國立清華大學 電機工程學系

實作專題研究成果摘要

Applying Block Floating Point (BFP) on the Attention Layer of the SD3.0 to Achieve Hardware Acceleration

將區塊浮點數應用於 SD3.0之注意力層以達到硬體加速

專題領域：系統領域

組　　別：B616

指導教授：黃朝宗

組員姓名：張有為、葉濬誠、蔡宜勳

研究期間：2025年02月01日至2025年11月21日止，共10個月

摘要

近年來，有越來越多影像的生成式模型發展出來，這類模型可以接收文字而生成與文字相符的影像。然而，即便模型的架構已經非常完善，這樣的大模型往往有個常見的問題，就是需要很大量的浮點數運算以及龐大的記憶體存取，生成一個影像就需要很大的運算資源，導致其在硬體實作與資源受限的環境下受到挑戰。因此，如何有效的降低運算成本與記憶體資源的需求，同時維持影像的品質，成為一個很重要的研究方向。而本專題是針對已經被訓練好的擴散模型 Stable Diffusion 3.0進行研究，嘗試以較低精度的數據儲存格式進行運算來達到加速，並維持影像的生成品質。

我們主要聚焦在區塊浮點數(Block Floating Point, BFP)的量化方法上，並將其運用在 Stable Diffusion 3.0模型中，以驗證此量化方法在生成式模型上的可行性以及效能。由於模型架構十分龐大，其中的輸入張量(tensor)值以及權重(weight)的數量非常多，於是本專題試著從減低資料通過 Attention layer 中的 QKV 卷積層(QKVconvolution layer)的計算量著手。做法是藉由改變數位運算格式，從原本精度高但硬體實作成本昂貴的 FP16改為 BFP，犧牲部分數值的精度換取運算上的速度以及面積的改善。

在研究過程中，我們首先針對 Stable Diffusion 3.0模型與 BFP 量化的原理進行分析與模擬，驗證其在軟體層面的可行性，並以不同的數據儲存格式操作在此模型上，並將生成的影像以 LPIPS、SSIM 兩種評估方法與原精度 FP16所生成的影像做比較。

在完成 BFP 在軟體層面上的實驗後，我們進一步去設計對應的硬體架構，將我們在 Stable Diffusion 3.0中針對 Attention layer 中的 QKVconvolution layer 的 BFP 卷積計算實作到硬體上，並用與軟體同樣的方式，更改 BFP 量化的 exponent 與 mantissa 的位元數，來對相對應的硬體進行合成，並記錄合成後的面積和 cycle time。

最後，我們將軟體與硬體的結果相互整合，發現在(sign, exponent, mantissa)位元數分配為(1, 4, 3)時，可以在維持影像生成品質的同時，最有效降低硬體使用面積與 cycle time，面積從原精度的37557 um^2，降到20428 um^2，而 cycle time 則是從5.3 ns，降到4.5 ns，達到硬體加速並兼顧影像品質的目標。

1. 研究背景與動機

我們的專題旨在透過區塊浮點數(Block Floating Point, BFP)的量化方法，將其運用在擴散模型 Stable Diffusion 3.0中，以降低生成式模型大量運算與記憶體儲存的成本。動機源自近年來，有越來越多影像的生成式模型發展出來，這類模型可以接收文字而生成與文字相符的影像。然而，即便模型的架構已經非常完善，這樣的大模型往往需要很大量的浮點數(floating point)運算以及記憶體存取，生成一個影像就需要很大的運算資源與功率，導致其在硬體實作與資源受限的環境下受到挑戰。因此，如何有效的降低運算成本與記憶體資源的需求，同時維持影像的品質，成為一個我們專題的研究方向。而本專題是針對已經被訓練好的擴散生成模型 Stable Diffusion 3.0進行研究，嘗試以較低精度的數據儲存格式進行運算來達到優化。

