Convolution Co-processor for ZYNQ7000 processing system

Joey De Smet Sam Decorte

Faculty of Engineering Technology, KU Leuven - Bruges Campus
Spoorwegstraat 12, 8200 Bruges, Belgium
{joey.desmet, sam.decorte}@student.kuleuven.be

Abstract

Keywords— Co-Processor, SIMD

I. Introduction

Image processing is everywhere: take programs like Photoshop or Photopea for example. But for real-time applications like video processing or shaders, this becomes a lot more difficult.

If a CPU has to do all this sequentially, it would struggle to get real-time performance. Take this simple example: we have HD video (1920 pixels by 1080 pixels, 60 fps) we want to process. We would have to process $1920 \cdot 1080 \cdot 60 = 124416000$ pixels per second, giving us only 8,04 ns per pixel. Doing this sequentially on a CPU, means that the CPU has a very high workload, or might even be impossible.

Instead, you can offload this work to a co-processor, that will process an image in parallel. This makes the whole system faster, and the CPU has time to do other work.

In image processing, a kernel is a small matrix that can be used to add an effect to an image, by convoluting the kernel over the image. Different kernels achieve different tasks, like blurring, sharpening, embossing... The output of a pixel A does not have any effect on a different pixel B, meaning we can fully parallelize the process.

II. Implementation

A. High level overview

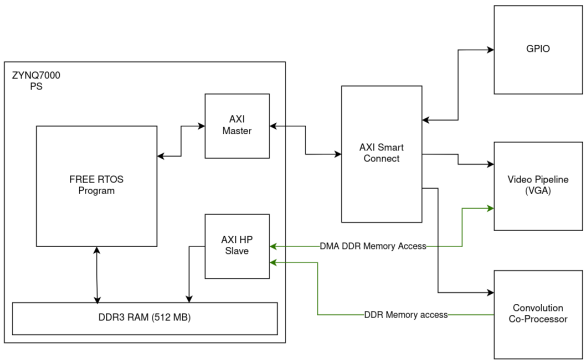

Fig. 1. Overview interconnect architecture

B. Convolution unit

The convolution unit is the heart of our system. It loads pixels into its buffer, and calculates the result of each pixel when the buffers are full enough. The system is shown in Figure 2.

Some kernels require fractional coefficients. This is implemented as a bit shift at the end of the calculation. This is not a catch-all solution, as only divisions by 2^n can be constructed, but it's much easier to implement and faster than using fixed-point or floating-point arithmetic.

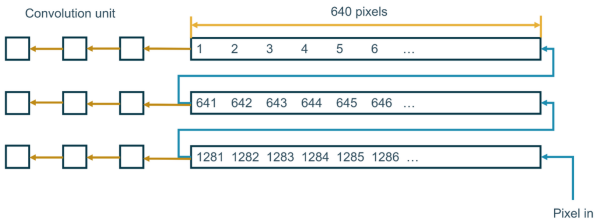

Fig. 2. Convolution unit schematic

The convolution unit contains three buffers, each 640 pixels in length. A pixel is shifted into the lower buffer each clock cycle. When the buffer is full, ejected bits are inserted into the middle buffer. When the middle buffer is full, the pixels are shifted into the upper buffer. This nicely arranges the pixels, allowing the module to ingest pixels from three image rows simultaneously.

To further increase speed, the convolution process is pipelined.

1) Stage 1: Calculate P_1c_1, P_2c_2, P_3c_3 in parallel
2) Stage 2: Calculate $P_1c_1 + P_2c_2$
3) Stage 3: Calculate $P_1c_1 + P_2c_2 + P_3c_3$
4) Stage 4: Calculate $(P_1c_1 + P_2c_2 + P_3c_3) >> s$

C. Effect of block size

To calculate the convolution of a block of size n by n pixels, we need the surrounding pixels as well, as shown in Figure 3. The efficiency, is given by $\eta = \frac{n^2}{(n+1)^2}$. For low block sizes, there are a lot of pixels that overlap between adjacent blocks, resulting in wasted re-fetching. However, larger block sizes mean less parallelism, resulting in a slower system.

