 在所有核心完成運算後回傳輸出 tensor 至 Python。

此流程串起前端 Python API 與底層 Tensix 硬體，使得一行 ttnn.conv2d 能在硬體上完成高效的 tile-based 卷積運算。

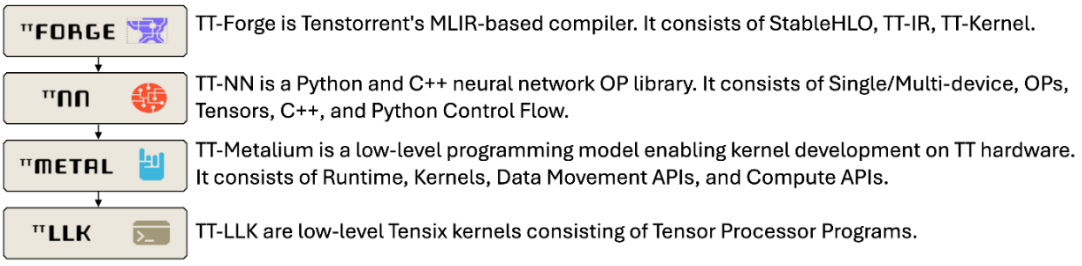

Fig. 2-2 Tenstorrent 軟體架構(來源:Tenstorrent)

2-2. Bottleneck Profiling

A. Profiling Method

本研究使用 Tenstorrent 官方推薦方式進行 operator-level profiling，即使用開源軟體 tracy。

其可取得每個 operation（如 Matmul、SliceDeviceOperation、HaloDeviceOperation）的裝置端執行時間（device kernel execution time）與執行內容（如 Matmul、SliceDeviceOperation、HaloDeviceOperation）

透過對這些 operation 進行加總、分類與排序，即可了解整個 SD pipeline 的運算熱點。雖然該數據並不直接等於最終 wall-clock latency，但能完整揭露 Device 端計算與資料搬移的時間分佈，是識別瓶頸的關鍵依據。

B. Profiling Result

對 Stable Diffusion 的 ops profiler 數據進行排序後顯示：

Top operations 之中，有超過一半的總時間來自 Block-Sharded conv2d（BS

conv2d）相關的 data movement。

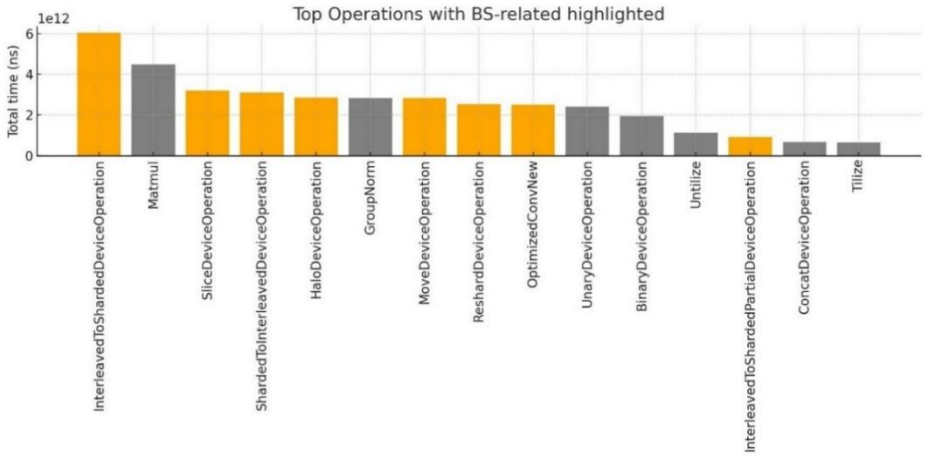

Fig. 2-3 各 operation 的執行時間 (橘色代表與 BS conv2d pipeline 相關)

從圖片中可觀察到以下事實：

i. InterleavedToShardedDeviceOperation（屬於 BS conv2d 前置步驟）為全模型中最耗時的 op，總時間超過 6×10^{12} ns。

ii. 其他 BS pipeline op，如 SliceDeviceOperation、HaloDeviceOperation、MoveDeviceOperation、ReshardDeviceOperation、ShardedToInterleavedDeviceOperation 亦名列前茅。

iii. Conv 本體 OptimizedConvNew 也具有極高的總時間。

C. Why BS conv2d is bottleneck?

Block-Sharded Conv2d（BS conv2d）是 Stable Diffusion UNet 中使用最頻繁、同時對硬體資源需求最高的運算模式。與一般的 matmul 或 non-sharded conv 不同，BS conv2d 涉及多種跨核心資料交換與 layout 轉換，因此形成相當複雜的資料搬移路徑，其成本幾乎高於純計算本身。

