Memory Request Ordering

Aditya Goel — 2019CS10671

April 10, 2021

Program Design Specifications: Input & Output Format

- The C++ Program can be compiled using the following command line instruction
  g++ -o main main.cpp
- The executable can then be run by the following command line instruction
  ./main ROW_ACCESS_DELAY COL_ACCESS_DELAY INPUT_FILE_NAME
- ROW_ACCESS_DELAY and COL_ACCESS_DELAY are the delays in number of cycles for accessing row and column from the DRAM respectively.
  - An "Error: ROW_ACCESS_DELAY not specified correctly" or "Error: COL_ACCESS_DELAY not specified correctly" is printed if the ROW_ACCESS_DELAY or COL_ACCESS_DELAY are not integers respectively.
- INPUT_FILE_NAME is the name of the MIPS assembly language program (as text file, NOT machine instructions).
  - An "Error: File does not exist or could not be opened" is printed if the file doesn't exists or is not a text file.
- At every clock cycle the program prints the cycle number, the address of completed instruction and any modified registers, any modified memory locations or any activity on DRAM.
- After execution completes, the program prints relevant statistics such as the Total execution time in clock cycles and the Number of row buffer updates.

Strategy & Approach Used

- Each instruction in the input file is stored in a vector line by line. Any instance of a comment is ignored while any instance of a label is stored in a HashMap that maps that label as a key to the corresponding line number as its value.
- As we iterate through the instructions one by one in a sequential manner, each itertion of an instruction involves processes of parsing the instruction and performing relevant operations on concerned registers or memory locations.
- Appropriate errors are raised during parsing each line, if any instruction other than add, addi, sub, mul, slt, j, beq, bne, lw or sw is encountered or if these instructions are not found in the format accepted as MIPS assembly language.
- The DRAM is implemented as a 2-Dimensional array of size 1024 x 1024 bytes. A single Read/Write in a lw/sw instruction corresponds to accessing a 32-bit (4 byte) value from the DRAM. Within a row, a given piece of data is located at a column offset, so a memory address could be thought of as consisting of a ROW ADDRESS and COLUMN ADDRESS.
- ROW_BUFFER is implemented as a 1-Dimensional array of size 1024 bytes which helps in both accessing the value stored at a particular memory address or store and update any memory address by copying the necessary row from the DRAM and writing it back to DRAM whenever required.
- The reordering of DRAM requests as an ordered queue has been implemented using an ordered map, with the memory address as the key to the corresponding instruction, without causing any change to the program's semantics. More about reordering of DRAM requests and benefits of reordering has been mentioned in the strengths & weakness section.

Strengths & Weaknesses

- Use of Arrays and HashMaps for DRAM, ROW_BUFFER and Labels helps in efficient access to the memory and labels during the execution of the program.
- The reordering of DRAM requests as an ordered queue has been implemented using an ordered map, with the memory address as the key to the corresponding instruction. DRAM requests associated to lw/sw instructions can be reordered in non-blocking memory access fashion unless we find another instruction involving the register as associated with previous lw/sw instructions.
- The reordering of DRAM requests is done in a stable manner, such that DRAM requests associated to any given memory address are executed in the same order as given in the MIPS assembly language code.
- Reordering of DRAM requests helps improve the Total Execution Time in clock cycles of the program as we cut on significant clock cycles otherwise required for row access delay and column access delay.
- Our implementation of Non-Blocking Memory Access, allows for execution of instructions even if a lw/sw instruction has not completed, only until we do not encounter any instruction that involves any register that was mentioned in the corresponding lw/sw instruction or a bne/beq/j instruction, or until we execute instructions one less than the delay in number of cycles that the corresponding lw/sw instruction had to face. All such instructions which cannot be executed when the execution of corresponding lw/sw instruction has not completed are said to be unsafe for execution.
- The Non-Blocking Memory Access further considerably reduces the total execution time in clock cycles. The effect increases in case we have a large value of ROW_ACCESS_DELAY and COL_ACCESS_DELAY.
- This reduction in total execution time in clock cycles depends on encounter of instructions which are marked unsafe to be executed while a lw/sw instruction is not completely executed. Programs which involve such instructions just after the lw/sw statements will lead to low reduction in the total execution time in clock cycles.

Test Cases & Corner Cases Handled

1. In case of a comment both, as a separate line or along with those Instructions are completely ignored by the program.
   - A test on input
     ```
     #My New Increment Program
     addi $t0, $t0, 1 #Incrementing $t0
     ```
     resulted in a successful execution of the program and incremented the value of $t0 by 1 , ignoring any comments that came in between.
   - A test on input
     ```
     #My New Increment Program
     #addi $t0, $t0, 1 Incrementing $t0
     ```
     resulted in a successful execution of the program with no Instructions to be evaluated.
2. Entering any invalid instruction, other than those specified in the program design specifications prints "Error: Invalid operation" error.
   - A test on input
     ```
     move $t0, $t1
     ```
     resulted in an "Error: Invalid operation" error.
3. If results of instructions such as add, sub or addi results in an arithmetic overflow, corresponding errors are raised.
   - A test on input
     ```
     addi $t0, $t0, 1
     addi $t0, $t0, 2147483647
     ```
     resulted in an "Addition Overflow" error.

