High-Level Synthesis Based Acceleration of LLaMA2 Inference on FPGA

利用高階合成加速LLaMA2模型在FPGA上的推論設計

組別：62

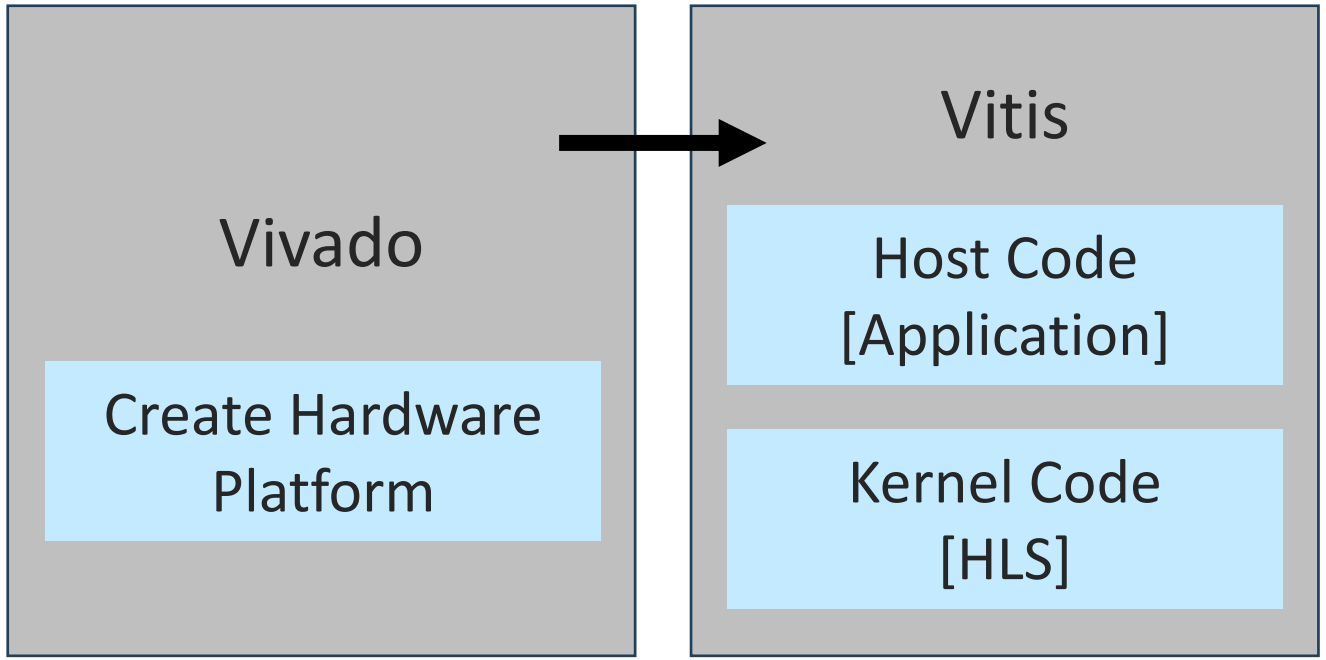
組員：趙堉安、金以凡

指導教授：陳添福 教授

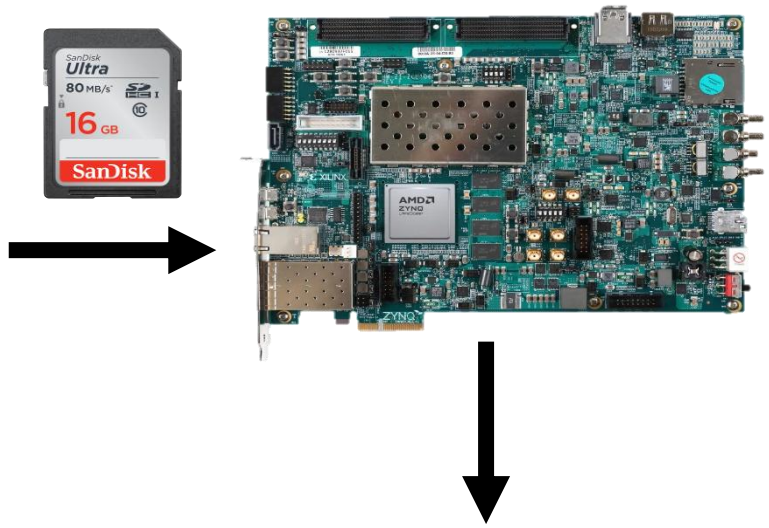
INTRODUCTION

- Background:
  - LLM rely heavily on matrix operations, making them ideal candidates for hardware acceleration
  - While cloud GPUs are powerful, they are not ideal for embedded devices or edge deployment. FPGA gives a better solution for accelerating inference
- Our main goal:
  - To accelerate inference speed by offload operations to FPGA kernels
- We use:
  - LLaMA2.c
  - AMD Zynq UltraScale+ MPSoC ZCU106 board
  - Xilinx Vitis 2024.2
  - High-Level Synthesis (HLS)

