COL216 Assignment 5

Harshil Vagadia
Rishi Shah

April 2021

1 Problem Statement

Extend the MIPS interpreter to accommodate multiple CPUs with a single DRAM and MRM.

2 Design Choices

We have made the following design choices:

1. All the input files are to be put in a single folder. The number of CPUs and the total cycles to execute the program are given as command line arguments.
2. Each CPU executes its instruction independently of others, but they share a common DRAM to access memory and MRM to manage requests.
3. To avoid unforseen behaviour, multiple CPUs cannot access same memory address. This is ensured by dividing the memory into blocks and a block is assigned to each CPU. The memory address in each input file is relative to its block.
4. Only one request can be added to the DRAM in a single cycle (due to hardware restrictions). Therefore multiple CPUs cannot execute lw/sw instructions in the same cycle.
5. We remove some lw/sw requests if they are redundant i.e does not affect the output of the program. The redundant requests are mentioned in section 3.
6. We forward the values of some register in a instruction to another instruction. Forwarding is described in section 4.
7. The maximum size of the DRAM is set to 15. This size is determined after experimenting on some standard programs and finding the size that gives the maximum throughput. The experiment details are given in section 8.
8. We incorporate some delay in MRM i.e. the selection of which request to process next takes some finite clock cycles. This delay is a function of the current DRAM size and maximum DRAM size (shown in section 7). The selection of next request can occur in parallel to a DRAM request execution.
9. The selection of next request is done on the basis of certain rules described in section 5.
10. While the DRAM is executing, we can execute CPU instruction which are safe i.e. the instruction is not dependent on the lw/sw requests currently being processed. Safe checking heuristics are defined in section 6.
11. In accordance to MIPS architecture, we do not allow writeback to multiple registers in a CPU in the same cycle.

3 Redundant Requests

The following scenarios result in redundant requests:

1. In case of multiple consecutive lw instructions to the same register, all request except the last one are redundant.
2. In case of multiple consecutive sw instructions to the same memory location, all requests except the last one are redundant.

4 Forwarding

Forwarding can occur when this scenario occurs:
A request is issued to write a register to a memory location and then is followed by a load instruction to the same location. In this case the value of the first register can be forwarded to the second register without waiting for the first request to complete.

5 DRAM Request Selection

Whenever a DRAM is done processing a request, it must choose between the available requests. The target is to reduce the number of clock cycles to promote faster execution. Our algorithm chooses the next request only based on the information available on that cycle i.e. there is no lookaheads of instructions (greedy approach). The following rules are obeyed while choosing the next request:

1. All the requests involving a particular register should be processed in the order (but not necessarily consecutively) they are received to preserve semantic action. For this, we create 32 queues of requests, one for each register. While processing the requests, we add the request to the corresponding queue. While choosing a request to process, we can choose only from the front of the queues.

2. While choosing a request from the front of all the queues, we choose a request (if it exists) belonging to the same row as the buffer row. This will reduce the number of buffer row reads and writes, and in turn decrease the number of clock cycles.

3. If there are multiple requests corresponding the same DRAM row at the front of the queues, then we choose the request which was added earlier. This is to preserve semantics of the instruction accessing the same memory location.

4. If there are no requests corresponding to the current buffer row, then we are free to choose any request from the front of the queues. However, we first check if there is any unsafe instruction currently pending. If yes, then we choose the row which corresponds to the unsafe register queue's front request. This way we can ensure that the main instructions are also executed as quickly as possible.

5. Finally, if none of the rules apply, we just process the request that was added earliest. This ensures that all the CPUs gets a chance to process their request.

6 Checking Safe Instructions

We use the following rules to check the safety of an instruction.

1. j instructions are always safe to execute.

2. lw and sw instructions are safe to execute if their 2nd register is not invloved in any lw request.

3. All the other requests are safe if their first register is not involved in any lw/sw requests and the second and third registers are not involved in any lw requests.

7 MRM Hardware and Delay

We keep the MRM delay i.e. the number of cycles to determine the next request to process as MAX_SIZE/2. We envision the following DRAM structure:
The DRAM is cache like linear memory. Each new request is added to the leftmost or rightmost end, whichever is empty. While removing a request, we shift it to one of the ends, whichever is closer, and then remove. This ensures the empty spaces are always present at the ends.
Further there are control signal emerging from each block in DRAM and going to a control unit. The control unit determines which request to select, forwarding and redundant requests. All this can be done in a single cycle by combinational circuits.

8 Determining Optimal DRAM Size

The optimal DRAM size was calculated by performing suitable experiments. There are two opposing factors in general, more DRAM size will mean more requests can be added, but it will result in longer delay.
Thus we measured the throughput of sorting programs running on multiple CPUs after varying the DRAM size. We used sorting beause they are the most common and frequent programs to be computed. The figure below shows the graph, and we can see that a DRAM size of around 15 gives optimal performance.

