ECE 437L Final Report

Course – ECE 437 L
Tessie McInerney
Pranjit Kalita

TA: Chuan Yean Tan

April 29, 2016

1 Overview

In this report, we will be comparing our single core pipelined (with and without caches) processor to the dual core processor. For this analysis, we compared the clock frequencies, instruction latency, performance in MIPS, and the FPGA resources utilized by each design. Finally, we calculated the observed speedup from sequential to parallel execution of the same program. The data on these comparisons come from running mergesort.asm (for single core) and dual.mergesort.asm (for dual core), in mapped version. We selected this program due to its abundance of hazards and complex use of branches, which cause a lot of cache activity, and believe that it will be a good depiction of how various elements of our forwarding, branch prediction, cache hits (for cache design), cache coherence (for multicore) are tested. To get these results, we used a latency of 7.

Both single core and dual core processors have their share of advantages and disadvantages. The single core is expected to take longer number of cycles to complete and by extension, a longer time to finish program execution. The dual core on the other hand, is expected to take full advantage of parallelism and in the best case, cuts program execution time by two. On the flip side, a dual core is generally more complex to design and considerations of factors foreign to a single core such as cache consistency, atomicity, lock handling, best lock implementation (whether to use ll-sc or other versions of RMW locks for example), etc. take priority. Moreover, since memory access is a major bottleneck and the requirement of cache consistency is such an important factor to maintain correctness of the program (eliminating any possibility of stale copies), every time there is a permission change in multicore, a cache-to-cache transfer is followed by a cache-to-memory transfer. This means a huge penalty in terms of clock cycles. Moreover, specific to our implementation, unless the other core has write permissions on them thereby invaliding the requesting cache, we always do memory-to-cache transfer instead of cache-to-cache from the cache having read permissions. This was a design opportunity which if we had been successful in taking advantage of, could have saved us a lot of clock cycles. But since we did not, we have to incur a lot of penalties in terms of clock cycle delays.

