1. Addition of peripherals on the CVA6 core using HLS

Advisor: Nicolas Derumigny

Prerequisite: Linux / 16 GB RAM / 50 GB available disk drive

Summary: The CVA6 is a low-power, open source CPU written in SystemVerilog. Using it in real-life requires some essential peripherals (DRAM controller, firmware, etc) that the Core-V APU implements under a very basic form. In the team, we have running version of this design for two FPGAs, the ZCU104 and the PYNQ-Z2. On the Core-V APU, memory and peripherals are accessed using a central addressable AXI bus. In this project, the goal is to use the HLS (High-level Synthesis) framework to write AXI-accessible devices using C/C++, implement a working design integrating them as well as the corresponding driver.

2. Customization of the CVA6 core

Advisor: Nicolas Derumigny

Prerequisite: Linux / 16 GB RAM / 50 GB available disk drive

Summary: The CVA6 is a low-power, open source CPU written in SystemVerilog. In the team, we use it to run a small hypervisor, Fork-Nox. The goal of this project is to identify, quantify and if possible implement microarchitectural characteristics that would decrease the overhead of Fork-nox compared to native execution. One lead is to study the influence of data / instruction cache sizes, branch predictor, frequency and TLB size. Another lead is to modify the granularity of virtual-to-hypervisor interception mechanisms, which requires deeper modification of the code base.

3. MARS Attack: Optimizing CPU to FPGA data transfers through coalescing, compression and data layout reordering

Advisor: Nicolas Derumigny, potentially Elizabeth Brunet

Prerequisite: Linux / 16 GB RAM / 50 GB available disk drive

Summary:: FPGAs are a class of chips that can be configured as compute accelerators. In this setup, the goal is to deport a part of a computation named tile on the FPGA, whose on-chip memory is limited. When memory bandwidth becomes the issue, using efficiently the limited memory interfaces is crucial. We have developed an optimization framework relying on compression, packing and data layout modification through an analysis called MARS to minimize latency, i.e. reduce the communication time between the host and the FPGA. In this context, you will either:

- leverage MARS to accelerate new benchmarks from the Polybench/C suite, by manipulating abstract program representations and transformations to

generate new modules.
- integrate existing MARS transfers modules to an end-to-end accelerator for benchmarks extracted from the Polybench/C benchmarking suite.