114學年度大學部專題海報展

國立清華大學 資訊工程學系
National Tsing Hua University Department of Computer Science

FPGA-Based Acceleration of TinyLLaVA Model Inference via High-Level Synthesis on Alveo U280

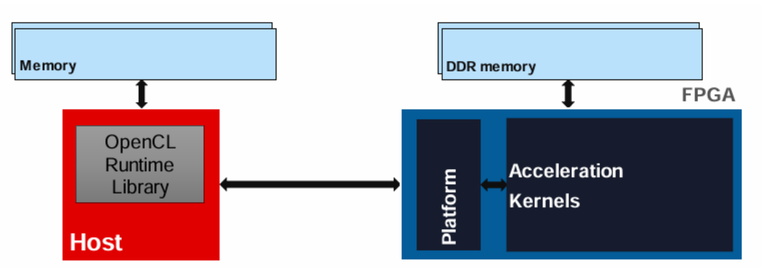
劉祐廷、林芷韻、黃子恩

摘要

本專題旨在探討如何**將多模態大型語言模型 TinyLLaVA-Phi-2-SigLIP-3.1B 的推論流程，透過 High-Level Synthesis (HLS) 技術部署至 AMD Alveo U280 FPGA 平台，以實現模型推論加速**。由於此模型運算涉及大量矩陣乘法與權重存取，本研究特別著重於資料路徑設計與記憶體存取策略的最佳化，以降低 I/O 瓶頸並提升資源利用率。在實作方面，我們完成了 SigLIP、Connector 與 Phi-2 三個模組的硬體合成，並驗證其可分別於 U280 上正確運行。針對記憶體部分，透過將權重合理分散至 HBM 不同 bank，搭配 AXI Burst 傳輸 與 tiling 計算策略，成功達成資料平行化與吞吐量提升。在 SigLIP 模型的 Attention 模組中，為避免除法與取餘數運算造成的邏輯資源浪費，我們改以 counter-based index 計算法 取代原始除餘數運算，大幅降低資源使用並提升效率。在 Phi-2 模組中，我們進一步利用權重矩陣旋轉技巧，降低 URAM 的緩衝區需求，並將原本的 float16 運算改為定點數 (fixed<40,16>)，使整體運算速度明顯提升。綜合結果顯示，本研究提出的設計方法能**有效縮短推論時間並提升 FPGA 資源使用效率，展現出以 HLS 為基礎進行多模態大型語言模型硬體加速的可行性與潛力**。

實作方式

我們先將 TinyLLaVA 拆解成 SigLIP (vision encoder)、Connector、Phi-2 (LLM decoder) 三個部分，接著將其轉換為 High-Level Synthesis 的 C++ code，並且將想要加速的 blocks 做成 Kernels，透過 Vitis 平台使用 XRT OpenCL 的方式，將 Kernels 部署到 Alevo U280 上，透過 Host (CPU端) 呼叫 Kernels 的方式，完成整個模型的推論，透過此方法我們可以**實現不同的 block (Kernels) 平行運算**。最後再進行效能分析，針對瓶頸的部分透過 #pragma 增加運算平行度，並且針對記憶體存取方式，對程式碼進一步優化。



