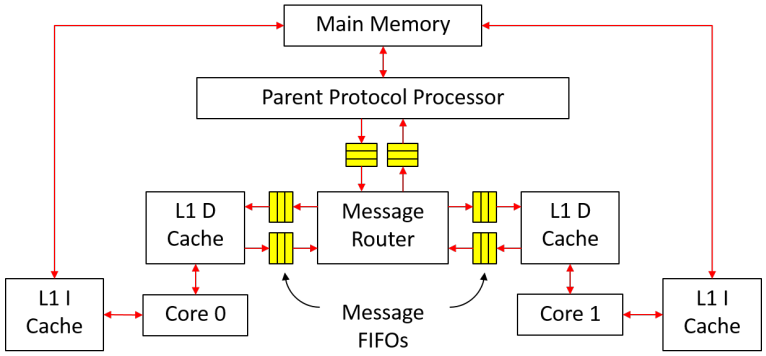
6.175 Final Project

Lasya Balachandran, Sanjay Seshan
lasyab@mit.edu, seshan@mit.edu

May 4, 2023

1 Multi-core Processor

Our first plan is to implement a multi-core processor. To do this, we will build off of our Processor and Cache labs and implement a shared cache hierarchy that maps to a main memory unit.



We will include two cores, so the processor will be able to run two instructions at the same time. These cores communicate using a message router to enqueue memory requests. Each core shares a single memory for instruction and data accesses so we will work to develop a hierarchy that will allow shared data usage. This means that higher cache levels will have to force an upgrade on the cache when shared core memory is used to avoid overwrites. Each core should be modular and easily scaled to meet the requirements of the system.

We will have our system consist of a variety of parts.

The first is the message FIFO. This is used to transfer upgrade and downgrade responses in a cache hierarchy. This works by having a response FIFO and a request FIFO that together form the message FIFO with associated logic.

We then have a message router to connect all the cores' L1 caches to the parent protocol processor. This part will send messages from the parent to the right L1 cache and vice versa.

Next, we will have a L1 data cache for each core. This is generally a normal cache interface.

Finally, we have the parent protocol processor which will handle all the messages to and fro memory and the cache. It handles upgrade and downgrade requests.

2 FPGA

Furthermore we plan to run this system on an FPGA unit. This unit will be cloud based and provided by the 6.175 instructors. Since the multi-core processor will likely only run on the server-based unit rather than the small FPGA, we will use the server-based one. We expect that this will involve some work with Verilog and learning to program the controllers. The Bluespec compiler will generate the Verilog code, from which we will use the associated synthesizing tools to push to the FPGA.

3 Other

We expect that we will have to first merge our code from the Processor and Cache labs to create a fully functioning pipelined processor. We will also add register and memory access bypassing. We will also have to modify these afterward to implement the aforementioned properties. This will still be implemented in a four stage pipelined processor.

4 Evaluation

We propose a series of tests to evaluate our system.

- Run two different programs, such as two different sorts, on each core
- Run two different sets of instructions
- Distributed matrix multiply that will divide operations between each core

We will use the timing results on this and a standard single core processor to evaluate the improvement in performance.

5 Sources

http://csg.csail.mit.edu/6.175/labs/project-part2.html
Course material from Canvas

6 Weekly Updates

6.1 Week 0

Initial version of the outline above.

6.2 Week 1

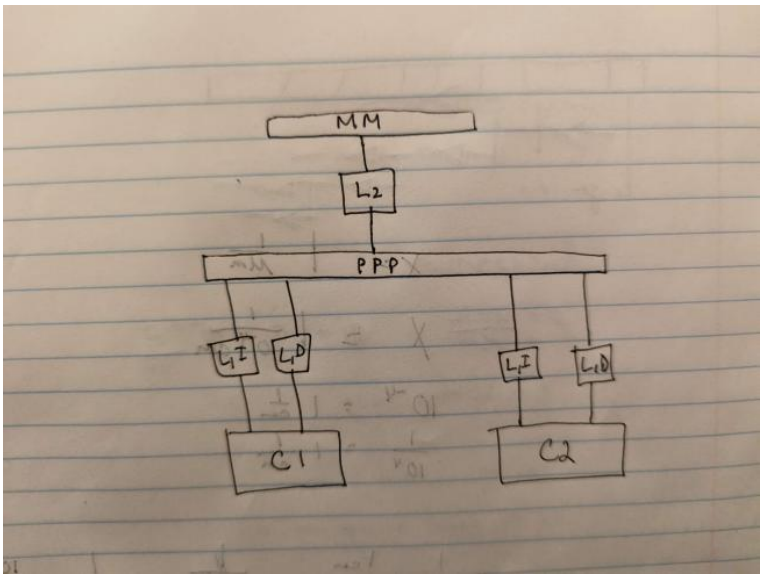
We have outlined above in more detail our plans to implement our system in the coming weeks. Next week we will start on the actual code for the Multi-core processor. Eventually we will start learning about FPGAs.

6.3 Week 2

Worked to create a L1 and L2 cache within a single core processor. Worked to merge the cache code with the RISC-V processor code. We created a two layer cache with 32-bit addressing.

6.4 Week 3

We finished merging the cache into the processor. This functions as a shared L2 cache for data and instructions, while each has its own L1 cache. L2 is connected to a single ported main memory which stores lines of data. L1 returns words. L2 is twice the size as L1. The processor uses a new cache interface in lieu of directly connecting to memory. We also started implementing the multicore processor, specifically the aforementioned parent protocol processor. We will adjust the cache hierarchy to match. We will keep a shared L2 cache before main memory.



6.5 Week 4

Bonjour, Thomas. Comment ça va? Nous avons connecté le cache au processeur. Nous avons créé le processeur de protocole parent aussi.