2 Processor Design

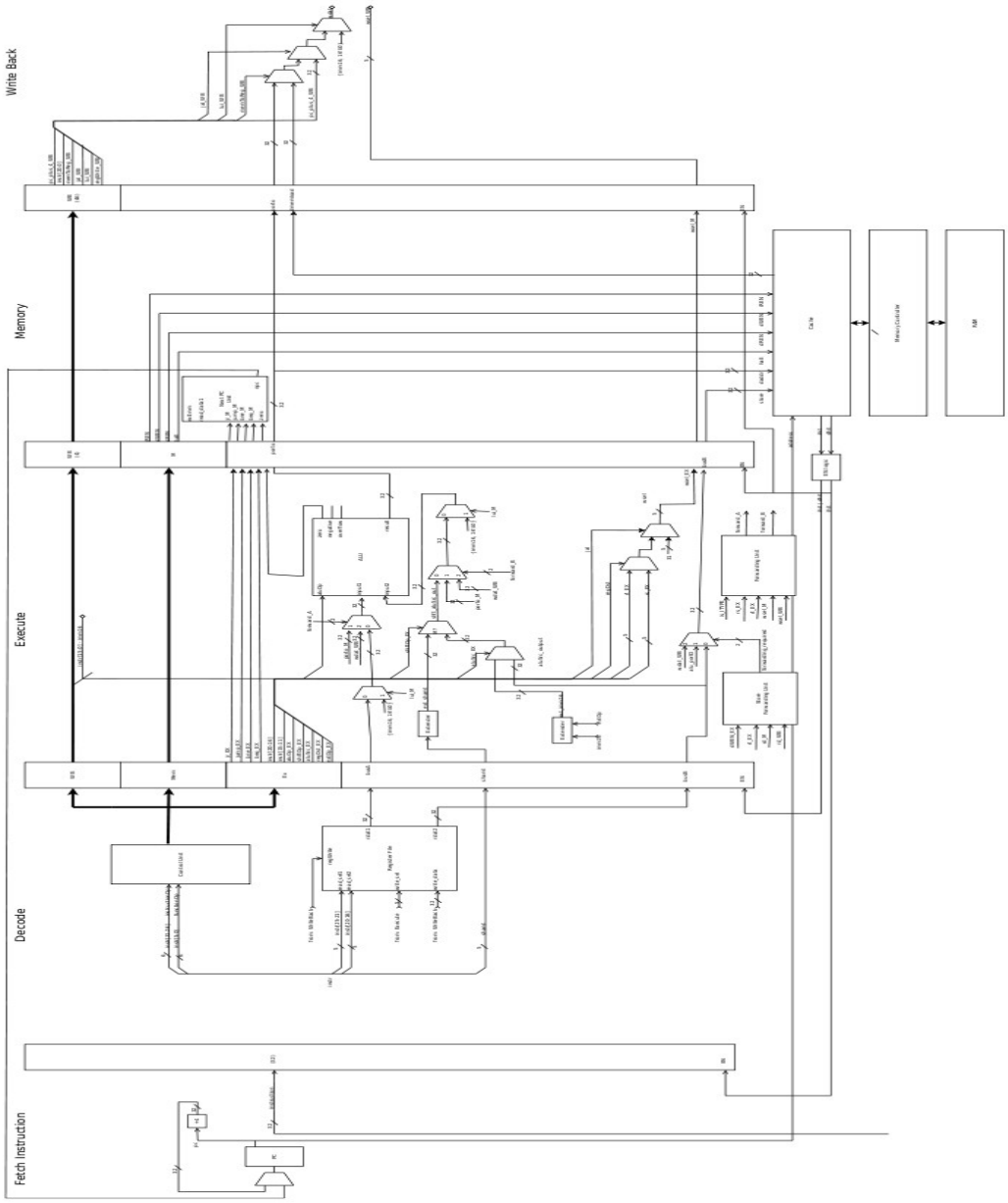

Fig. 1 : The pipelined processor datapath.