4. If at any stage of program, the number of instructions executed is found to exceed 131,072, the execution stops, and print "Number of Instructions exceeded the maximum allowed values" error.

5. If the input to commands such as j, bne, beq is a label which is not present in the Source Code, a "Label Not Found" error is raised.

   – A test on input
   ```
   label1:
   addi $t0, $t0, 1
   bne $t0, $zero, label2
   ```
   resulted in an "Label Not Found" error.

6. Entering a memory address beyond the allowed bounds of (0-1048576) or a memory address that is not a multiple of 4, prints "Invalid Memory address" error.

   – A test on input
   ```
   addi $s0, $zero, 1048580
   sw $t0, 0($s0)
   ```
   resulted in an "Invalid Memory address" error.

7. Entering an input with register names other than the 32 registers specified, prints an "Error: Invalid register" error. Also entering non-32 bit integer values in instructions such as addi, prints an "Error: Given value is not a valid number" or "Error: Out of Range Integer" error respectively.

   – A test on input
   ```
   addi $m0, $m0, 1
   ```
   resulted in an "Error: Invalid register" error.

   – A test on input
   ```
   addi $t0, $t0, abc
   ```
   resulted in an "Error: Given value is not a valid number" error.

   – A test on input
   ```
   addi $t0, $t0, 2147483648
   ```
   resulted in an "Error: Out of Range Integer" error.

8. Reordering of DRAM request corresponding to lw/sw instruction resulted in a correct output, i.e. similar to the case of not reordering the DRAM requests. It reduced the Total Execution Time Considerably as opposed to the case of DRAM requests not being reordered.

   – A test on input
   ```
   main:
   addi $s0, $zero, 1000
   addi $s1, $zero, 2500
   addi $t0, $zero, 1
   addi $t1, $zero, 2
   addi $t2, $zero, 3
   addi $t3, $zero, 4
   sw $t0, 0($s0) store 1 at location 1000
   sw $t1, 0($s1) store 2 at location 2500
   sw $t2, 4($s0) store 3 at location 1004
   sw $t3, 4($s1) store 4 at location 2504
   lw $t5, 0($s0)
   lw $t6, 0($s1)
   lw $t7, 4($s0)
   lw $t8, 4($s1)
   exit:
   ```
   The Program with ROW_ACCESS_DELAY = 10 and COL_ACCESS_DELAY =2, with reordering of DRAM requests took 60 cycles as the Total Execution Time, while an execution without reordering of DRAM requests took 180 cycles as the Total Execution Time.

9. Reordering of DRAM request corresponding to lw/sw instruction was done in a stable manner, i.e., instructions corresponding to a given memory address is executed in the same relative order as written in MIPS assembly language program.

   – A test on input

   ```
   main:
   addi $s0, $zero, 1000
   addi $s1, $zero, 2500
   addi $t0, $zero, 1
   addi $t1, $zero, 2
   addi $t2, $zero, 3
   addi $t3, $zero, 4
   addi $t4, $zero, 5
   sw $t0, 0($s0) store 1 at location 1000
   sw $t1, 0($s1) store 2 at location 2500
   sw $t2, 4($s0) store 3 at location 1004
   sw $t3, 4($s1) store 4 at location 2504
   sw $t4, 0($s0) store 4 at location 1000
   lw $t5, 0($s0)
   lw $t6, 0($s1)
   lw $t7, 4($s0)
   lw $t8, 4($s1)
   exit:
   ```

   The Program with ROW_ACCESS_DELAY = 10 and COL_ACCESS_DELAY =2, with reordering of DRAM requests executed instructions corresponding to a memory address in order of their appearance in the MIPS assembly language program. For example: Instructions sw $t0, 0($s0), sw $t4, 0($s0) & lw $t5, 0($s0) all corresponding to memory address 1000, are executed only one after the another, respectively, such that a register $t5 contains a value of 5 and not 1.

10. The Non-blocking Memory Access Allowed for execution of safe instructions in cycles where the ROW_ACCESS_DELAY & COL_ACCESS_DELAY number of cycles are required for complete execution of the lw/sw instructions.

    – A test on input

    ```
    main:
    addi $s0, $zero, 1000
    addi $s1, $zero, 2500
    addi $t0, $zero, 1
    addi $t1, $zero, 2
    addi $t2, $zero, 3
    addi $t3, $zero, 4
    sw $t0, 0($s0) store 1 at location 1000
    sw $t1, 0($s1) store 2 at location 2500
    sw $t2, 4($s0) store 3 at location 1004
    sw $t3, 4($s1) store 4 at location 2504
    addi $t0, $zero, 10
    addi $t1, $zero, 20
    addi $t2, $zero, 30
    addi $t3, $zero, 40
    lw $t5, 0($s0)
    lw $t6, 0($s1)
    lw $t7, 4($s0)
    lw $t8, 4($s1)
    exit:
    ```

    The Program with ROW_ACCESS_DELAY = 10 and COL_ACCESS_DELAY =2, reorders the DRAM requests along with non-blocking memory access which allows for execution of the multiple addi instructions in a safe manner. For example: The value stored at an address of 2500 is 2 and not 20, i.e. the instruction addi $t1, $zero, 20 which is after the instruction sw $t1, 0($s1) has no effect on the value being stored at the memory address of 2500.