
mp_ooo Lab 4

How are you all doing?

- CP3 due Sunday @ 11:59pm!!!
- Performance Counters and Analysis
- Advanced Features
- Advanced Verification Tooling

CP3 Requirements

- Points lost if **any warnings** or **doesn't pass synthesis**
- Add support for control instructions
  - At minimum: flush everything when a mispredicted branch commits
  - Static-not-taken branch prediction
- Add support for memory instructions
  - At minimum: stall dispatch after a store is dispatched until it is committed
- Integrate with your mp_cache with competition memory (Instruction and Data)
- Should be able to run Coremark
  - If it does not run, **write your own testcode** to demonstrate the control (taken/not-taken) and memory (load/store) instructions
- Progress report + Roadmap

Why isn’t your processor fast?

Q: If your processor never stalls, what is your IPC?

A: 1

Q: What is your IPC right now?

What is slowing down your processor?

I have no idea.

But let’s find out, together!

Performance Counters

What are they?

- Count the number of times stuff happens

- What can we count?

    - Eg: branch mispredict/flush counter, memory stall counter, I-queue full counter, etc.

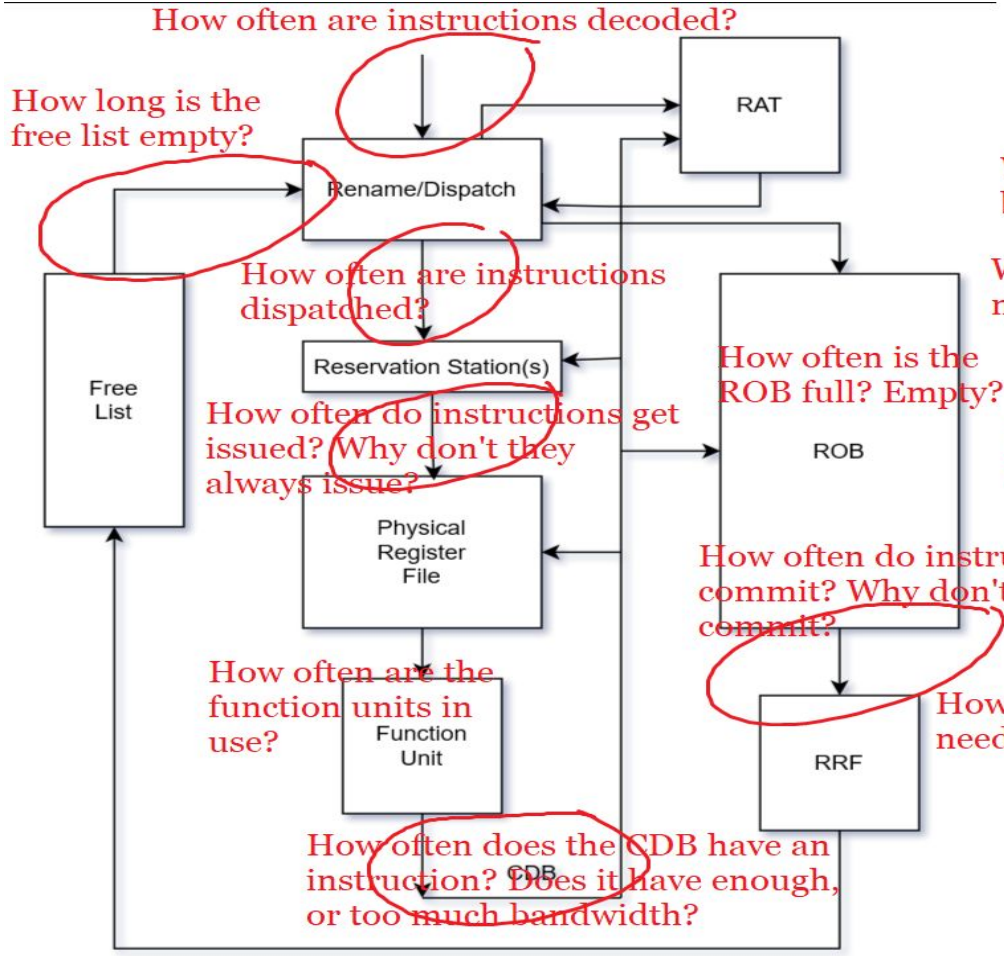
```
----- Stalls -----
if_stalls : 25690117
id_stalls : 9482
mm_stalls : 19621166
mul_stalls : 00
div_stalls : 00
----- Misc -----
control_misses : 150021
----- Ending Simulation -----
```

Example perf counters on an in-order pipelined processor

How to implement them?

```
final begin
   ...
end
```

- 2 main ways to add perf counters
  - In processor HDL
  - In testbench HVL
- HDL approach
  - Add counter variables in desired modules
  - Don’t reference them within other modules!!! (make sure synth can ignore them)
- HVL approach
  - Add counters within top_tb.sv or other testbenches
  - Use hierarchical references to access in-DUT signals and update counters
- Make sure to print out counters or write them to a file when sim is finished!!!
  - `final` – just like `initial`, but at the end