整體而言，BS conv2d 所需的資料分片、halo 交換、weight/activation 重組與 memory sharding 占據了模型執行時間的主要部分，是最明顯的效能瓶頸，正因如此，我們選擇了對 BS conv2d 進行優化。

2-3. Original Behavior

A. Original Behavior:

以下大綱描述 Block sharding conv2d 在 Wormhole N150s 上的原執行方式，包含以下：

i. 將 input tensor 切成 input block

ii. 將 input blocks 和 weight 切分為 input activation matrix 和 weight Matrix

iii. Per-Core Conv2D Execution Dataflow

i. 將 input tensor 切成 input block

在 UNet 的部分大型卷積（特別是 channel 數較大、且 feature map 高度較小的情況）中，系統會判定當 channel 數大於320時，以 Block Sharded 的形式來執行 Conv2d。Block Sharded 的流程如下:

首先將四維的 input tensor (batch, in_channel, height, width) (以下以 NCHW 簡寫)進行重排，變成 NHWC 的形式，再將其轉換成(N*H*W,C)的二維形式，以 NHW 為高、C 為寬，並根據輸入的 configuration 進行切分，讓不同的核心分別計算不同的資料區塊。這樣的切法有以下優點：

一、每個核心僅需處理部分 channels:減少 activation 與 weight 所需的 L1 占用。

二、核心利用率更高，NHW 與 C 同時切割後，資料分布較平均，改善 load imbalance。

三、較適用於高 channel 的 UNet 中後段 layer 因為 channel 數很大，切 C 能有效降低單核心負擔。

Block sharding 有以下兩個重要參數:

- Core Grid：代表會存放 tensor 分片的核心。每個核心都會擁有一個分片。
- Shard Shape：以元素數量來表示的「單一分片的形狀」。也就是每個核心上所保存的 tensor 子集合的形狀。

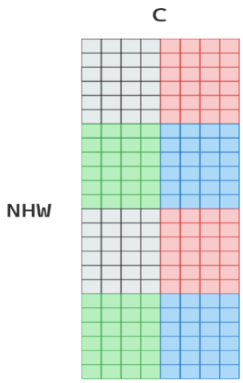

Fig.2-4 block sharded 後的 tensor(來源:tenstorrent)

Core grid = 4x2, shard shape = [5,4] in tile

在 BS conv2d 中，core grid = 8x4, shard shape 會隨 tensor 大小變化。

另外，input tensor 儲存在 DRAM 時並未經過 padding 及 halo，各 core 在讀取 input 的時候，會以 on-the-fly 的方式，各 core 在計算及抓取 data 時，自動補上應有的 padding 及 halo。

ii. 將 input blocks 和 weight 切分為 input activation matrix 和 weight matrix

Conv2d 程序在建立，初始化的同時會載入 kernel height(K_h), kernel width(K_w) 兩

個參數，此時 DRAM 中已準備好 input activation tensor$(N, C_i,$ $H_{ph}, W_{ph})$與 weight tensor(C_o, C_i, K_H, K_W)，其中，weight tensor 在被 writer 載入時會被攤平成一個二維矩陣，weight matrix，其大小為：

$$((K_h * K_w * C_i)\,,\, C_o)$$

而被 sharding 好的 input activation tensor(i.e. input block)會：

一、對C_i做 padding 至32的倍數 (tile 對齊)

二、將 input block 轉換成 input activation matrix，其 size 為

$$(bH_0, (K_h * K_w * C_i)\,)$$

其中$bH_0 = number\ of\ windows$

$$= N \times \frac{H_{original} + 2p_H - ((K_H - 1)d_H + 1)}{s_H} \times \frac{W_{original} + 2p_W - ((K_w - 1)d_W + 1)}{s_W}$$

