Computer Architecture
Final Project
Part -2

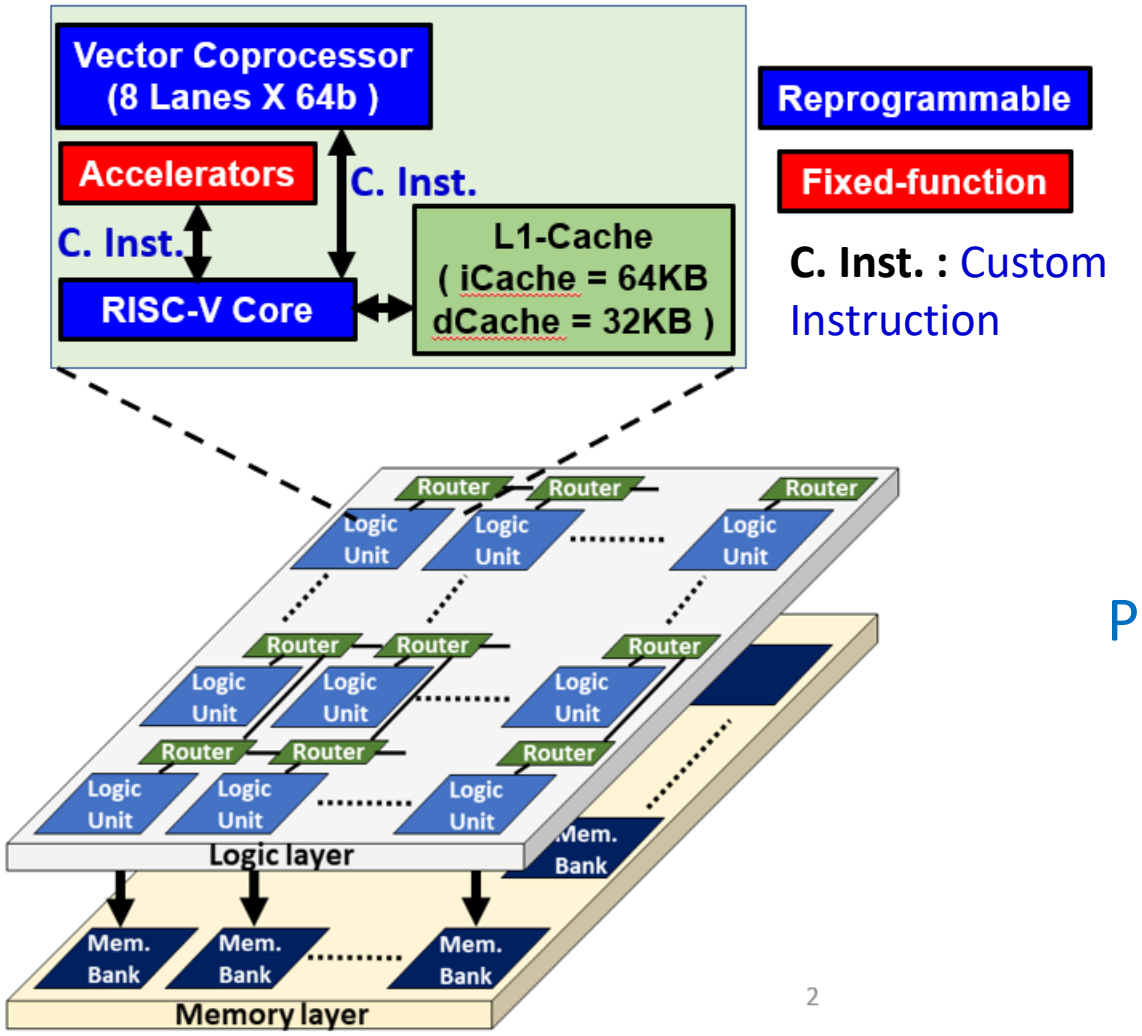
Enhance FP1 Design with Vector Coprocessors and Custom Accelerators

On Near-DRAM Platform

Problem Statement

- Objective: Improve RNA sequence Quantification performance on the Stacked DRAM (SEDRAM) platform
  1. Indexing: Hash-map generation (same as FP1)
  2. Quantification: Query Hash-map and generate final results (same as FP1)
- Methodology:
  - Improve Indexing and Quantification using SIMD (Vector Coprocessor) and Accelerator
    - Baseline: Your index.c and quantify.c FP1 implementation
    - Environment: New Docker image (amansinhaatnycu/ca-fp:v5)
    - Steps:
      1. Install the new Docker image and start container
      2. Make a copy of FP1 as FP2 in /home/ and implement index.c and quantify.c using accelerator (accelerator_reduce_sum instruction in sample.c) and vector coprocessor (vector_compare instruction in sample.c). Reuse the code structure from your FP1 for minimal effort and confusion.
         - You have your choice of functionality for the two instructions, which can be changed in /home/gem5/src/arch/riscv/isa/decoder.isa file. Rebuild Gem5 before simulation FP2.
    - Samples:
      - Accelerator (reduction sum operation of bytes within uint64_t): Lines 40-49 in sample.c
      - Vector Coprocessor (equality check of bytes in two unint64_t): Lines 58-67 in sample.c