由於 Stable Diffusion 3.0模型架構十分複雜，其中的輸入張量(tensor)值以及權重(weight)的數量非常多，尤其又以模型中的 attention layer 中的 QKV 卷積層(QKV convolution layer)運算量特別龐大，光是這幾層就需要計算512*128*128*4約三千三百萬次浮點數的相乘與相加，於是本專題試著將卷積層的輸入與權重張量(tensor)，依著通道(channel)方向分塊，並將各個區塊的浮點數量化(提取出指數最大值)，而後改變浮點數格式。模型使用 IEEE 的 FP16格式，16代表的是位元數，我們透過減低與測試使用的位元數，從原本精度高但硬體實作成本昂貴的FP16改為 BFP，犧牲部分數值的精度換取運算上的速度的改善。

2. 文獻回顧

2-1. Diffusion Model

Stable Diffusion 3.0 模型是一種先進的文生圖（text-to-image）生成式擴散模型。在實際運作時，輸入文字會先經由 Text Encoder 轉換成語意向量（latent vector），並與雜訊影像一起送入核心的 MMDiT（Multi-Modal Diffusion Transformer）架構。MMDiT 整合了多模態注意力模組（Self-attention），在反覆的去雜訊（denoising）推論步驟中逐步調整影像特徵，最終重建出高品質、語意一致的影像。

然而，擴散模型如 Stable Diffusion 3.0 擁有極其複雜的模型架構和龐大的參數數量。模型的運作依賴於數十到上百個推論步驟，每一步都涉及大量的高精度浮點數矩陣運算與高維度特徵處理，例如模型中 Attention 的 QKV 卷積層運算量尤為龐大。這種對 FP16 或更高精度浮點數的依賴，導致其在生成單一影像時，會產生巨大的運算資源消耗（TFLOPs）與記憶體存取成本。於是我們的專題針對 Attention Layer 中的 QKV 卷積層進行優化。

2-2. Block Floating Point

BFP 是一種低精度的表示方式，將數個原先以 floating point 16格式儲存的數值列為同一區塊(Block)，對於同一區塊的浮點數會取出一個共享的指數(shared exponent)並以整數(int)形式儲存，再用各區塊的共同指數對對應區塊內的浮點數做量化(Quantization)，而後就可以用較低位元的格式儲存。缺點是低位元運算時會使得數值與原先值有誤差，而效益是面積較小、速度較快。BFP 的架構如下圖所示:

3. 實驗方法

3-1. BFP 設定

在我們的專題中，Block 的大小訂為16，且量化的方式為找出16數字中最大的指數作為共用指數，接著 Block 中數字的指數部分就減掉該共用指數，而共用指數則另存。原為 FP16的數字被量化後便會被我們調整為較少位元數的 BFP 格式[1]。

由於專題目的是要藉由降低精度以換取硬體功效的提升，我們嘗試的 BFP (block floating point)格式為不可更動的1個符號位元(sign bit)、2~5個指數位元(exponent bits)與1~10個尾數位元(mantissa bits)的所有組合(IEEE754_FP16為e5m10)[2]。

3-2. QKV Convolution Layer

我們選擇以 Diffusion model 中的 QKV projection layer 進行探討與實驗，完成我們的硬體加速器，這是在模組中含有大量成加法運算的部分，我們希望透過 BFP 的運算來做硬體層面加速。探討在不同 BFP 的 exponent 及 mantissa 位元數，對於影像品質、面積、運算時間等效能，會造成什麼樣的影響。

QKV projection 的運算如下，為一個1x1的卷積，輸入為三維的矩陣 A(activation)，大小為(Cin, H, W)，及二維的權重(weight)矩陣 W，大小為(Cin, Cout)，乘加完之後會加上對應的矩陣 B(bias)，大小為(1, Cout)，輸出的三維矩陣大小為(Cout, H, W)，矩陣各個元素以下列的數學式表達:

[1]為便於了解，報告以 BFP 格式泛指數字量化後，各種我們嘗試的格式，並在註腳處的下一段已將該格式所有可能列出。

[2] e5m10 代表 1 個 sign bit、5 個 exponent bits 和 10 個 mantissa bits 的浮點數

$$O_{chw} = \sum_{i=0}^{C_{in}-1} W_{c_{out}i} * A_{ihw} + B_c$$

3-3. 產生驗證資料

在研究過程中，我們首先對 SD3.0模型軟體改寫，設計出自己的 myconv，與 conv2d[3]有相同功能，藉此我們才能針對卷積層的輸入(activation)和權重(weight)進行操作。而後根據 BFP(block floating point)的原理加入量化與改變格式的功能，將輸入與權重都改以 BFP 的格式[4]，並以 BFP 格式進行卷積，並將不同數據儲存格式生成的影像以 LPIPS、SSIM[5]，兩種常見用於評估影像差異的數值指標，與原精度 FP16所生成的影像做比較。