，並且bH_0會 在載入時自動 padding 使得其$mod32 = 0$

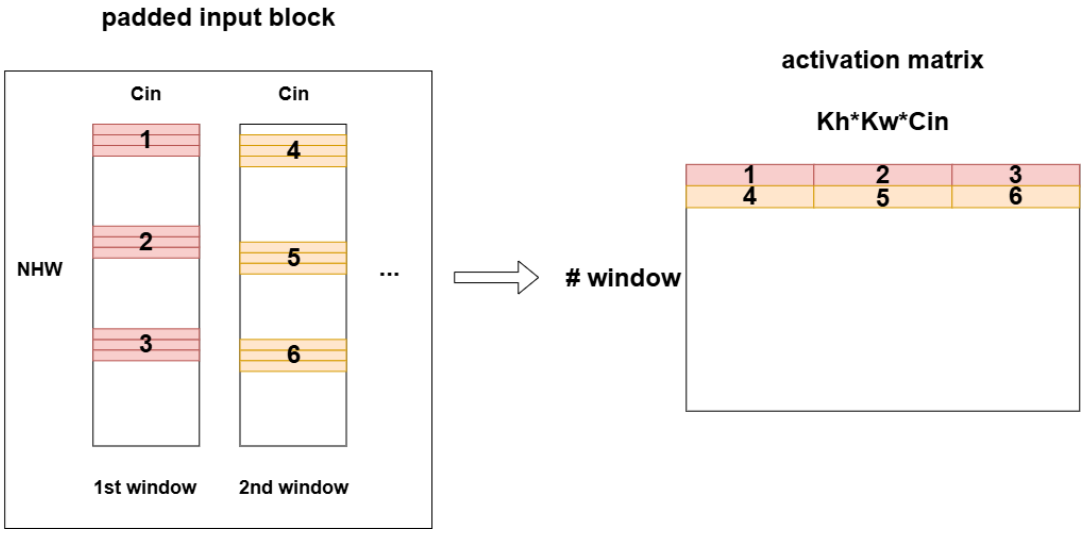

Fig.2-5 從 DRAM 取出 activation matrix

三、input activaiton matrix 和 weight matrix 做矩陣乘法得到結果

整體流程如下圖：

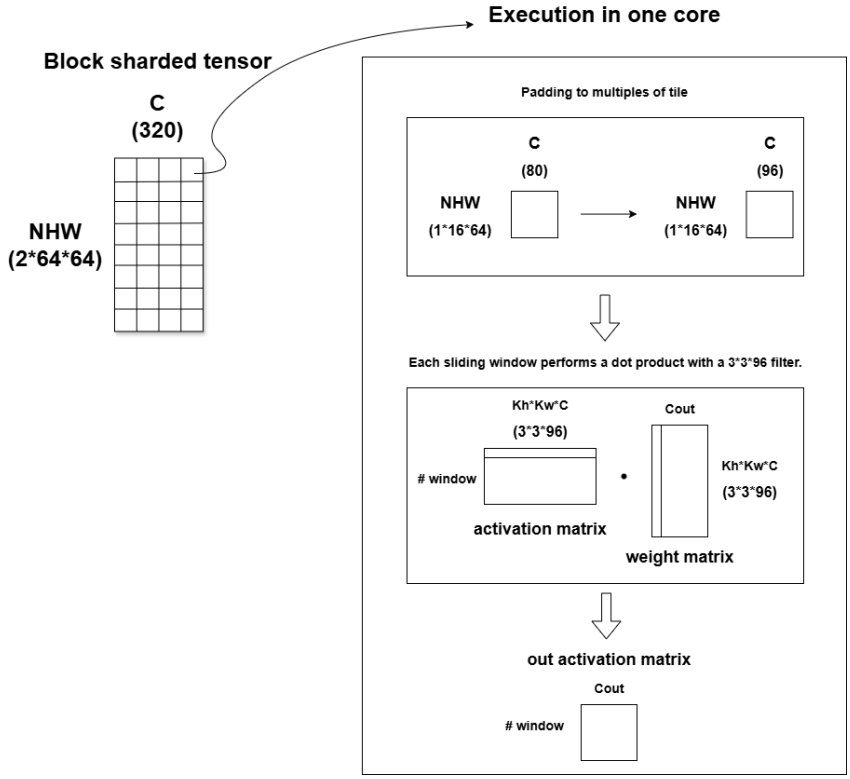

Fig.2-6 per core convolution calculations

如圖所示，activation matrix 的每個 row 代表一個 window，weight matrix 的每個 column 代表一個 filter(C_o)，兩矩陣相乘即獲得該 core 負責 block 的對應區塊之 output。

iii. Per-Core Conv2D Execution Dataflow

在資料搬移方面，兩個 data-movement 核心負責將該 core 所需的全部輸入資料載入至 L1 SRAM。其一為 activation reader kernel，它會從 Global DRAM 中載入對應 activation block，並以 DMA 的方式搬運至本地 L1 中的 circular buffer。另一個核心則執行 weights reader kernel，將 weight 載入至 L1 內部的另一 circulaNr buffer 中。

在計算部分，另外三個 compute 核心針對 L1 內的 activation 與 weight 共同執行 unpack、math、pack 等子任務。首先，由 unpack core 從 L1 取出 activation matrix 和 weight matrix，並存進 source registers。