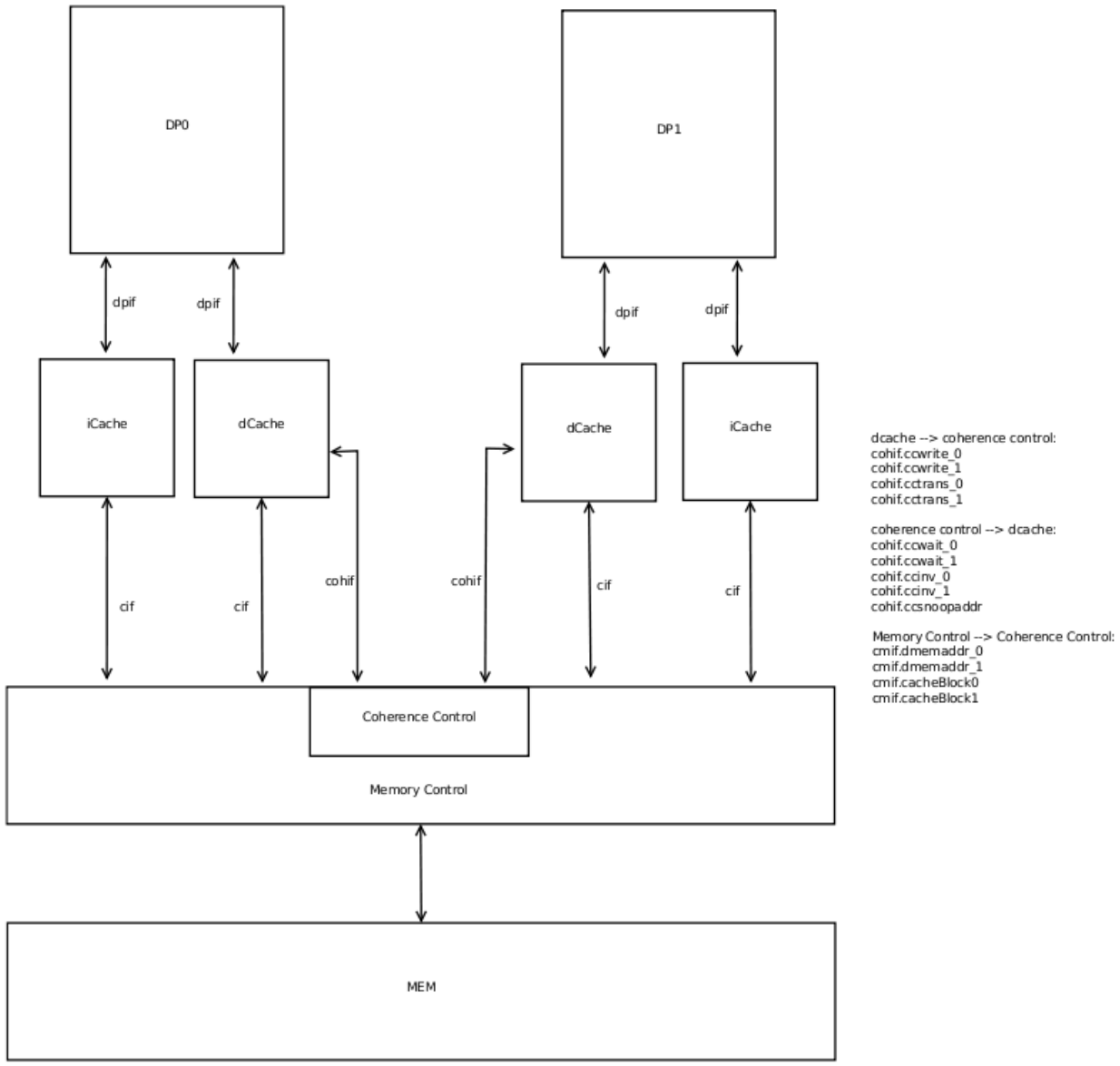

Fig. 2 : Block Diagram showing the interfaces of each core with the caches, and from caches to the memory controller.

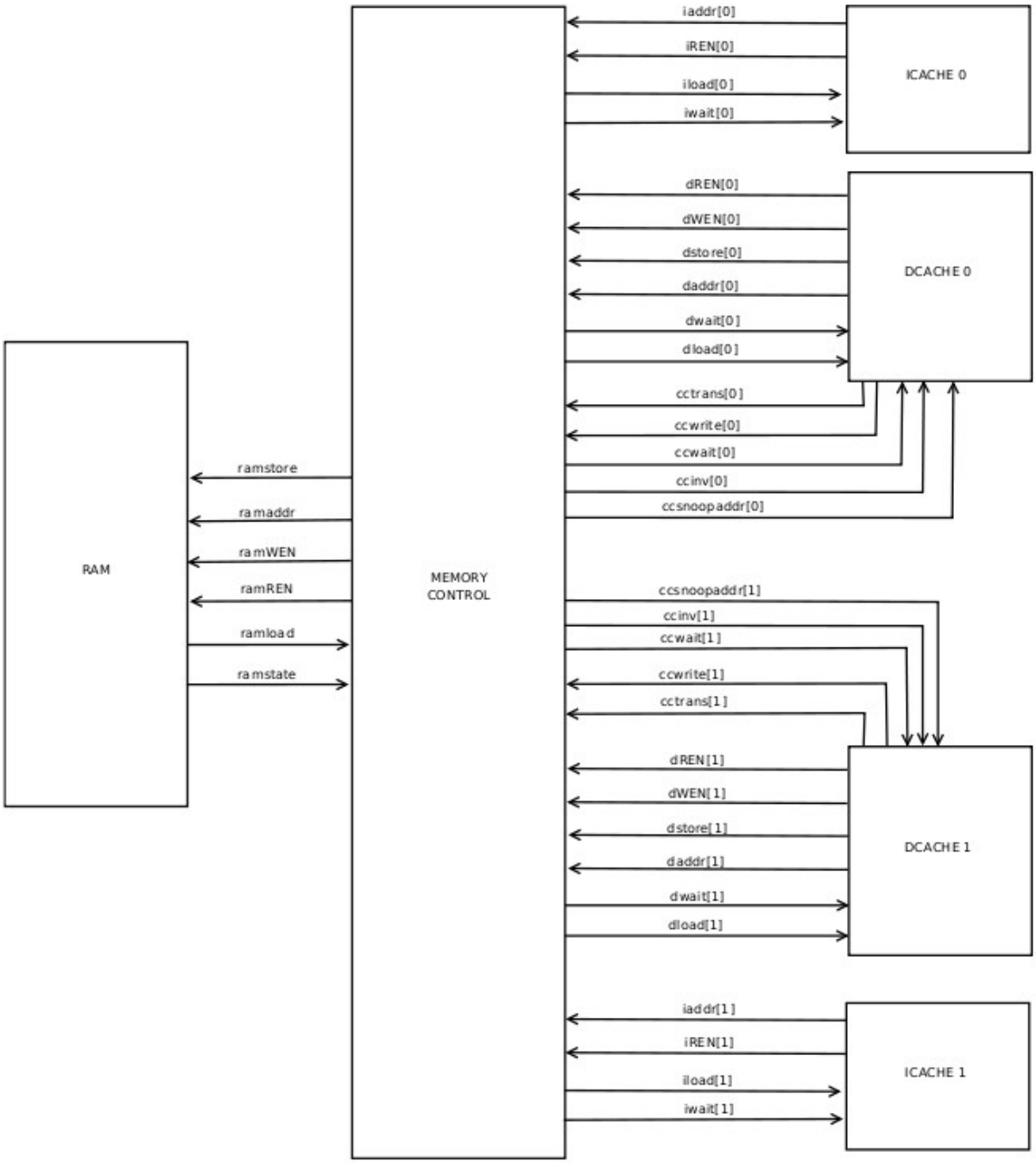

Fig. 3 : Interface between I- and D-cache of each processor and the memory controller.