How often are instructions decoded?
How long is the free list empty?
RAT
Rename/Dispatch
How often are instructions dispatched?
Reservation Station(s)
Free List
How often do instructions get issued? Why don't they always issue?
Physical Register File
How often are the function units in use?
Function Unit
CDB
How often does the CDB have an instruction? Does it have enough, or too much bandwidth?
How often is the ROB full? Empty?
ROB
How often do instructions commit? Why don't they commit?
RRF
How often do you need to flush?
What is the AMAT of the L1 I cache? DCache?
What is the utilization of the banked memory port?
What is your average misprediction penalty?
What is your control instruction prediction accuracy?
There are so many things you can count!

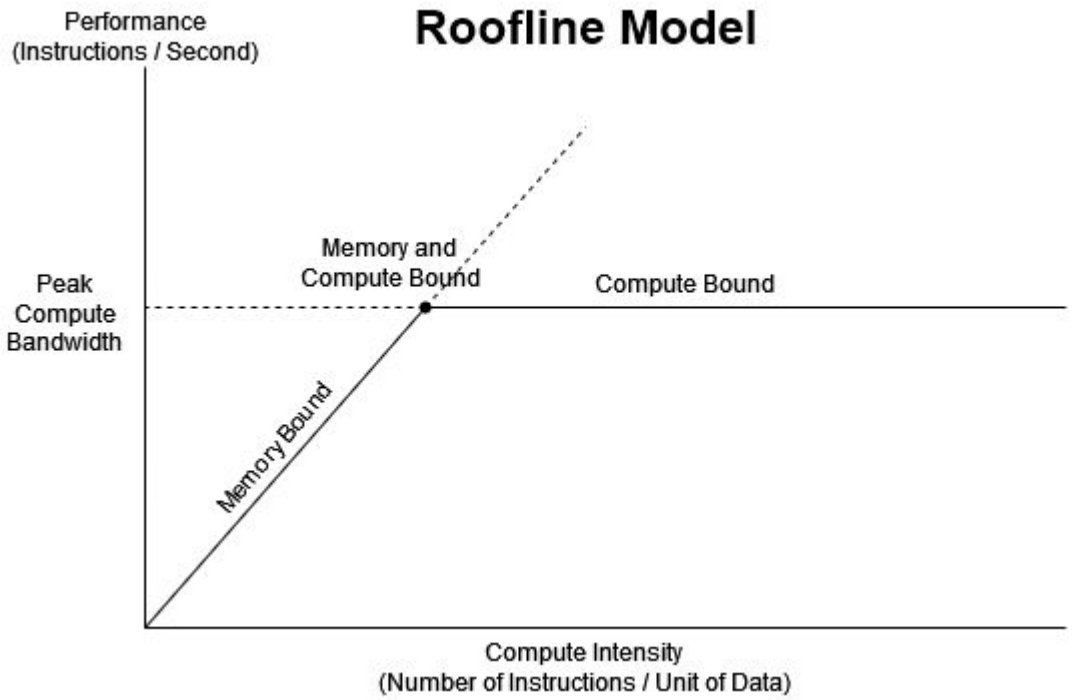
Analyzing Performance - Roofline Model

A program's peak performance is either bounded by:

- **Compute** (ALU instructions / second)
- **Memory** (bytes / second)

Programs have varying **Compute Intensity** (instructions per unit data)

Make sure you are optimizing the right parts!



Advanced Features

Advanced Features

- Need 20 points worth of features
  - Any more on top is extra credit!!!
- Don’t need them in competition submission if it degrades your score
  - You will still get points for demonstrating that the implementation works
- Complete list of features detailed in ADVANCED_FEATURES.md
  - If you want to propose/implement something not listed, please talk to us!
  - Will cover more feature specifics next week
- Competition deadline is 4/29!

Recommendations:

- 1 (b)ranch predictor
  - Something better than a saturating counter please :D
- Pipelined L1 instruction cache (read only)
  - mp_cache is too slow for instruction fetch
- Prefetcher (Next line/strided)
  - L2 cache makes this much easier as well
- Better multiplier (at least fewer cycles, ideally a tree multiplier)
  - The given one is **extremely** slow!
- Divider (Synopsys IP or your own implementations)
  - Would support full RISC-V M extension

Super cool features that we like:

Superscalar: Multi-instruction fetch, decode, dispatch, issue, writeback, (commit needs small patch)

- Separates the OoO from the pipelined processors

Non-Blocking Cache

- Banked memory supports multiple (16) outstanding requests

Memory Disambiguation/Speculation

- Make your memory operations actually out-of-order (to an extent)