過程中的難點是要讓 python 做到像是硬體的行為。原因是硬體在運算時都是以位元為單位做計算，軟體則是以浮點數值做計算。所以我們在 python 的卷積層中的乘法與加法，都是透過變換數值型態(type)與遮罩(mask)和位移(shift)等功能來模擬硬體加法器與乘法器的行為。

最終以我們更改過後的 SD3.0去生圖片，將軟體 QKV convolution layer 的輸入(activation)、權重(weight)作為硬體的輸入;軟體的輸出作為硬體輸出的正確答案，以此為基準去驗證硬體的正確性。

3-4. 硬體實作(Verilog)

原先在模型上的數值是以 IEEE754_FP16的方式儲存，而我們先將卷積層的輸入以及權重存於兩個不同 SRAM 記憶體，透過硬體設計讀取 SRAM 記憶體裡面的資料，進行 QKV Layer 數學運算的實現，將運算結果存下來後，與我們於 PYTHON 產生的驗證資料進行比對。

通過驗證程序後，我們調整 BFP 的 mantissa 與 exponent 的位元個數，採用 GSC Library 45nm、SS corner、operating condition 為 PVT_1P08V_125C 的製程

[3] Conv2d 為 python 內建函式庫 pytorch 中的用語，其功能為建置一個卷積層，並可以調整 channel, height, width, stride 等基本參數。

[4] 此格式及前面所說，我們會 40 種組合，並在實驗結果出來後提供最優的格式建議。

[5] LPIPS、SSIM 是評估兩張圖片差異的工具。LPIPS 是訓練過的模型，類似於人眼感知對兩張圖片的相似度;SSIM 則是基於結構、對比、亮度等資料做比較。

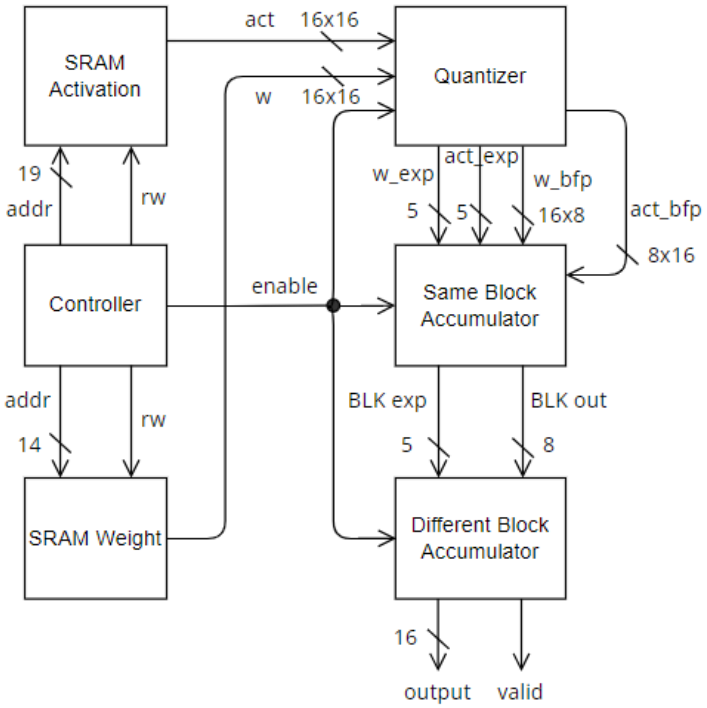
檔，進行電路合成，蒐集電路合成完的面積、時間(Cycle time)數據，與軟體的影像品質數據結合，探討何種 BFP 格式底下才是對於卷積層最好的電路設計。