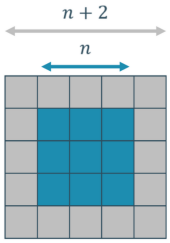

Fig. 3. Pixels to calculate (blue) and apron (grey)

III. Performance Analysis

In this section, we evaluate the performance of the proposed convolution co-processor. Metrics include processing throughput, latency, resource utilization, and energy efficiency. Comparisons are made with a reference CPU-only implementation on the ZYNQ7000 processing system.

A. Experimental Setup

The experiments were performed on a Digilent ZedBoard development board with the following specifications:

- Processing System: Dual-core ARM Cortex-A9, 667 MHz

- FPGA: XC7Z020 (Artix-7), 53k LUTs, 106k FFs, 4.9 Mb BRAM
- Clock frequency of co-processor: $100\,\mathrm{MHz}$
- Test images: resolution 640×480, 32-bit RGBA
- Convolution kernel: 3×3

B. Latency and Throughput

The latency T_{latency} of the co-processor is measured as the time between issuing a convolution request and receiving the processed data:

$$T_{\text{latency}} = T_{\text{transfer}} + T_{\text{compute}} + T_{\text{response}} \tag{1}$$

Throughput $R_{\text{throughput}}$ is calculated as:

$$R_{\text{throughput}} = \frac{\text{Number of pixels processed}}{T_{\text{latency}}} \tag{2}$$

TABLE I. Latency and throughput for processing new versus in-memory images

In memory	Latency [ms]	Throughput [MPix/s]
No	–	–
Yes	–	–

C. Resource Utilization

The FPGA resource usage of the convolution co-processor is summarized in Table II:

TABLE II. FPGA Resource Utilization

Resource	Used	Available
LUTs	–	–
Flip-Flops	–	–
BRAM [Kb]	–	–
DSP Slices	–	–

D. Comparison with CPU Implementation

For reference, a CPU-only implementation, as a FreeRTOS task with highest priority, was run on the ARM Cortex-A9 core. Table III summarizes the speed-up achieved:

TABLE III. Speed-Up of FPGA Co-Processor vs CPU

CPU Latency [ms]	FPGA Speed-Up
–	–

E. Energy Efficiency

Energy consumption was measured for the convolution co-processor using onboard power monitoring or external measurement tools. The energy efficiency η is defined as the number of pixels processed per joule of energy consumed:

$$\eta = \frac{\text{Number of pixels processed}}{E_{\text{total}}} \quad [\text{MPixels/J}] \tag{3}$$

where E_{total} is the total energy consumed during the convolution operation.

TABLE IV. Energy efficiency of the co-processor for CPU and FPGA

Platform	Energy [mJ]	Efficiency [MPix/J]
$FPGA$	–	–
CPU	–	–

IV. Conclusion

Add conclusion here

V. Future work

- Splitting the data into the different buffers to allow for more parallelism, is now managed by the processor. A hardware implementation could make it possible for data to be streamed in bigger burst which would decrease te delay for data transfer.
- Currently only 3×3 kernels are supported some minor changes could be done to expand this to a $n \times n$ kernel.

Acknowledgment

The autors used generative AI tools to assist with language refinement, LaTeX table template generation and grammar correction during the preparation of this paper.

References

[1] Digilent, *ZedBoard User's Guide*, 2014. Available: https://files.digilent.com/resources/programmable-logic/zedboard/ZedBoard_HW_UG_v2_2.pdf

[2] ARM, *AXI specification*, 2025. Available: https://developer.arm.com/documentation/ihi0022/latest/

[3] AMD, *Zynq 7000 SoC Technical Reference Manual*, 2023. Available: https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM/Register-ICCIDR-Details