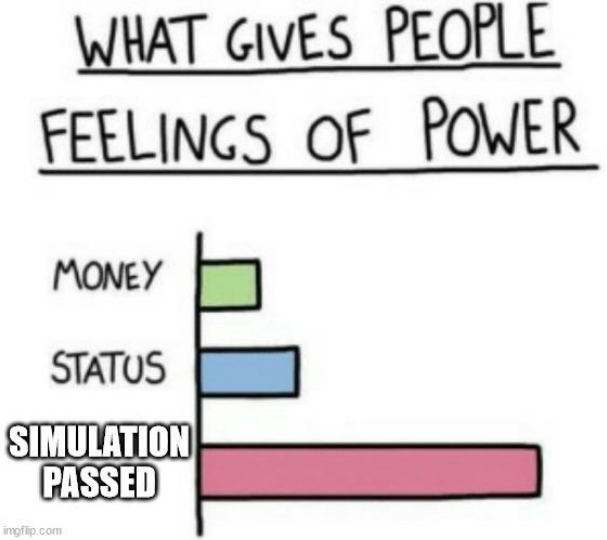
Early Branch Recovery

- Branch predictors can only do so much... and their accuracy increases with this

Ask your mentor/course staff if you have questions about specific features!

Alternate Verification Tooling

What is Verilator?



- Like VCS, converts your hardware to C++. But it's open source!
  - Can run on your local machines as a result, and is cross-platform
- Drawbacks: there's **a lot**.
  - Cycle-level simulation
  - *Only supports dual-state simulation*
  - Also pickier about the quality of your HDL (stricter linting)
  - Incomplete SV support compared to VCS
  - Harder to support threaded models
  - Can't access internal signals by default
  - Traces are way bigger compared to VCS (15GB for all of Coremark with struct extraction)
- Why bother using it at all?
  - ***Speed* - "all of CoreMark within seconds" type speed.**

Before

After

Verilator Testbenches

- Overall simulation logic is no longer managed by an SV initial block
- Need to write C++ logic to interface with the DUT, manage CLK, etc.
  - Can still write performance monitoring in SV, but be more careful
- State management in C++ -> powerful software models in a familiar language
  - Lack of bit-sliced access can sometimes make memory access challenging
- Many other features, so definitely look at the documentation on their website!

Ventilator

- We’re nice, so we made you some tooling
  - Does the same thing as top_tb for the most part, just faster
  - Be careful when dumping traces
  - Not currently supported on EWS (both good and bad?)
- Further instructions will all be in README.org (ventilator subdirectory)
- Memory model in CP3 patch is for old burst mem, new banked mem model + better superscalar should be out by now

Cocotb

Website: https://www.cocotb.org/

Documentation: https://docs.cocotb.org/en/stable/

“cocotb is an open source coroutine-based cosimulation testbench environment for verifying VHDL and SystemVerilog RTL using Python.”

This means we get to use Python for debugging! :D

It is also entirely free to use!

To install on EWS, you should use Python virtual environments.

https://www.freecodecamp.org/news/how-to-setup-virtual-environments-in-python/

Cocotb Example

```
module my_design(input logic clk);

  timeunit 1ns;
  timeprecision 1ns;

  logic my_signal_1;
  logic my_signal_2;

  assign my_signal_1 = 1'bx;
  assign my_signal_2 = 0;

endmodule
```

```
async def generate_clock(dut) -> None:
    """Generate clock pulses."""

    for cycle in range(__stop/10):
        dut.clk.value = 0
        await Timer(1, units="ns")
        dut.clk.value = 1
        await Timer(1, units="ns")


@cocotb.test()
async def my_second_test(dut) -> None:
    """Try accessing the design."""

    await cocotb.start(generate_clock(dut=dut))
    # run the clock "in the background"

    await Timer(5, units="ns")
    # wait a bit

    await FallingEdge(dut.clk)
    # wait for falling edge/"negedge"

    dut._log.info("my_signal_1 is %s", dut.my_signal_1.value)
    assert dut.my_signal_2.value[0] == 0, "my_signal_2[0] is not 0!"
```

Example from
https://docs.cocotb.org/en/stable/quickstart.html

Why Cocotb?

Python is **much** faster to write than Verilog.

- Quickly create testbenches and models of hardware blocks
  - This has the added benefit of using two very different languages to describe modules, reducing the likelihood your simulated design and actual hardware have identical bugs

You can easily leverage existing Python libraries to make nice visualization tools:

- Matplotlib for performance plots
  - You could plot # of commits vs cycle to see how it varies over time
  - Or processor PC vs time to see if there are any patterns
- Tkinter for interactive GUIs
  - You could make nice tools to see where instructions are in your pipeline
    - Might be helpful for debugging or performance insights!

Sophisticated custom tools can net you some advanced feature points!