除了 BFP 量化的架構之外，我們也進行傳統 FP16的卷積層硬體實作，透過相同的流程進行電路合成，預期在 BFP 的架構底下，電路效能能有效提升。

4. 硬體架構

4-1. Top Level Architecture

我們設計的電路架構如下，會有兩個 SRAM 存放權重(Weight)和輸入(Activation)，由 controller 計算卷積層需要的地址以及讀寫訊號，而獲得的資料會先經過量化(Quantization)輸出區塊指數及浮點數，接著會將同一個 block 的數據進行計算，再將不同 Block 的區塊進行累加，如下圖硬體架構(以 BFP_8BIT 為例)。



圖一、Block diagram: TOP(BFP8)

4-2. Quantization(量化)

以 BFP8(e3m4)為例，RTL 會從模型上下載的16個 FP16(e5m10)浮點數，取出指數部分(exponent)進行比較，找出最大的指數當成是區塊指數，再將這16個浮點數進行量化，所量化形成的 BFP8要與原先數值相同，兩者位元所代表的數值如下:

$$FP_{16} = (-1)^{S} * 2^{(\exp_5 - BIAS_5)} * (1.MANT_{10})$$
$$BFP_8 = (-1)^{S} * 2^{(BLK_{EXP} - BIAS_5)} * 2^{(BFP_{EXP} - BIAS_3)} * (1.MANT_4)$$

故量化 BFP8(e3m4):

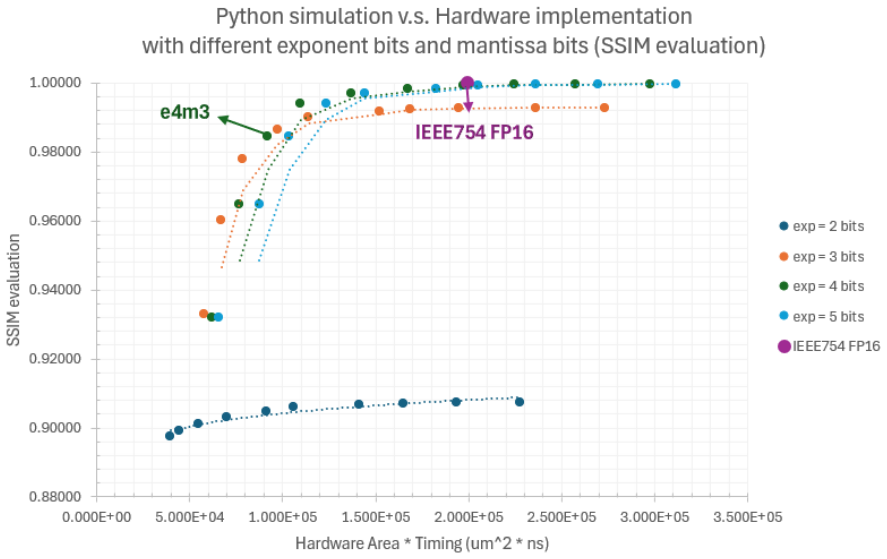
$$BFP_{EXP_3} = BLK_{EXP_5} - \mathrm{EXP}_5 + BIAS_3$$
$$BFP_{MANT4} = FP16_{MANT}[10:7]$$

以此類推，我們進行不同 exponent 及 mantissa 數的 BFP 量化，因為取用更少 BIT 表示指數，當同一個 BLOCK 的量化前 FP16浮點數與區塊指數差異太大，會將其量化後 BFP 指數設為0。