接著由 math core 完成最主要的計算任務。它讀取 unpack 後的 activation matrix 和 weight matrix 進行矩陣乘法，同時計算出加上 bias 的部分輸出結果。隨著每個 block 的所有輸出位置被逐步計算完成，結果會被寫入 destination registers 中。

最後由 pack core 將 destination registers 中的 partial outputs 搬回 L1 的 output buffer，並由 Output Writer Kernel 寫回 global memory。

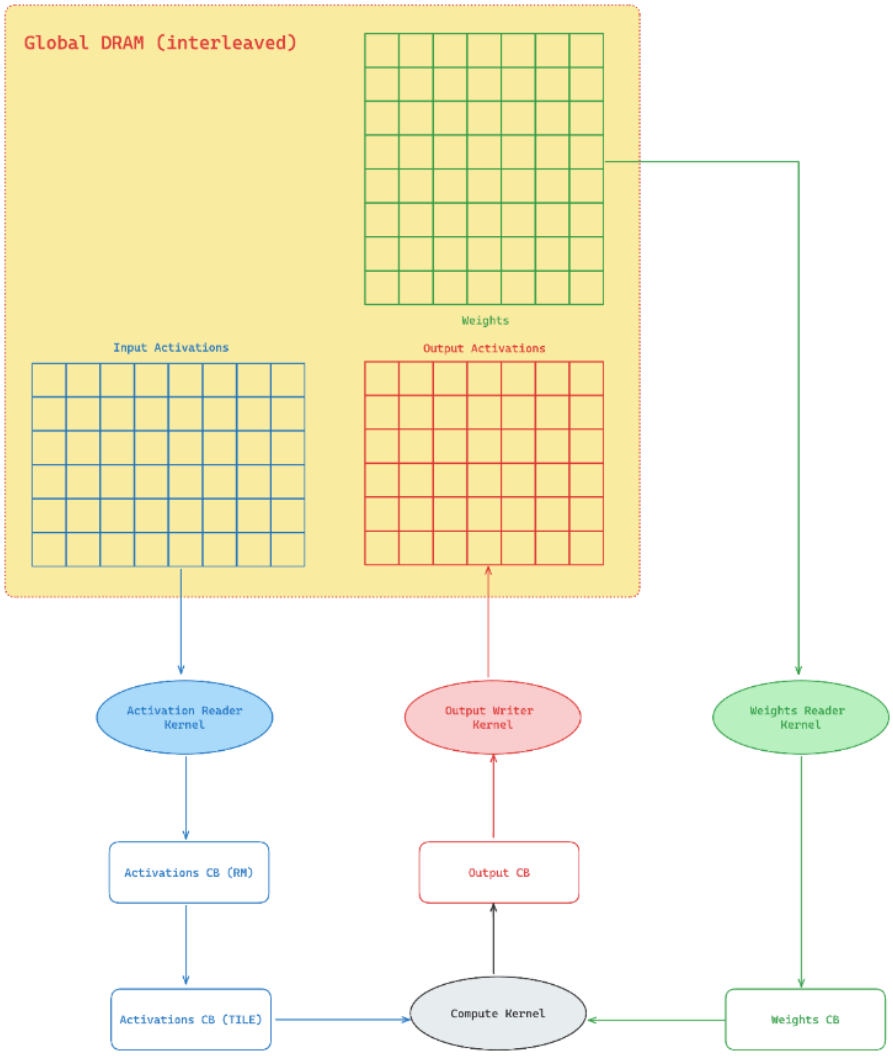

Fig.2-7 Per-core conv2d Dataflow

B. Problem: L1 Memory Pressure

原先的方式，在計算每個輸出 block 時，BS Conv2D 會一次性處理 activation matrix 和 weights matrix 的 inner dimension，也就是說，整個 $K_h \times K_w \times C_{in}$ 長度的 data 都必須被載入 L1，使得加總後的記憶體占用量非常高，讓 L1 SRAM 成為限制效能的核心因素之一。在 L1 空間受限的情況下，限制了計算併行度，也妨礙 double-buffering 等資料重疊機制的啟動。

並且，由於未切分的 activation matrix 和 weight matrix 可能無法完整裝進 L1 中，在計算時只會取需要的區塊存入 L1，這導致每次都需要從 DRAM 重新讀取，而從 DRAM 讀取資料相對費時，導致先前提到的 pipeline 可能會有 stall 的狀況，導致整題延遲變高。

