A New Memory Scheduling Policy for Real Time Systems

Ankita Samaddar*, Moumita Das† and Ansuman Banerjee‡
Indian Statistical Institute
Kolkata 700108, India
Email: *anki.samaddar@gmail.com, †moumita.das@isical.ac.in, ‡ansuman@isical.ac.in

***Abstract*—In this paper, we propose a new memory DRAM controller scheduling policy for scheduling tasks in real time systems. Our proposal involves a memory bank aware partitioning strategy to partition the requests across banks based on a cost function on some task parameters to schedule memory requests so that the number of deadline misses get reduced significantly in a real time system. We used the Malardalen Worst Case Execution Time (WCET) benchmark programs as our real time tasks. We generated traces of these benchmark programs by running them on an X86 processor. We have developed an end to end setup from processor to memory and our results have been compared with state of the art DRAM controllers. Experimental results on these benchmark programs show the efficiency of our proposed scheme.**

I. Introduction

A real time system is an information processing system which has to respond to an externally generated input stimuli within a finite and specified period. Correctness of a real time system depends not only on the logical result of input but also on the time at which the result is produced. In these systems, the main challenge is to guarantee maximum number of execution of real time tasks by reducing the number of deadline misses. Several scheduling algorithms have been proposed to schedule the tasks at the processor level. Uniprocessor schedulers mostly use Earliest Deadline First (EDF) [12] or Rate Monotonic Scheduling (RMS) [13] to schedule tasks at the processor level. Scheduling of tasks in multi-core systems has also been proposed in [3]. Each task instance consists of multiple instructions, some of which are compute intensive, while others are memory intensive. In modern DRAM architectures, instructions are executed on the basis of a row-hit policy, i.e., instructions which result in row hit are given preference to execute first. Though EDF or RMS is carried out at the processor level, most of the real time tasks are not executed in the same sequence in the memory as they are scheduled at the processor. As a result, most of the real-time tasks fail to meet their deadlines while waiting in the buffer for memory access if not scheduled and executed within their deadline.

Several techniques have been proposed and adopted in real time predictable DRAM controllers. In [14], PALLOC, a DRAM bank aware memory controller has been proposed which exploits the page-based virtual memory system to avoid bank sharing among cores. [7] proposes techniques to provide a tight upper bound on the worst-case memory interference in Commercial off-the-shelf multi-core systems. A predictable DRAM controller design has been proposed in PRET [9], where DRAM act as multiple resources that can be shared between one or more requests individually by interleaving accesses to blocks of DRAM. [1] proposes bank interleaving and a close page policy with a pre-defined command sequence. Again [2] suggests a Credit-Controlled Static-Priority to provide minimum bandwidth for each request with bounded latencies. [8] deals with an analytical model to compute worst case delay considering all memory interferences generated by co-running tasks. [4] proposes a method for composable service to memory clients by composable memory patterns. In [6], memory requests are scheduled using time-division-multiplexing scheduler and a framework has been developed to statically analyse the tasks to meet the timing requirements of all tasks.

In this paper, we propose a bank aware memory scheduling policy to schedule tasks at the memory controller on the basis of a cost function. We have implemented a two-level scheduler, one at the processor level and the other at the memory controller, to schedule tasks in real time platforms. Our proposed method has been compared with existing state-of-the-art memory controllers on different benchmark programs of the Malardalen WCET [5]. Results on different benchmark programs show the efficiency of our proposed method.

The rest of the paper is organized as follows: Section II discusses some background concepts. In Section III, we discuss the problem in the context of DRAM with the help of an example and also our proposed solution. Section IV describes the implementation and results and Section V concludes the paper.

II. Background

This section consists of a brief description of the DRAM.

A. Organization of DRAM

Main memory is stored in DRAM cells with higher storage density. DRAM chips are large, rectangular arrays of memory cells with support logic for reading, writing data in the arrays and refresh circuitry to maintain the integrity of stored data [11]. According to storage oraganization, memory is hierarchically organized into rank, bank and array. DRAM organization has been described in detail in [10]. Fig.1 shows the organisation of a DRAM.

