CS422 - Project Report

Sourav Anand, 13709

Memory Access Scheduling Algorithm

Abstract:

There has been a great advancement in the technology used in processors. The processor's clock speed has increased significantly. However, the memory system has not developed at the same pace. The memory devices act as the bottleneck for the execution speed of the program as the memory latency is high as compared to clock speed of processor. This problem can be reduced by using the memory system efficiently. One of the way to do so is to schedule the memory requests to memory device. Scheduling algorithms which try to improve the row hit rate for the memory device helps in improving the performance.

Introduction:

The present day DRAM work in such a way that a row has to be opened in a row buffer before data from that row can be read. Incase, the data is to be read from a different row, the row currently in the row buffer should be closed first and should be precharged and after that the requested row has to be loaded into the row buffer. The opening and closing of the rows in row buffer takes a large amount of time leading to large overall completion time. Memory access scheduling algorithm plays a significant role in decreasing the number of rows opens and close calls (or serving the requests in such a way which leads to more row hits). This project looks into a scheduling algorithm for the memory device which increases the rows hits and at the same time is fair to all the memory requests so that in order to maximise the row hits, other threads do not starve.

The scheduling algorithm used in this project is such that it first checks if there is any thread with an outstanding request from a long time. If one such thread is found, the thread's request is issued. If not, then the scheduler tries to maximise the row hit rate by finding a page which hits will hit in the row buffer if issued. If one such request is found, the scheduler issues that request else it chooses the first request it can issue. The scheduler also schedules Read request first and drains write requests only if it crosses a given threshold (write buffer has only few free entries left).

The scheduling algorithm is implemented on the memory simulator USIMM and it is tested on the provided workloads for single and multi threaded programs.

Scheduler Design:

The scheduler is designed in such a way that it tries to increase the row hit rate and at the same time tries to be fair to the other threads. The scheduling policy is as follows:

1. The scheduler first checks if the write queue is above a given threshold (i.e. only few entries are left in write queue). If yes, then the scheduler schedules itself to drain the write queue first until it reaches a low threshold.
2. If Write drain is active, the scheduler first passes through all the requests and finds if there is a pending request with its priority equal to highest possible value of priority. If yes, the scheduler then schedules this request. If not then during the iterations, the scheduler remembers the first thread (if any), that will give a hit to the current DRAM configuration and issues this thread as it would lead to a row hit.
3. If there is no thread with highest priority or any access request which will hit in the row buffer, the scheduler simply chooses the first thread from the write queue which is issuable and issues its request.
4. Similarly, if write drain is not active, the read requests are issued in the Highest priority->Row Hit->First issuable order as it is done for write drains.
5. At the end, if there was a row hit for the last issued instruction, the scheduler simply checks both the queues to see if the currently open row will hit in any other request or not. If not, then the scheduler issues an auto precharge command to that row so that precharging is done as soon as the data is transferred. This helps in reducing the time taken to manually precharge when a different row's data is required. Also, the priorities are increased for all the threads(cores) for the current channel by the scheduler.
6. If there were no instructions in this cycle, the scheduler pre charges any already open row in order to save precharging time if any other new request comes for a different row.

The above scheduler design policy performs in such a way that it tries to increase the number of row hits and at the same time it is fair to all the cores having a memory access request by fulfilling them periodically. This favours the cores which are having CPU intensive program as their memory access request have some amount of time gap between them and since in last few issues of memory request, there was no access from that core, the scheduler will give it a higher priority when it comes. Also, the cores which are running Memory intensive program will have a lower priority. This can be seen that fulfilling the CPU intensive program's request will have better utilisation of the CPU . Also, if there is a request from a program but it is not being issued as the scheduler is trying to maximise the row hit rate, after some issues of other requests, the priority of the Memory intensive program reaches highest value after which the scheduler is forced to issue that request. It can be seen

that the scheduler is being fair to all types of threads and at the same time is improving the memory access time.

If the scheduler doesn't issue any instruction in a cycle, The scheduler issues precharge (if the precharge is allowed i.e. the row to which the scheduler is issuing precharge is not having any pending request) to all the banks having an open row. This helps in improving the access time for any other request which is added to scheduler's read or write queue and needs to perform its action on a different row. Since the row was already closed, the time taken for precharging would not be seen by the new request.

Results:

The above scheduling algorithm was implemented on memory infrastructure simulator USIMM and tested on the included workloads. The following are the results:

Workload	Config	Sum of execution time for FCFS scheduler (10 M cycle)	Sum of execution time for proposed scheduler (10 M cycle)	EDP (J.s) for FCFS scheduler	EDP(J.s) for proposed scheduler
MT-canneal	1 chan	418	420	4.23	4.29
MT-canneal	4 chan	179	178	1.78	1.77
bl-bl-fr-fr	1 chan	149	146	0.5	0.48
bl-bl-fr-fr	4 chan	80	76	0.36	0.32
c1-c1	1 chan	83	83	0.41	0.4
c1-c1	4 chan	51	47	0.44	0.36
c1-c1-c2-c2	1 chan	242	239	1.52	1.5
c1-c1-c2-c2	4 chan	127	118	1	0.85
c2	1 chan	44	43	0.38	0.36
c2	4 chan	30	27	0.5	0.39
fa-fa-fe-fe	1 chan	228	222	1.19	1.13
fa-fa-fe-fe	4 chan	106	99	0.64	0.55
fl-fl-sw-sw-c2-c2-fe-fe	4 chan	295	280	2.14	1.9
fl-fl-sw-sw-c2-c2-fe-fe-bl-bl-fr-fr-c1-c1-st-st	4 chan	651	625	5.31	4.85
fl-sw-c2-c2	1 chan	249	244	1.52	1.44
fl-sw-c2-c2	4 chan	130	121	0.99	0.83
st-st-st-st	1 chan	162	159	0.58	0.56
st-st-st-st	4 chan	86	81	0.39	0.35
Total:		3312	3211	23.88	22.41

The above table contains comparison of key metrics on FCFS and proposed schedulers. c1 and c2 represent commercial transaction-processing workloads, MT-canneal is a 4-threaded version of canneal, and the rest are single-threaded PARSEC programs.

Conclusion

We can see from the results that the proposed scheduler has a smaller total execution time for all the programs than the FCFS scheduler. It improves the running time by approximately 3%. Also, the Energy delay product for the proposed scheduler is small than the FCFS scheduler by 6.15%.