2-4. Optimization Strategy

Proposed Method: Slicing Activation by Kernel Height

BS Conv2D 的主要效能瓶頸來自 activation matrix 與 weight matrix 在 L1 SRAM 中的

佔用量過大，導致單個 Tensix 核心無法同時容納所有運算所需資料，進而造成 double-buffering pipeline 無法完全展開。本研究採用並驗證 Tenstorrent 提出的「activation slicing by kernel height」概念，使 activation matrix 能夠以更小 size 載入，大幅降低 L1 需求並提升 dataflow 重疊效率。

本方法的核心概念為：
不再一次性讀入整個 kernel 的 inner dimension（$K_h \times K_w \times C_{in}$），而是依照 kernel height（Kh）進行切片，逐段載入 activation 的部分 window，並在每段計算後累加結果。

此方法有效地將每次計算所需的 activation 子矩陣縮小至原本的 $1/K_h$，如下述流程所示：

i. Activation Matrix、Weight Matrix 切片策略：

原始 Block-Sharded Conv2D 會先將每個 input block 轉換成大小為：

$$(bH_0, (K_h * K_w * C_i))$$

$$, bH_O = N \times \frac{H_{original} + 2p_H - ((K_H - 1)d_H + 1)}{s_H}$$

$$\times \frac{W_{original} + 2p_W - ((K_w - 1)d_W + 1)}{s_W}$$

的 input activation matrix。

並將每個 weight tensor re-permutation 成大小為：

$$((K_h * K_w * C_i), C_o)$$

的 weight matrix。

在考量 input activation matrix、weight matrix、output block 同時佔有 L1 SRAM 的情況下，有以下不等式須滿足：

$$[bH_0 \times C_o + K_h \times K_w \times C_i \times (bH_o + \mathrm{C_o})] \times sizeof\ (datatype) < L1_{free}$$

然而，在 UNet 底部，channel 數高的運算中($C_{in} > 320$)，Activation Matrix 和 Weight Matrix 龐大的C_{in}維度經常讓 L1 SRAM 不堪負荷，新的方法將 inner dimension 分拆為K_H份：

$$(K_H \cdot K_w \cdot C_{in}) \rightarrow K_H個(K_w \cdot C_{in})slice$$

此方法雖然使執行的次數乘上K_H倍，但切片後的 activation matrix 和 weight matrix 皆可存進 L1 中的 circular buffer，並常駐其中，如此在計算時便不需不斷從 DRAM 讀取資料，只有在換至下一個切片時需要進行 DMA，整體 pipeline stall 會減少，並且由於 L1的空間壓力減少可以啟用 double buffer，也就是在計算一個切

片時，就讀取另一切片進 L1，整體的執行效率也會增加。

ii. Per-Slice 計算與部分累加 (Partial Accumulation)