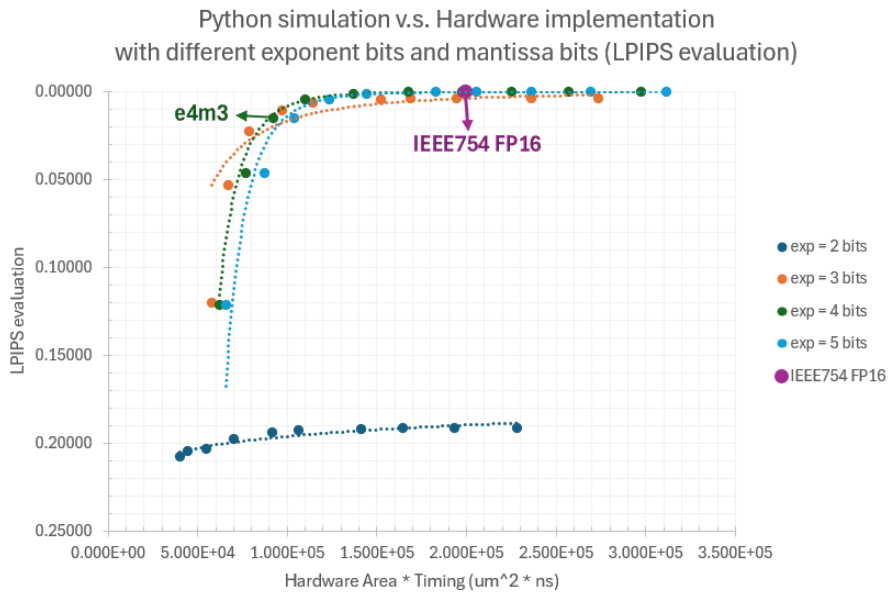
4-3. Calculation

對於我們的區塊浮點數，之後我們會先做同一個 BLOCK 的運算，以 BFP 的格式進行乘法以及加法累加，採用較少位元的 BFP 運算器。而後對於不同的 BLOCK，因為區塊指數的不同，先將低位元的 BFP output 轉為 FP16的格式，再以 FP16的加法器做累加，最後加上對應的常數(bias)，得到最後的結果。

5. 實驗結果分析



圖二、不同位元的硬體效能與 SSIM 分析圖

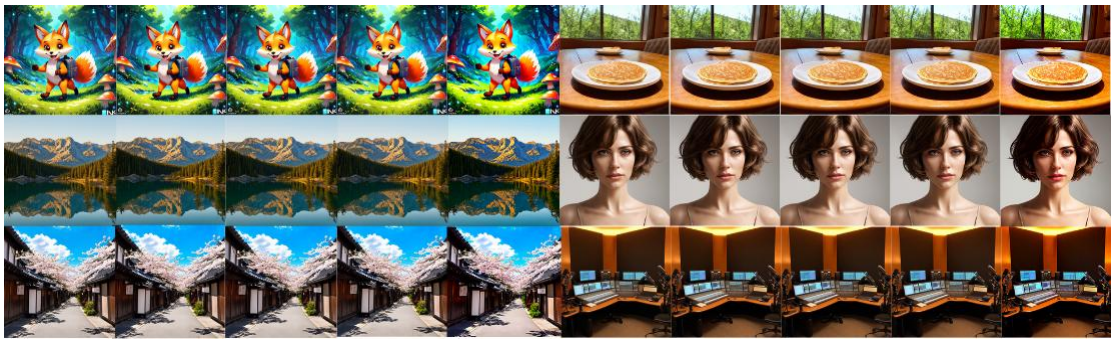


圖三、不同位元的硬體效能與 LPIPS 分析圖

以上是我們改變 BFP exponent 位元數(2-5)、mantissa 位元數(1-10)所合成的面積及 timing 數據，與所生成的影像品質的分析圖，橫軸是 area(um^2) * timing(ns)，越往左表示硬體效能越好;縱軸是 LPIPS 及 SSIM，越往上表示圖片品質越好。數據曲線同一顏色代表的是固定 exponent 數，而在同一曲線上的點由左至右是 mantissa 數由1bit 到10bit 的趨勢。

可以發現到 exponent 為3到5bit 時，當 mantissa 數越來越少，影像品質會造成影像品質流失太多，在兼顧影像品質以及最佳化電路效能情況，最左上角的數據點最好，也就是 BFP8、e4m3的格式:面積為20427(um^2)，cycle time 為4.5(ns)，與 IEEE754之 FP16對標有較好的表現，且 LPIPS=0.0153 (接近0越好)、 SSIM = 0.9846 (接近1越好) 表示影像品質與原圖相差無幾。

圖四、由左到右分別為使用 IEEE754_FP16, BFP_e4m3, BFP_e2m3格式做卷積所生成的圖



圖五、由左到右為 IEEE754 FP16, BFP e5m3, BFP e4m3, BFP e3m3, BFP e2m3

	AREA(um^2)	TIMING(ns)	PERFORMANCE
BFP8(e4m3)	20428	4.5	1.088E-05
IEEE754 FP16	37557	5.3	5.02E-06
Improvement	45.61%	15.09%	116.7%

表一、Final result (Performance = 1/Area*Timing)