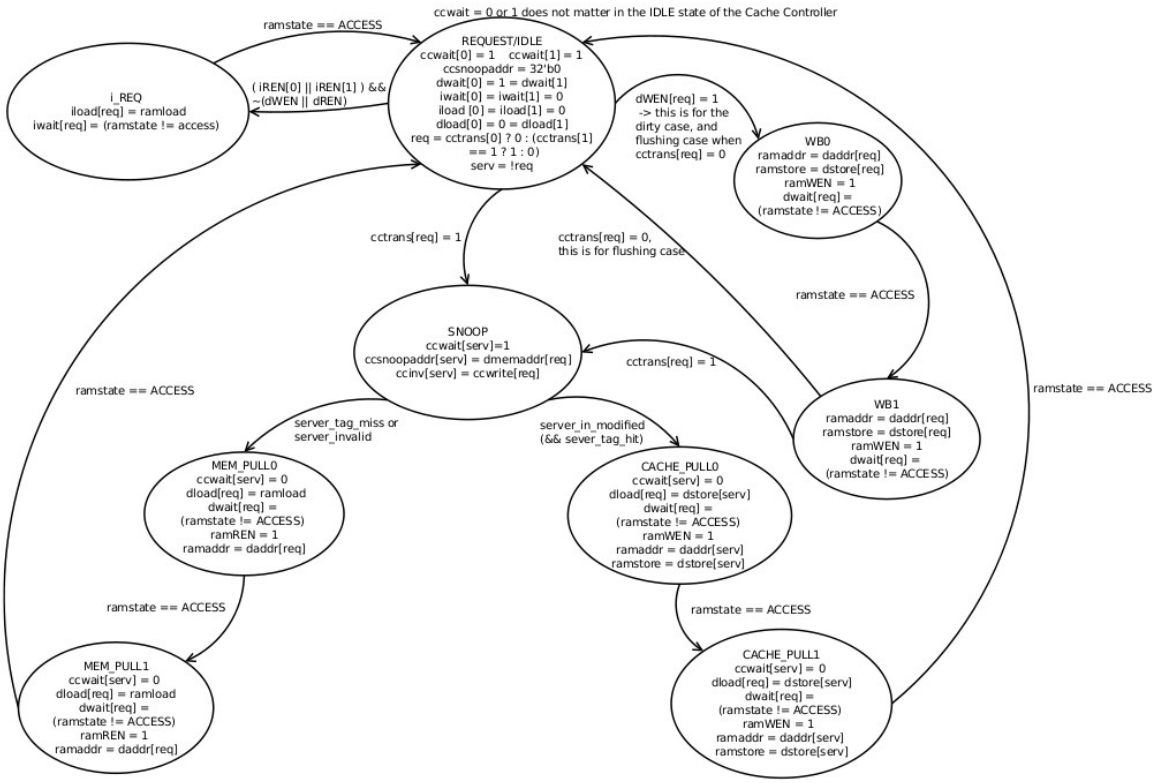

Fig. 4: State Transition Diagram of the coherence controller

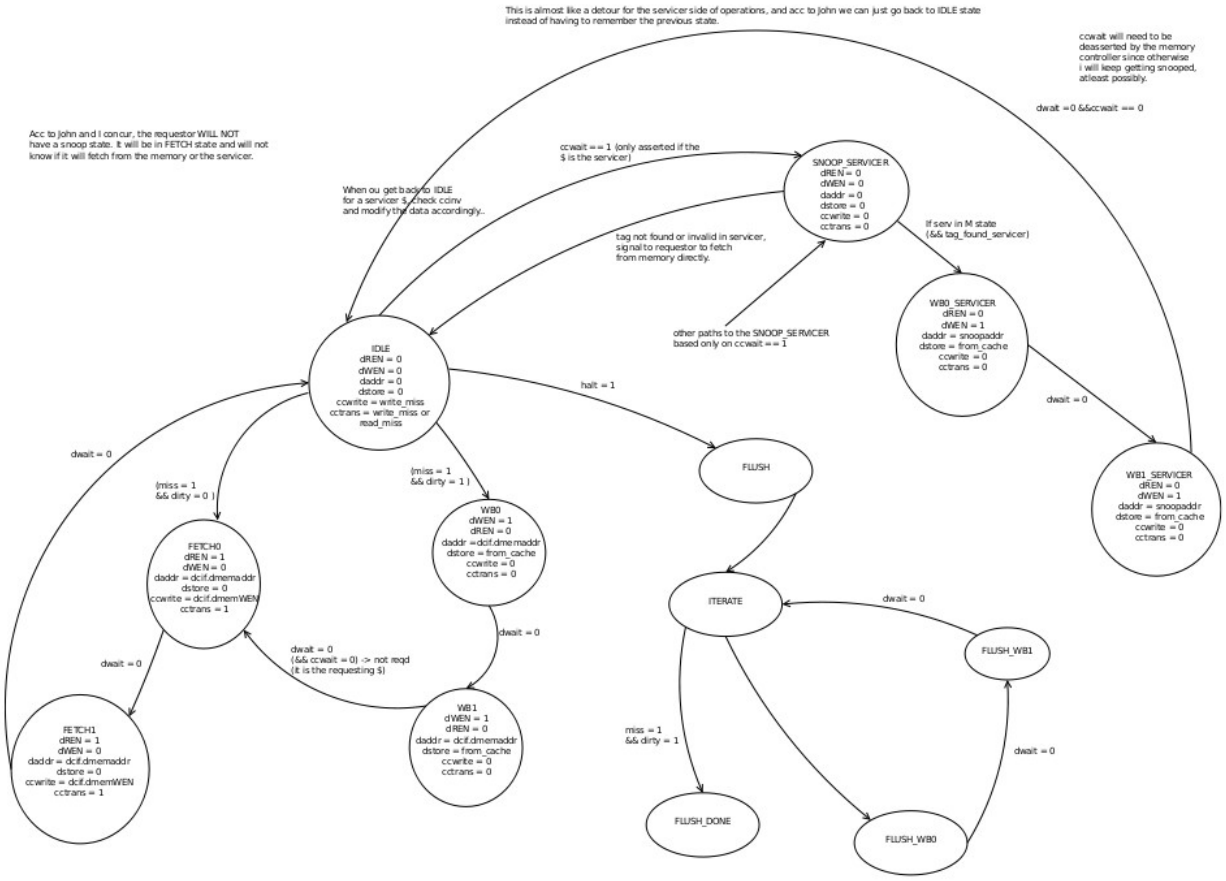

Fig. 5 : State Transition Diagram of the Dcache

3 Results

	Pipelined without caches	Pipelined with caches	Dual Core
Frequency	65.29 MHz	64.5 MHz	133.94 MHz
Average CPI (CPU Cycles / # of instructions)	17.91 = (96705 / 5399)	3.47 = (18771 / 5399)	4.26 = (23093 / (3164 + 2250))
Latency (5 * Period)	76.58 ns	77.52 ns	37.3 ns
Performance (MIPS) (Freq / CPI * 10^6)	3.64	18.59	31.44
FPGA Resources	Total combinational: 3,640 Logic registers: 1,976	Total combinational: 6,799 Logic registers: 4,447	Total combinational: 15,243 Logic registers: 8,763

Speedup from sequential to parallel program

= Time (Pipelined with caches) / Time (Dual Core)

= Performance (Dual Core) / Performance (Pipelined with caches)

= 31.44 / 18.59

= 1.69

4 Conclusions

As expected, implementing a parallel execution version of mergesort in dual core led to a speedup by a factor of ~1.7 compared to the single/sequential version of the same. This is consistent with the structure of mergesort's algorithmic breakdown, which breaks down a given list of numbers into two halves, and each half is then executed independent of each other in the divide part of the overall divide and conquer algorithm. Creating these two divisions ideally would be expected to yield a speedup of 2, but one of the reasons for not exactly getting 2 as the calculated speedup is due to the conquering part of the two divided merged lists at the final stage, which according to the given .asm file takes place in Core 1. Thus, the way the program is written is not aimed at maximizing the benefits of divide-and-conquer fully, but some of them cannot be avoided either. Furthermore, maintaining cache coherence implies additional overhead in terms of clock cycles incurred as penalty when writing back to memory and/or receiving updated values from memory, which is an extra overhead in dual core that a single core with caches does not have to deal with.

As expected, individual data values such as frequency and average CPI increased in the dual core compared to its sequential counterpart. An anomaly was observed in the huge CPI calculated for the pipelined processor without caches, but this could be attributed solely to the large number of instructions executed by the processor, which in turn was magnified by the presence of branch prediction logic, which incurs a large penalty each time there is a misprediction in the 2-bit branch predictor. The way our branch predictor is initialized is not customized for this mergesort assembly provided; however it works well for all the other files.

5 Contributions

Both Tessie and Pranjit contributed to the design of the caches and multicore, and both worked together throughout the design phase to create the State Transition Diagrams for the multicore's memory controller and dcache. Furthermore, both were present throughout the coding and debugging process, and worked together to fix the issues that arose in the design(s).