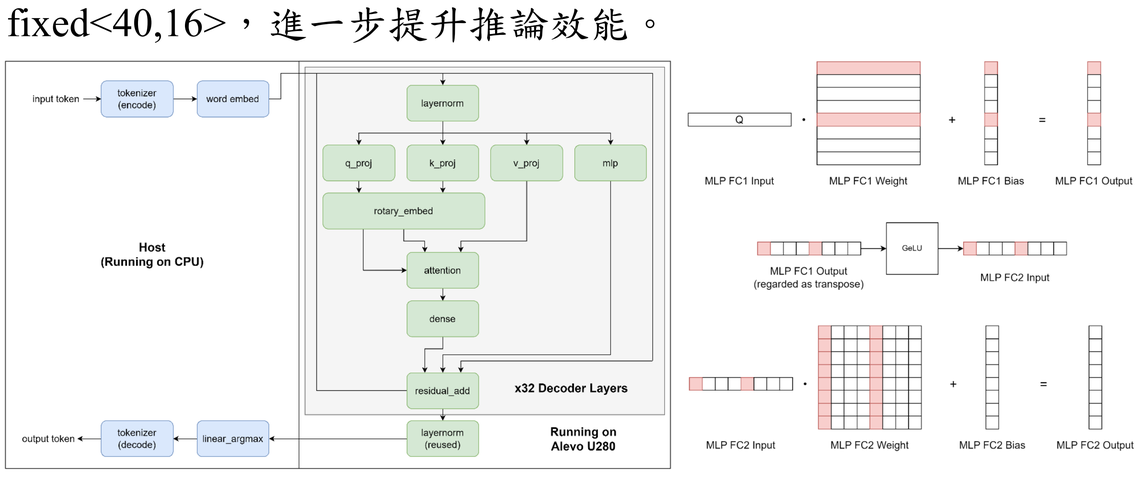
SigLIP (Vision Encoder) 加速方式

為了降低 Kernel 從 HBM 讀取資料的時間，我們將 Kernel 的 Input 與 Weight 暫存至 FPGA 的 BRAM 中，以縮短資料存取延遲並提升運算效率。然而，由於 BRAM 容量有限，對於運算量較大的 Kernel（例如 Connector），無法一次性容納所有資料，因此我們採用了 Tiling 技術。簡單來說，我們將 Input 與 Weight 拆分為多個較小的區塊（tile），每次僅將一個 tile 載入 BRAM 進行運算，運算完成後再將對應的輸出結果寫回 HBM，接著再載入下一個 tile 繼續計算。此方法能有效降低 HBM 的頻繁訪問次數，同時減少硬體資源占用，並有助於運算的平行化，最終達到提升整體推論速度與效率的目的。

另外，在 SigLIP 模型的 Attention 模組中，於計算 Query、Key 與 Value 的過程中，需要對 Linear layer的輸出進行 reshape 與 transpose。原始實作中是透過除法與取餘數運算來計算轉換後的 index，但這類運算在 FPGA 上會消耗大量邏輯資源，對硬體合成造成巨大的負擔。為了解決此問題，我們分析運算規則後，改以counter追蹤 head_num 與 offset 的位置，利用累加計數即可生成對應的index，從而完全避免除法與取餘數操作。此方法不僅大幅節省了 FPGA 資源，也進一步提升了 Attention 模組的運算效率，使整體系統在資源使用與效能之間取得更佳平衡。

Phi-2 (LLM Decoder) 加速方式

我們首先將 Phi-2 拆解成不同的 blocks，並將想要加速的 blocks 做成 kernels 部署到 Alevo U280 上。經過執行結果分析後發現 Phi-2 的運算瓶頸在 MLP 的兩個 Linear layer 讀取權重的過程上，因此針對 MLP 進行加速。針對記憶體存取的部分，我們採用 Burst Transfers 的方式，重新改寫 Kernel 的讀寫架構，大幅減少 MLP 從 HBM 讀取的時間。另外，我們也重新設計了 MLP 的計算流程，透過預先將 FC2 transpose 讀取的方式，FC1 與 FC2 中間的 GeLU 輸入輸出的 buffer 從原本需要兩個長度為 2560 (hidden size) 的陣列，縮減至只需要兩個長度為 8 的陣列，減少了 URAM 的使用量又不影響執行結果。最後，由於 FPGA 更擅長整數運算，因此將 data type 從原本 Phi-2 使用的 float16 改成 fixed<40,16>，進一步提升推論效能。



HBM 配置與記憶體存取優化

由於效能考量，我們想要能將運算所需的權重與中間計算產生的 buffer 全部放到 U280 的 HBM 裡，避免 Host 與 Kernel 間頻繁來回傳輸而造成延遲。考慮到 U280 的 HBM 只有 8 GB 且被拆分成 32 個 bank (256 MB per bank)，而我們的權重總空間就需要 6 GB，因此在存放權重的安排上我們也做了細緻的規劃，最終得以將所有的權重與計算過程的 buffer 放置在 HBM 中。

優化成果

SigLIP 的部分比起 C++ 運作在 i9-14900k 的版本約有五倍的性能提升，而 Phi-2 的效能也可以接近針對 CPU 有深度優化的 Pytorch 所實作出來的版本，展示了使用 FPGA 加速模型推論的可能性。

SigLIP Version	SigLIP Execution Time
Pytorch Version (Running on i9-14900k)	12.844 s
No Acceleration C++ (Running on i9-14900k)	2267.46 s
Original (Running on U280)	2290.55 s
Optimization (Running on U280)	385.92 s

Phi-2 Version	Phi-2 Execution Time
Pytorch Version (Running on i9-14900k)	2.206 s / token
No Acceleration C++ (Running on i9-14900k)	2029.44 s / token
Original (Running on U280)	9.093 s / token
Optimization (Running on U280)	3.986 s / token