透過以上圖表，我們可以確認使用 BFP e4m3(1 sign bit, 4 exponent bits, 3 mantissa bits)，不僅硬體上(包含量化等硬體)會有較好的功效，生成圖片的品質在肉眼或是評估工具上，亦明顯沒有顯著的降低，所以我們認為使用 BFP e4m3 格式是加速運算的理想方案。

此外本研究發現在 exponent=2bit 時，數據曲線出現了與前三者不一樣的趨勢，且影像品質過差，過度飽和、對比太強、畫質降低。因為在本次實驗的2bit exponent 並無規範在 IEEE754浮點數架構，是我們延伸 IEEE754規格所創造。在2bit 所能表達的 exponent 在標準化的數值表達，指數所能用的部分只有 2'b01、2'b10，因為所能表達的數值範圍太少，容易 underflow 與 overflow，因此我們不建議將 exponent 設為2bit。

6. 結論與未來展望

在本次研究中，我們目標是利用 Block floating point 的方式，在盡量不影響影像品質的情況底下，用硬體加速 Stable Diffusion 的 QKV Convolution Layer 中，大量乘法及加法的運算時間。在本次的實驗的硬體架構中，BFP 取8bit，格式中 exponent 取4bit、mantissa 取3bit 為架構中最好的設計。

此外，本研究嘗試延伸 IEEE754浮點數規格，將浮點數的 exponent 數減少為 2個 bit，結果是影像品質流失太大，無法將其效能加入比較。因此，如何以2bit exponent 為 BFP 架構進行實驗，是未來可延伸的方向。

7. 參考資料

[1] Esser, P., Kulal, S., Blattmann, A., Entezari, R., Muller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., Podell, D., Dockhorn, T., English, Z., Lacey, K., Goodwin, A., Marek, Y., & Rombach, R. (2024). Scaling Rectified Flow Transformers for High-Resolution Image Synthesis. *ArXiv, abs/2403.03206*.

[2] Hao-Jiun Tu (2024). Near-Lossless Microscaling (MX) Inference with Orientation-Aware Tiling for EfficientDet Backbone. http://etd.lib.nthu.edu.tw/detail/57387e9b972fa474d7ad4c551e7287e8/

8. 心得感想

在這次的實作專題中，我們針對 Stable Diffusion Model 3.0 中的 Attention layer，透過更改數值儲存格式，將原本運算的 FP16格式量化為 BFP 格式，成功達到降低硬體使用面積以及縮短運算延遲，並保持影像生成品質的目標。在專題初期，由於我們對於 Diffusion Model 架構、Attention 機制以及 BFP 量化完全不熟悉，因此我們花了大量的時間閱讀論文、研究相關背景知識，並逐步釐清整體系統的運作流程，才確立了可行的實作方向。

在實作階段，我們遇到最大的瓶頸是軟硬體行為的一致性驗證，而驗證的重要性在於硬體需要與軟體行為一致，才能確保圖片品質的測量具有意義。我們一開始在軟體端直接使用 Python 內建的浮點乘法與加法，而在硬體端則使用 DesignWare 的浮點運算模組。然而，我們發現軟硬體的行為不太一樣，導致兩者輸出誤差非常明顯，且 DesignWare 無法支援部分 BFP 量化設定，例如 exponent 僅2bit 與 mantissa 只有1bit 的情況，因此，我們需要在軟硬體開發符合需求的運算模組。

面對這些問題，我們重新在軟硬體兩端設計 BFP 乘法器與加法器，包含細部的 exponent 對齊、mantissa 正規化、進位與截斷邏輯等。經過多次模擬與比對，我們最終成功將軟硬體模擬輸出誤差控制在合理範圍以內，並使整個硬體架構能夠支援所有的 exponent / mantissa 位元組合。

回顧整個專題，我們從初期對模型與量化技術的陌生，到能夠獨立完成演算法分析、RTL 設計、軟硬體整合與驗證，每一步都累積了寶貴的經驗。這次專題實作讓我們更深刻體會到 AI 模型與硬體架構結合的複雜度，也學會在缺乏現成模組時，如何自行建構運算單元與驗證流程。整體而言，這是一段極具挑戰性但收穫滿滿的學習過程。