Detailed Target Architecture



- **RISC-V Core:** In-Order @800 MHz
- **Vector Coprocessors:** 8 lanes X 8b, Controlled by RISC-V Custom Instruction (vector_compare)
- **Accelerator:** Controlled by RISC-V Custom Instruction (accelerator_reduce_sum)
- **DRAM Bank:** 256MB @4266 MBPS per pin

Programmed only RISC-V Cores using pre-existing instructions in FP-1

Custom Instructions for Vector Coprocessors & Accelerators to be implemented in FP-2

Simulation Setup & Evaluations

Simulation of Code to be Evaluated

1. Copy your CA-FP1 evaluation folder that you implemented and rename as CA-FP2
2. Change to the directory with RISC-V code (single Logic Unit + Vector Coprocessor + Accelerator + DRAM Bank) to be evaluated, implement your Indexing and Quantification algorithm
   - cd /home/CA-FP2
   - vi index.c ; vi quantify.c
3. Modify the definition of instructions “accelerator_reduce_sum” and “vector_compare” in the file “/home/gem5/src/arch/riscv/isa/decoder.isa”
4. Re-compile Gem5 simulator:
   - cd /home/gem5/ && scons build/RISCV/gem5.opt -j 12
5. Write your own Makefile by looking at /home/CA-FP2-Sample/Makefile. Compile Indexing and Quantification implementations accordingly.
6. Simulate Indexing and Quantification implementations, by looking at /home/CA-FP2-Sample/Makefile. Use FP-1 scheme to profile time.

Zip & Submit:

/home/CA-FP2

decoder.isa

DO NOT MODIFY Gem5 other than the specified instruction definitions

Custom Instructions for Accelerator and Vector Coprocessor

/home/gem5/src/arch/riscv/isa/decoder.isa

```
        //custom
0x16: decode FUNCT3 {
    format ROp {
        0x2: decode FUNCT7 {
            0x0: accelerator_reduce_sum({{
                Rd = 0;
                // do vector reduce sum here
                for(int i=0;i<8;++i)
                    Rd += (uint64_t)(((uint64_t)Rs1 >> (8 * i)) & 0xFF);
                for(int i=0;i<8;++i)
                    Rd += (uint64_t)(((uint64_t)Rs2 >> (8 * i)) & 0xFF);
            }});
        }
        0x4: decode FUNCT7 {
            0x30: vector_compare({{
                // do vector compare here
                Rd = 0;
                for(int i=0;i<8;++i) {
                    uint8_t byte_a = (Rs1 >> (i * 8)) & 0xFF;
                    uint8_t byte_b = (Rs2 >> (i * 8)) & 0xFF;
                    if (byte_a == byte_b) {
                        Rd |= ((uint64_t)0x01 << (i * 8));
                    }
                }
            }});
        }
    }
}
```

Accelerator

accelerator_reduce_sum: Free to use the two 64-bit operands, Rs1 and Rs2 to achieve any objective. E.g., Hashing. Latency = 1 cycle

Vector Coprocessor

vector_compare: Can implement only **SIMD** computation using the two 64-bit operands, Rs1 and Rs2 to achieve any objective. E.g., comparison of k-mers. Latency = 1 cycle

Change only the definitions of the two instructions in the decoder.isa, DON'T CHANGE ANYTHING ELSE

Samples for Custom Instructions

/home/CA-FP2-Sample/test.c

```
uint64_t accelerator_reduction_sum_result = 0;
asm volatile (
                "ld x10, %[a0]\t\n"
                "ld x11, %[a1]\t\n"
                "accelerator_reduce_sum %[s], x10, x11\t\n"
                : [s]"=r"(accelerator_reduction_sum_result)
                : [a0]"m"(inputData[i + 0])
                , [a1]"m"(inputData[i + 1])
                : "x10", "x11"
                );
```

```
uint64_t vector_comparison_result = 0;
asm volatile (
                "ld x10, %[a0]\t\n"
                "ld x11, %[a1]\t\n"
                "vector_compare %[s], x10, x11\t\n"
                : [s]"=r"(vector_comparison_result)
                : [a0]"m"(inputData[i + 0])
                , [a1]"m"(inputData[i + 1])
                : "x10", "x11"
                );
```

R-type Instructions. Use them as you need. Just make sure the final results are the same as FP1.

Evaluation Methodology & Scoring

- Total Score: 100
  1. Indexing speedup > 1.0 compared to FP1 (index.c): 30
  2. Quantification speedup > 1.0 compared to FP1 (quantify.c): 30
  3. Total Runtime (Ranking of Indexing + Quantification): 30
  4. Report (Details of proposed algorithms 1-2 & their performance results): 10