**Approach:**

Extension of assignment 4 there we did same thing for single core CPU but here we will make program to run on multi-core CPU.

Store all programs(on-by-one) in instruction memory and make pointer for each program(starting) so that pointers will increase accordingly. For ‘n’ files there will be separate n registers and main memory will be divided in N equal parts in chunk like first 1/N part of memory for file1(also the start of a new program memory is chosen such that it is divisible by 4). Now all CPU has independent registers and memory range and we had 1024 sized row of DRAM. When we encounter lw/sw and if DRAM is not busy then simply use buffer row else enqueue the instruction in their respective queue (with the efficiency if Assignment4) and if there is blocking then code will look “regRowGroup” and cleans the blocking.

At last, some information has been printed:

* Total clock cycles.
* Total numbers of Instruction executed.
* IPC
* Final value of each register.
* Changed memory location and it’s value.

**DELAY ESTIMATION:**

1) If a pair of instruction in queue gets consecutive to each other such that first one is and sw

and second one is a lw from same memory (and register not busy anywhere else) then a

store to load transfer happens and the second instruction is put in queue to be serviced in

DRAM. As this just requires a transfer of values from a MRM memory to register so it takes a one cycle delay.

2) If a pair of instruction in queue gets consecutive to each other such that first instruction is lw

and second instruction is also a lw at same register then the previous lw is popped from the

queue as it would be overwritten later. This just requires a MRM request to be deleted whose location would already be know and hence it would just take one cycle.

3) If a duplicate instruction comes to MRM that is currently being run in the DRAM and the MRM is empty already than this instruction is not needed to be services in the DRAM and hence in one cycle this request is rejected.

4) If a pair of instruction in queue gets consecutive to each other such that first instruction is

sw and the second is a sw at same memory location then the previous instruction is popped

from the queue and is not serviced in DRAM. Similar as in case 2 it requires one cycle.

5) If a pair of instruction in queue gets consecutive to each other such that first instruction is lw

and the second is a sw at same memory location using the same register as in load

instruction then the current instruction is not put in queue as it does not change anything. So the current request is just rejected and requires one cycle of delay.

6) In case the queue is full when a DRAM request is issued to go to MRM than one cycle is needed to free up the space for the queue(other than the DRAM execution cycle) as it just requires a check for the 32 size spaced queue is full or not and till that time the request is halted and so other instructions requiring MRM requests had to wait for the completion of these delay cycles.

7) A copy of data takes place from the MRM to the DRAM processor which is estimated to give a one cycle delay.

8) When the next row group is to be decided to get executed in DRAM after executing all instructions of ongoing row group than a delay in MRM corresponding to popped requests will take place. This delay in worst case would be total number of instructions of lw/sw type executed during the whole run of program and during a single usage of the corresponding architecture it would be the number of total instructions that are allowed at once at max, but in average case this delay would be one. This can be implemented in the in looped manner and in the corresponding time if any instruction is to send a request to MRM than it would halt(similar to what happens in pipelining).

**Strengths:**

* Redundant instructions has been taken care. (eg. sw followed by sw at same address then first sw has no meaning) .
* Most of the cases only 1+1 clock cycle consumed to decide and queue to DRAM (average).
* Area required is high but very efficient in terms of clock(trade-off).
* This is a feasible linear time algorithm with respect to the number of instructions to which reduces the overall cycles requirement by reordering the DRAM servicing of instructions and reducing the number of writebacks and loading of rows required.

**Weakness:**

* If some instruction starts before cycle count reaches M and after M that instruction will end than we will let instruction execute.
* File count(N) greater 50 not allowed.
* Size of queue is max 32.
* MRM, if many checking happens then maximum 4-5 cycles can be consumed.
* Area requirement is high.

It’s a follow up of all earlier assignments and all other aspects have been explained earlier. The optimizations done to avoid some instructions to be executed in DRAM remains the same and the reordering optimizations also hold here. The difference is that the MRM delay for these optimizations is incorporated in the simulation now which are mentioned in delay optimizations.