運算過程如 Fig 2-8所示：

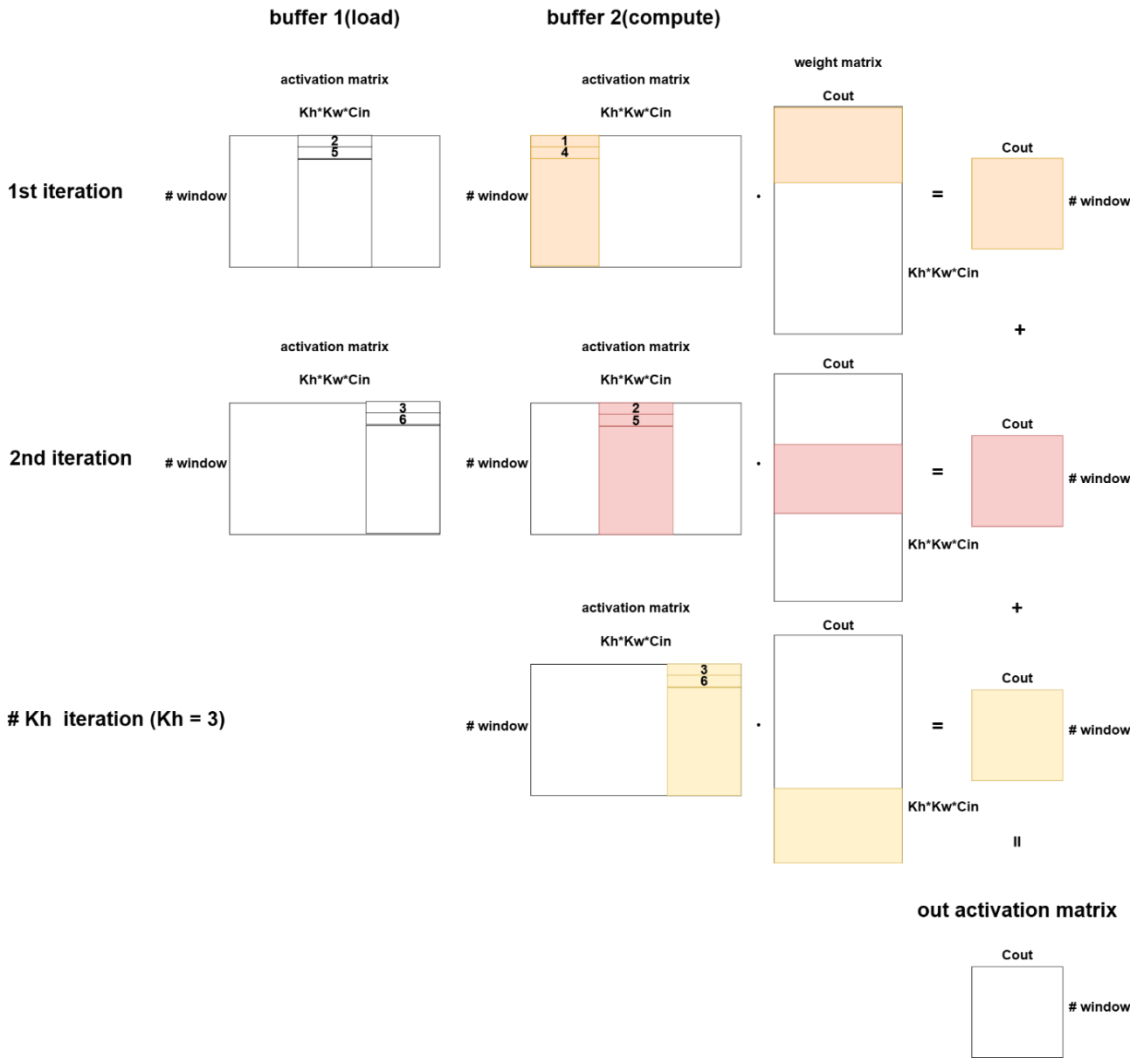

Fig.2-8 K_H-sliced convolution calculations

對於每一 slice，core 會執行：

一、載入 activation slice 到 L1。

二、載入對應 weight slice。

三、執行 GEMM： $Out_{partial} = A_{slice} \times W_{slice}$

四、將 partial output 累加至 output accumulation buffer，直到所有 kernel height 的 slice 完成後，即可得到完整輸出：

$$Out = \sum_{i=1}^{K_H} Out_{partial}(i)$$

2-5. Optimization implementation

此處介紹 Reader 程式負責讀取資料的部份，其主要工作是在 conv2D 計算開始前，根據不同的 sharded block，以及 host 端預先記下的 LIST(LIST[0]表示該處理 activat-ion 範圍有多少個不同的 W 片段將被存取，LIST[1:]則包含這些 W 片段在 DRAM 中的地址上下界)，使用 LIST[idx]以 on-the-fly 形式，依 stride 與 dilation，針對每個 output pix el 需要讀取的水平範圍 (w1, w2)進行讀取，拼出對應的 activation window，形成後端矩陣乘法所需的 row-major activation window。

Pseudo Code：

Original	*With K_h slicing*
1: procedure READER **2: for h = 0 to H_blocks-1 do** **3: n <- LIST[h][0]** **4: idx <- 1** **5: for t = 1 to n do** **6: (w1, w2) <- LIST[h][idx];** **7: idx <- idx + 1** **8: for w = w1~w2 step stride_w do** **9: READ-WINDOW-FROM-DRAM** **10: end for** **11: end for** **12: end for** **13: end procedure**	**1: procedure READER_SLICED** **2: start_idx <- 0** **3: for h = 0 to H_blocks-1 do** **4: offset <- ACT.base(h)** **5: for outer = 0 to KH_slices-1 do** **6: n <- LIST[h][outer][start_idx]** **7: idx <- 1** **8: for t = 1 to n do** **9: (w1, w2)<-LIST[h][outer][idx];** **10: idx <- idx + 1** **11: for w = w1~w2 step stride_w do** **12: READ-SLICED-WINDOW-FROM-DRAM** **13: idx <- idx + 1** **14: end for** **15: end for** **16: start_idx <- idx** **17: offset <- offset + row_stride** **18: end for** **19: end for** **20: end procedure**

Original 版本在處理每個 output pixel 時，會一次讀入整個 activation matrix（包含所有 kernel height），因此 inner dimension 會完整展開至 L1。這種方式簡單且資料連續，但需要較大的 L1 buffer 才能一次容納整個 window。