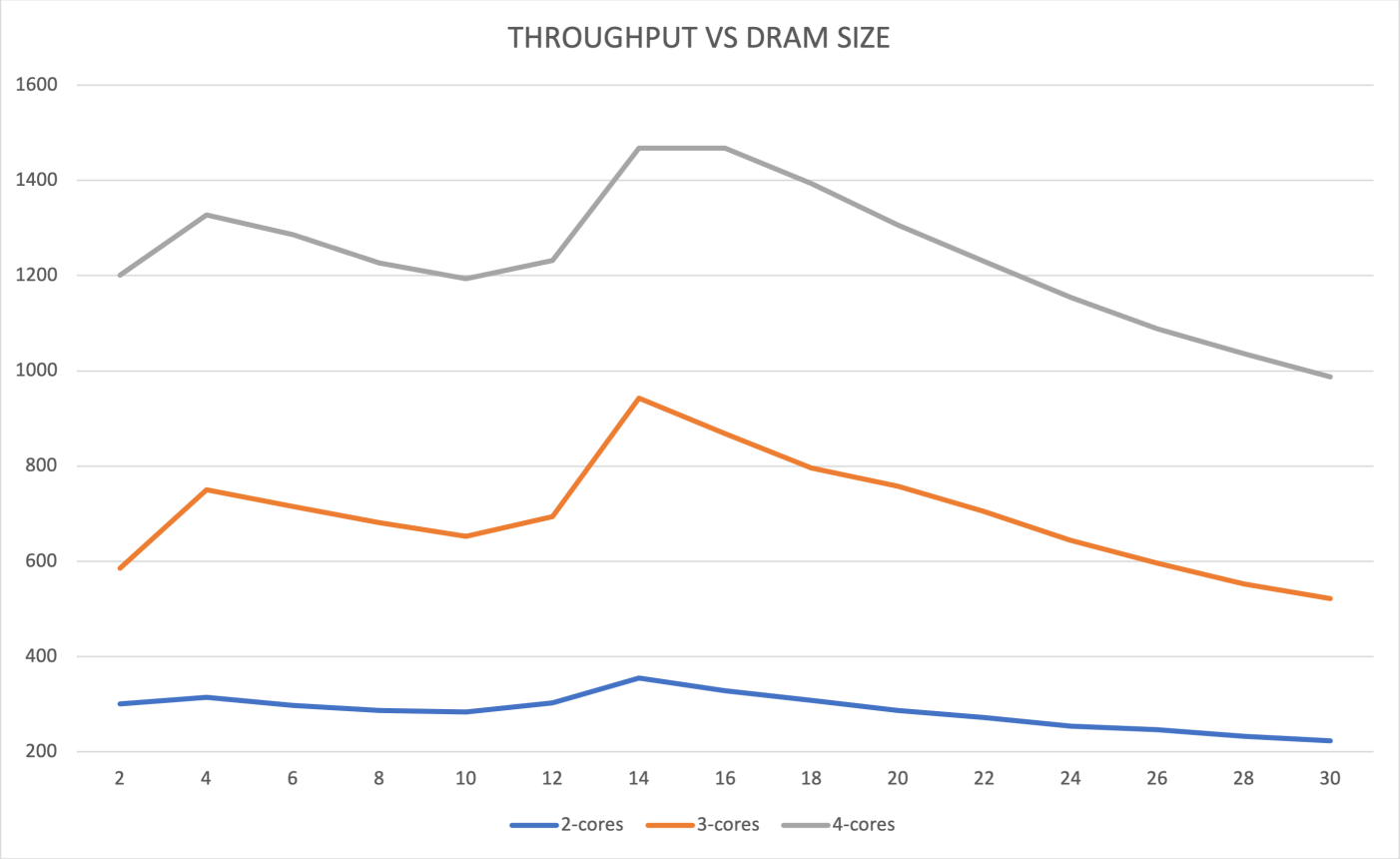

Figure 8.0.1: Parameter vs Error

9 Strengths and Weaknesses

9.1 Strengths

1. We achieve significant optimisation by following the reordering rules mentioned in section 3. Compared to the unoptimised counterpart, our program takes much less clock cycles, while ensuring program semantics are not violated.

2. There are no forward lookups in our algorithm. This means over interpreter is fast and takes $O(clock\_cycles)$ running time.

3. Our algorithm also tries to give priority to the CPU by trying to free up busy instruction as soon as possible.

4. Our algorithm also removes redundant instructions (if any), does forwarding when needed, that have no impact on program semantics.

5. Our program uses the optimal size of DRAM which was calculated from the graph on most typical algorithm used by programmers that is the sorting algorithm. This size will be optimal for most of the algorithms.

6. Our MRM is implemented keeping hardware structure in mind. We have provided a satisfactory hardware abstraction along with our software implementation.

9.2 Weaknesses

1. Our algorithm uses queues as a data structure, whose size may not be optimal for each and every case.
2. Since we follow a greedy algorithm, our optimisation might not be the best possible. In general, it might be more beneficial to parse through the entire program and figure out the most optimum ordering of requests.
3. Our algorithm does not optimises itself based on the number of cycles or number of CPUs given. For any cycles or CPUs given the execution will be the similar to maximise the throughput.

10 Testing Strategy

Extensive testing was done for the assignment.
We compared the output of the test cases on both optimised and unoptimised assignment-4 interpreters and ensured that the semantic actions are the same in both the cases. We also compared the number of clock cycles and the row buffer changes in both cases.
The folder funtion_tests contains test cases that show some algorithmic feature described above.The folder syntax_tests contains some tests to check for syntax errors. The folder tests contains various types of test cases like-

1. Typical test cases based on arrays. Sorting , reversing , addition of array elements are included.
2. Multi-core test cases are included to check the working up to 16 CPU units.
3. Test cases including large number of lw,sw instructions and large loops are included. This gives us better comparison between the unoptimised and optimised versions.
4. Some random test cases are included to complete the testing procedure.

11 Results

Our program achieved significant speed up over unoptimised counterpart while preserving semantic action.