WORKFLOW



Build Hardware and Deployment



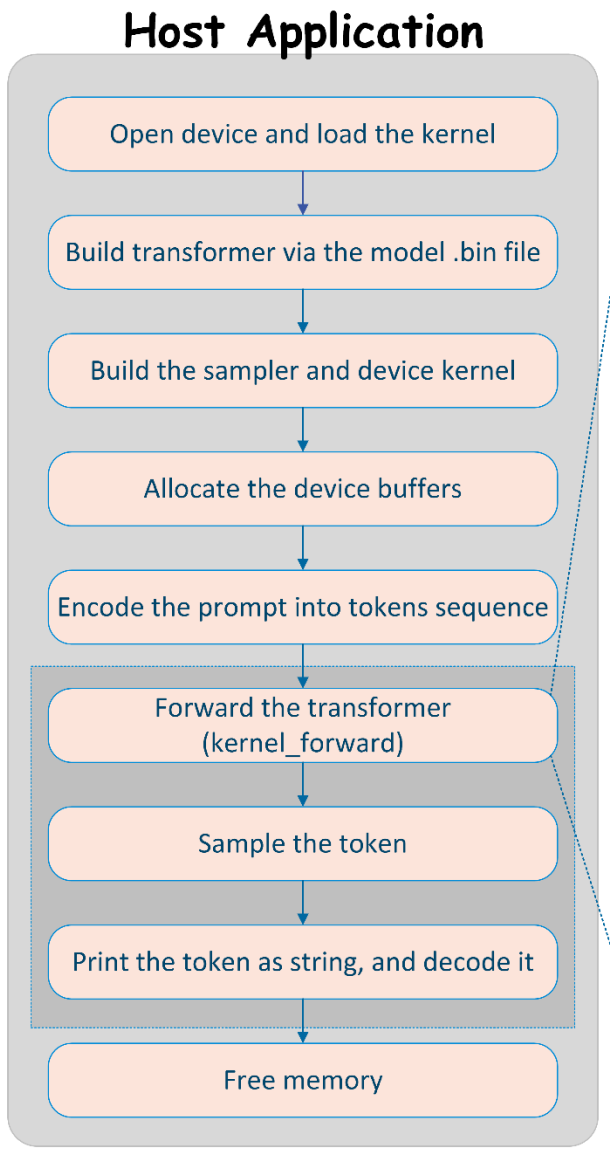
UART to connect monitor

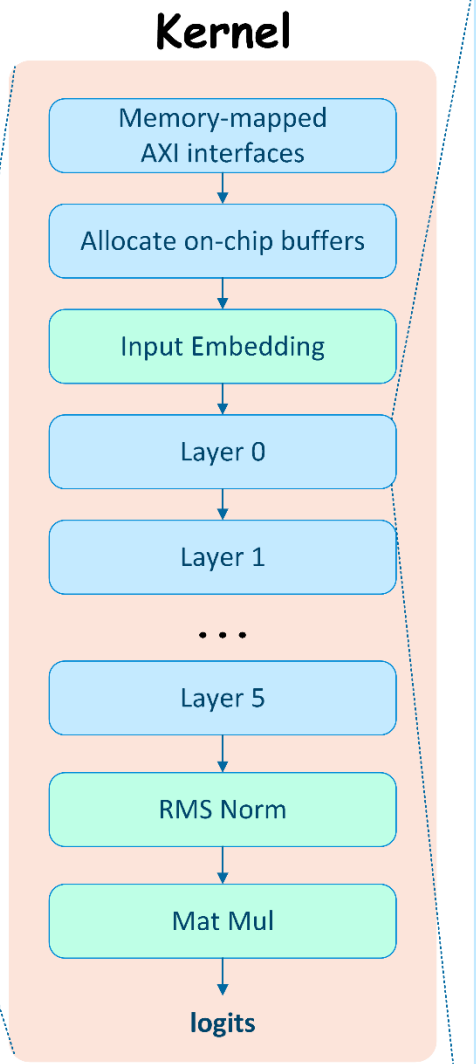
Host code load .xclbin to kernel

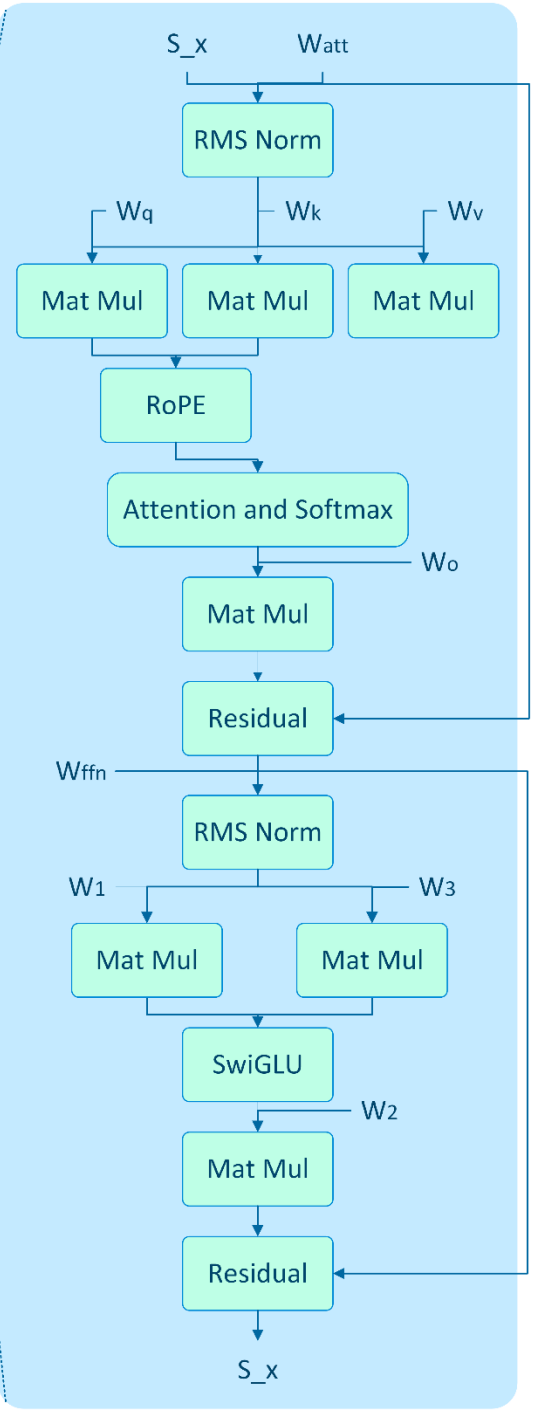
CONCEPTS

- LLaMA2 structure

- Offload six operations
  - RMS Normalization
  - Matrix Multiplication
  - Rotary Position Embedding (RoPE)
  - Self-Attention and SoftMax
  - Residual Connection
  - SwiGLU