Kh-Sliced 版本則將 activation matrix 的高度方向切成多個 slice，每次只讀入其中一行 k ernel height。程式中會多出一層 outer 迴圈來逐行處理，並透過 start_idx 與 offset 追蹤每個 slice 的 reader index 與 DRAM offset。這種切分方式能顯著減少單次需要的 L1 空間，避免 block-sharded 設定下出現 L1overflow，代價是需要反覆呼叫 Reader 來組合完整的 window，會略為提升 kernel time。

2-6. Experimental method

A. Test Model

- Model: Stable Diffusion 1.4
- Resolution: 512×512
- Steps: 50
- Scheduler: PNDMScheduler

B. Performance Measurement

為客觀評估本研究之優化對 Stable Diffusion 推論效能的影響，本專題以 generate_time 作為主要指標，量測從模型接收 prompt 到輸出影像的完整推論時間。為降低隨機波動，每組測試在相同條件下執行五次，並取平均作為最終結果。

C. Quality Verification

為確認最佳化不影響生成品質，我們比較優化前後的輸出影像。實驗採用相同 prompt 與 random seed，使生成流程理論上保持一致。完成後逐一比對兩版本影像的內容、細節、紋理與整體結構，確認品質無差異。

3. Experimental Results

Input prompt :“oil painting frame of Breathtaking mountain range with a clear river running through it, surrounded by tall trees and misty clouds, serene, peaceful, mountain landscape, high detail.”

Table 1
Generate time for two types of BS conv

Test case	BS conv (full inner dim)	BS conv (inner dim slice)
1st iteration	5.960s	5.577s
2nd iteration	5.960s	5.579s
3rd iteration	5.963s	5.578s
4th iteration	5.959s	5.578s
5th iteration	5.961s	5.577s
average	5.9606s	5.5778s

Speedup = 1.07x, Improvement = 6.4%

Fig. 3-1-1
Picture generated by full inner dim BS conv

Fig. 3-1-2
Picture generated by slice inner dim BS conv

透過五次重複實驗的推論時間可觀察到，原始版本的平均生成時間為 **5.9606 秒**，而優化後的平均生成時間為 **5.5778 秒**。相比基準版本，slicing 共縮短約 **0.38 秒**推論時間，達到**約 1.07 倍速度、約 6.4% 效能提升**。結果顯示 activation slicing 能有效降低 Block-Sharded Conv2D 在 L1 SRAM 的記憶體壓力，並減少 DRAM 資料讀取次數，同時改善計算核心間的 pipeline overlap，使閒置時間下降、整體吞吐量提升。

在品質方面，我們比較了優化前後的輸出圖像。圖 2 與圖 3 中，兩種方法在相同 prompt 與 seed 下生成的圖片於結構、色調與細節上完全一致，未觀察到品質退化或推論誤差。這證實 slicing 僅調整 Tensor 的切片策略與記憶體排程，並未影響模型參數或數值精度。

整體而言，實驗結果表明 activation inner-dimension slicing 能在**不影響輸出品質的情況下有效提升 BS Conv2D 的運算效率**，並驗證縮減 L1 circular buffer 大小有助於提升 pipeline utilization。

4. Conclusion

本專題以 Stable Diffusion 1.4 為平台，針對 Tenstorrent Wormhole N150s 加速器的 Block-Sharded Convolution（BS Conv2D）運算流程進行分析與效能最佳化。透過對 Tensix Core dataflow 的剖析，我們確認原始 BS Conv2D 是圖片生成的主要瓶頸。在研究過程中，我們注意到 PR #25805 指出的問題：**受限於 activation matrix、weight matrix 的尺寸與 L1 SRAM 容量，使 DMA 與 compute core 的 pipeline overlap 無法充分發揮**，導致 compute idle、DMA stall 與 L1 暫存區競爭，使 BS Conv2D 成為 UNet 的主要延遲來源並影響整體推論時間。

為改善此問題，本專題採用 PR #25805 的 **activation slicing-by-kernel-height** 方法，將每個 block 的 activation matrix 依 kernel 高度切片，以降低單次運算的 L1 SRAM 需求。此策略縮小 activation circular buffer，釋放額外 L1 空間，使 weight 與 activation 得以更有效地 double-buffering，並提升 unpack、math、pack 三個 compute kernels 與資料搬移核心間的 pipeline 利用率。此方法成功緩解 L1 記憶體壓力並提升 compute 與 DMA 的重疊程度，使核心運算更穩定且高效。

實驗結果顯示，在相同 prompt、模型與硬體環境下，採用 slicing 後，Stable Diffusion 的平均生成時間由 **5.9606 秒**降至 **5.5778 秒**，效能提升 **6.4%**，推論速度由 **1.00×** 增至 **1.07×**。同時，我們未觀察到輸出影像品質差異，顯示此最佳化僅調整資料排程與記憶體管理，不影響模型數值或生成內容。

