High-Level Synthesis Based Acceleration of Transformer-based Large Language Model Inference on FPGA

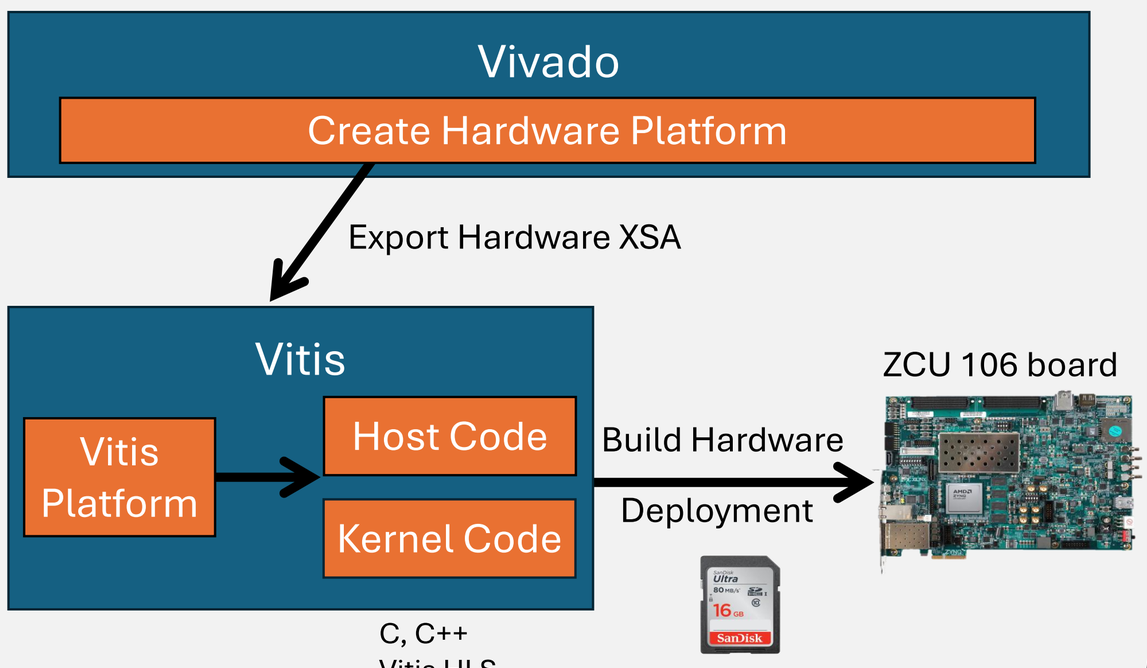
基於高階合成的大語言模型於FPGA推論加速設計

組別：62　組員：趙堉安、金以凡　指導教授：陳添福 教授

ABSTRACT

This project aims to accelerate inference of a lightweight language model trained on TinyStories by offloading **matrix multiplication** to an FPGA. Using the **llama2.c** implementation, the design is built with Xilinx Vivado and Vitis and deployed on the AMD Zynq™ UltraScale+™ MPSoC ZCU106 board. The FPGA kernel is implemented using High-Level Synthesis (HLS), which eliminates the need for manual RTL coding and significantly streamlines the hardware development workflow.

WORKFLOW



METHODOLOGY

Model Codebase: This project is based on llama2.c. The original implementation runs entirely on the CPU.

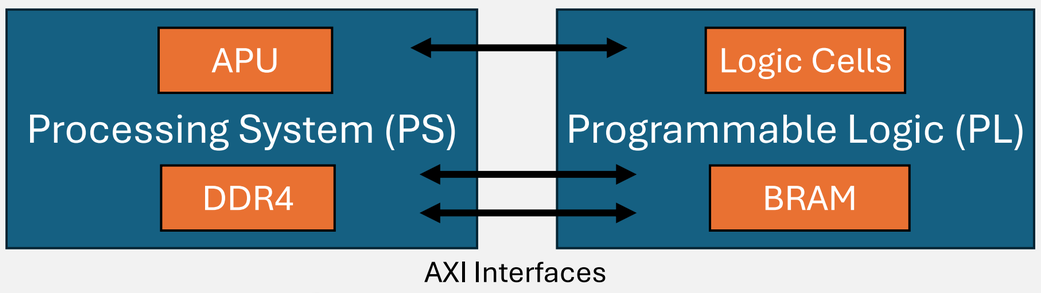
FPGA Acceleration Design: In the llama2 architecture, matrix multiplication is the most time-consuming operation during inference. To accelerate this bottleneck, we implemented matrix multiplication module on the FPGA. The kernel was developed using the OpenCL API.

Measurement and Evaluation: In the experiments, we measured performance based on the time taken to generate 256 tokens and the token generation speed (tokens per second). For the baseline, the APU-only implementation achieved 1.1 tokens per second.

EXPERIMENTS

A. The way of the host application and the HLS kernel transmit data:

- AXI interfaces to DDR4 memory connected to PS
- Data is copied from DDR4 to BRAM in PL

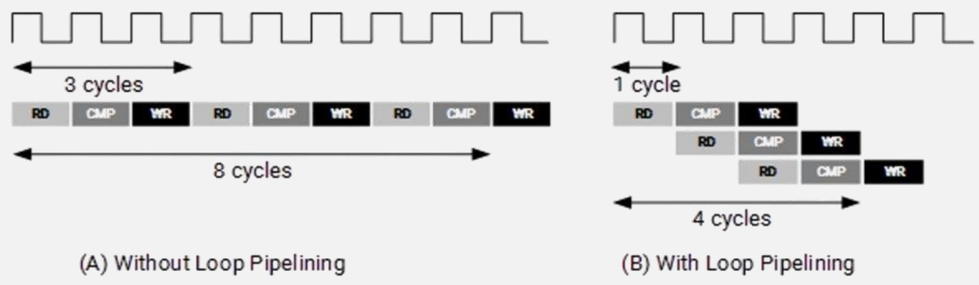


Experiment Result:

HARDWARE	SPEED (tokens/s)
AXI interfaces to DDR4	0.87 toks/s
Copy data to BRAM	**1.01 toks/s**

B. Apply two optimizations via HLS pragmas:

- Pipelining to increase throughput
- Loop unrolling to enhance parallelism



Experiment Result:

HARDWARE	SPEED (tokens/s)
without HLS optimizations	0.0533 toks/s
with HLS optimizations	**1.0186 toks/s**

CONCLUSION AND FUTURE WORK

Due to data movement overhead, kernel launch latency, and limited BRAM capacity (only 35 MB), the current FPGA implementation suffers from restricted hardware parallelism and therefore does not outperform the APU. We believe that by offloading more operations (such as RMSNorm and Softmax) into the kernel, the relative overhead of each kernel launch can be amortized, improving overall performance beyond that of the APU.