METHODOLOGY

- We designed and optimized the following key components for FPGA to accelerate inference:
- RMS Normalization Block and Self Attention Block

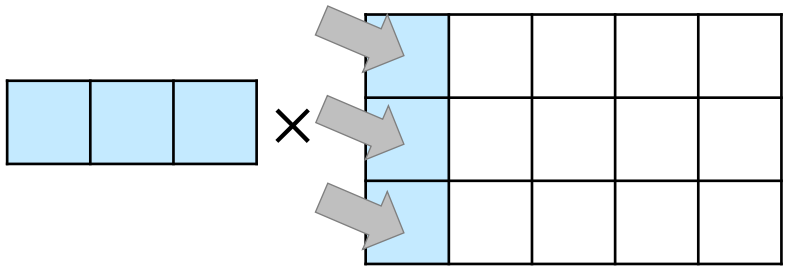
$$RMS(x) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} \qquad attention\ score = \frac{Q \times K^T}{\sqrt{d_k}}$$

  - Optimized approach: precompute the inverse or inverse square root values and store them as constants
  - Replace runtime division operations with faster multiplications
  - ***Respectively Speed Up 12.7x and 21.6x***
- Matrix Multiplication Block

METHODOLOGY

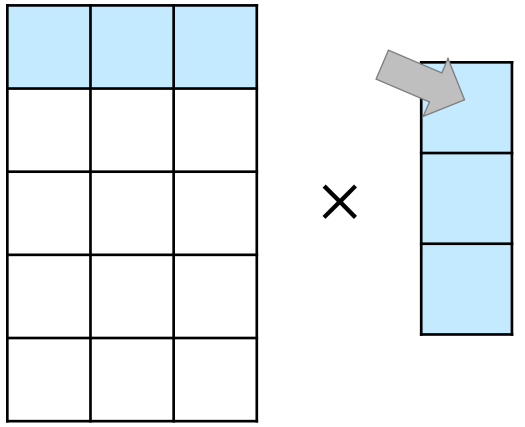
- Normalization Block and Self Attention Block
- Matrix Multiplication Block
  - For example: x size: n * 1, weight size: n * m

$$o = x^T \cdot w$$



*Need n*m non-continuous access*

$$o^T = w^T \cdot x$$



Need m non-continuous access

  - Mathematically equivalent but reduce hardware access times
  - ***Speed Up 10.6x***

METHODOLOGY

We applied several optimizations for blocks:

1. Memory Access Time Optimization by Preloading
   - Avoid multiple DRAM accesses by preloading data into BRAM
2. Array Partitioning and Loop Unrolling for Parallelism
   - Use pragma HLS ARRAY_PARTITION, HLS UNROLL
   - Combined loop unrolling and array partitioning to maximize parallel access and system performance
3. Pipeline for Throughput
   - Use pragma HLS PIPELINE
   - Achieved fully pipelined execution that processes one input per clock cycle

Achieved Total Speed Up: 13.4x

EXPERIMENTS

- Model we use: TinyStories 15M parameters
- Generate Stories: Once upon a time, there was a little boy named Timmy. Timmy was very brave and liked to climb trees. One day, Timmy saw a big wagon in the park. He wanted to climb on it, but he was scared …
- Result – Comparison of Inference Speed:

Hardware	Tokens	Time Spent (s)	Speed (toks/s)
APU	201	114.75	1.75
Logic Cells	225	25.83	**8.71**

-> Improved performance by around 5x

CONCLUSIONS

- FPGA Hardware Limitations
  - Limited DDR memory capacity restricts the deployment of larger models
  - Resource constraints (e.g., DSP, LUT) affect the degree of parallelism, limiting the extent to which computation can be unrolled or pipelined in hardware

- We successfully accelerated LLaMA2 inference speed by **5 times** using FPGA-based hardware optimizations
- Provided a Demonstration of the feasibility of Transformer inference on resource-constrained edge devices

Thanks for listening!