綜觀本研究，主要貢獻包括：（1）解析 BS Conv2D 在 Wormhole N150s 上的 per-core 資料流與 kernel pipeline 行為；（2）識別 activation block 對 L1 SRAM 的結構性瓶頸；（3）導入並驗證 activation slicing 的效能改善效果。本研究證實，透過適當的 block 切分與記憶體排程，即可在不修改模型的情況下獲得顯著推論提升。

總結而言，本專題展示軟硬體協同最佳化於專用加速器的重要性，證明依據 Tensix Core 架構特性進行記憶體切片與運算排程，能在不犧牲品質下帶來具體效能改善，並為後續開發提供明確方向。

5. Reference

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin,
"**Attention Is All You Need**," *Advances in Neural Information Processing Systems (NeurIPS)*, Long Beach, USA, Dec. 2017, pp. 5998–6008.

[2] O. Ronneberger, P. Fischer, and T. Brox,
"**U-Net: Convolutional Networks for Biomedical Image Segmentation**," *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Springer, Oct. 2015, pp. 234–241.

[3] J. Ho, A. Jain, and P. Abbeel,
"**Denoising Diffusion Probabilistic Models**," *Advances in Neural Information Processing Systems (NeurIPS)*, Virtual Conference, Dec. 2020, pp. 6840–6851.

[4] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer,
"**High-Resolution Image Synthesis with Latent Diffusion Models (Stable Diffusion)**,"
IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, USA, June 2022, pp. 10684–10695.

6. Reflection

本專題以提升 Stable Diffusion 在 Tenstorrent Wormhole N150s 上的推論效能為核心目標。從一開始的問題定義，到最終的優化驗證，我們逐步建立起分析生成式模型與異質加速平台之間互動的方法，並累積了軟硬體整合上的實作經驗。

首先，我們確立研究問題：Stable Diffusion 的 UNet 含有大量卷積與記憶體交換，在非 GPU 架構上容易受限於頻寬與資料流，尤其是 Block-Sharded Conv2D 更是延遲主要來源。因此，我們希望理解 N150s 的資料流特性，並找出能改善整體吞吐量的關鍵方向。

在架構理解階段，我們深入研究 N150s 的硬體組成：Tensix Core、DRAM、L1 SRAM、NoC 與各類 kernel 的角色。這使我們得以從硬體角度觀察模型執行行為，理解 activation、weight 如何在核心間流動，以及 reader、compute、writer kernels 如何形成完整 pipeline。同時，我們也補足生成式模型與機器學習基礎，包括卷積、Transformer、Attention、Diffusion 流程等，以便從模型運算本質出發分析瓶頸。

進入 code tracing 後，我們追蹤 TT-Metal 中 Stable Diffusion 的實作脈絡，逐層理解 kernel 如何被排程、資料何時進出 L1、程式碼如何對應到實際硬體運作。接著，我們測試多個 TT-Metal 版本，比對相同模型設定在不同更新下的效能落差，以提取哪些改動與推論速度最相關，並據此縮小優化方向。

經過整合分析後，我們確認最主要的瓶頸來自 activation block 在 L1 SRAM 佔用過大，使 DMA 與 compute core 的 pipeline overlap 無法完全發揮。基於此，我們選擇 PR #25805 提出的 inner-dimension slicing 方案，以 kernel height 為單位對 activation 進行切片。此做法有效縮小 circular buffer，大幅降低 L1 壓力，使 double-buffering 得以更穩定運作，也改善了 unpack、math、pack kernels 與資料傳輸間的整體配合程度。

在效能驗證階段，我們使用相同 prompt 與設定進行多次生成測試，並比較優化前後的 latency。結果顯示 slicing 成功縮短推論時間，同時保持影像輸出品質完全一致，證明此策略僅調整資料排程，不影響模型精度。

專題過程中，指導教授給予我們重要的技術與方法論指導。他提醒我們不能只描述現象，而應深入探討背後的原因；在推導上也要求更嚴謹，例如：不同分片策略的適用條件、為何特定配置無法選用 height-sharded 等。此外，教授也協助我們改善簡報與報告結構，讓分析邏輯更連貫、論述更具說服力。

回顧整個專題，我們最大的收穫在於建立跨硬體、軟體與模型三者之間的整合理解能力。我們學會如何分析加速器的資料流、如何閱讀大型 AI 系統的程式碼、如何找出真正的瓶頸並以實驗驗證假設。這些經驗讓我們對 AI 加速器上的模型部署有更清晰的認識，也為未來進行更深入的軟硬體協同最佳化奠定基礎。