B. Row Buffer Management Policy in DRAM

In DRAM devices, arrays of sense amplifiers act as buffers that provide temporary data storage. Row buffer management policies manage the operation of sense amplifiers. In ordinary DRAM devices, two commonly used page policies are-

- Open-Page policy: The open-page row buffer management policy usually favors memory accesses to the same row of memory by keeping the sense amplifiers open and holding the data for ready access. Whenever a row of data is brought to the array of sense amplifiers in

978-1-5386-3032-7/17/$31.00 © 2017 IEEE

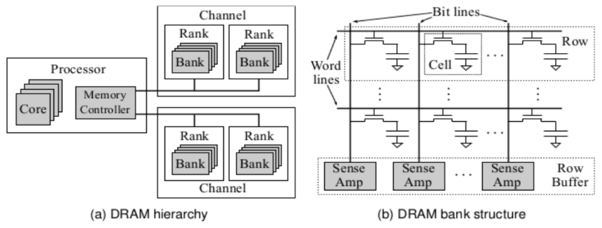

Fig. 1. Organisation of DRAM

a DRAM cell, different columns of the same row can be accessed again and again having only column access latency. But when a different row of the same bank needs to be accessed, the memory controller first precharges the DRAM array, activates the desired row and finally allows for column access. This policy works best for sequential memory accesses and performance increases by better exploiting spatial and temporal locality in memory.

- Close-Page policy: The close-page row buffer management policy works better when we have random memory accesses across different rows. This policy precharges the row after every memory access. So the bank is in Idle state after every row access avoiding the precharging overhead.

The state-of-the-art DRAM controllers mainly use open-page policy. As a result, some real time tasks get prioritised over other tasks in the waiting queue which may even lead to deadline miss of some of the real time tasks. In the next section, this problem has been addressed with the help of a motivating example.

III. Overview of the Problem

Given a set of real time tasks, each with a deadline, the problem is to schedule the tasks on the DRAM such that the fewest number of tasks miss their deadlines.

i^{th} task instance in a real time system, denoted by τ_i can be represented with the help of the following parameters -

$$\tau_i = (A_i, E_i, D_i, P_i)$$

where A_i denotes the arrival time of i^{th} task instance, E_i denotes worst case execution time of i^{th} task instance, D_i denotes deadline of i^{th} task instance, P_i denotes the time interval after which the next task instance arrives.

We present our problem with the help of a motivating example.

A. Motivating Example

Let us consider a simple DRAM with two banks, B0 and B1, each bank having a set of four rows. The bank mapping of the two banks are as follows -

- Bank B0 contains four sets of rows. Row 0 (addr 0-addr 3), Row 1 (addr 4-addr 7), Row 2 (addr 8-addr 11) and Row 3 (addr 12-addr 15).
- Bank B1 contains four sets of rows. Row 0 (addr 16-addr 19), Row 1 (addr 20-addr 23), Row 2 (addr 24-addr 27) and Row 3 (addr 28-addr 31).

We consider that our DRAM follows open page policy, i.e., memory requests which result in row hit will be allowed to access the DRAM. We consider two tasks T1 and T2 arriving at the same time instant and scheduled according to EDF at the processor. Each task consists of some CPU and Memory instructions. In this example, we have denoted the CPU instructions with <C> and the Memory instructions as <M addr>, which denotes a memory instruction at address "addr". We consider the task specifications given in Table I. Assume each CPU instruction takes 100 cycles to execute, each memory instruction takes 300 cycles to execute when there is a row miss and 200 cycles to execute when there is a row hit. Since tasks are scheduled according to EDF, T2 will be allowed to access the CPU earlier. A detailed picture of the waiting queue at the processor, the processor, the memory buffer, the memory and the status of T1 and T2 at different cycles is described in Table II. It can be inferred from this example, that due to address mapping policy, all the memory requests are mapped to Bank B1. Again, due to the open row policy, some of T1's memory requests although arrived earlier, could not get access to memory. As a result, T1 cannot complete its execution within its deadline.

TABLE I
Task Specifications

Task ID	Instructions	Bank Mapping	Deadline
T1	C		1800 cycles
	M 25	Bank1-Row2	
	M 20	Bank1-Row1	
	C		
	M 30	Bank1-Row3	
	M 28	Bank1-Row3	
T2	C		1500 cycles
	M 30	Bank1-Row3	
	M 29	Bank1-Row3	
	C		
	M 24	Bank1-Row2	

If T1's memory request is allowed to execute in Bank B0, then T1 can easily complete its execution within its deadline. But executing a memory request in some other memory bank incurs a lot of overheads. We propose a novel method of bank aware partitioning based on a cost function calculated locally taking into account all the overheads so that the tasks can complete execution within deadline. We discuss our proposal in the following subsection.

B. Detailed Methodology

We have implemented a two level scheduler, one at the processor and the other at the memory controller. We have applied EDF to schedule tasks at the processor, i.e., tasks whose deadline is earliest is placed at the front of the priority queue and are allowed to execute first. But as the memory requests arrive at the DRAM controller, they are not executed in the order they are scheduled. Open page policy prioritizes those memory requests which result in row buffer hit of tasks. We propose to keep a set of extra memory banks which will not participate in address mapping. These extra memory banks are also known as the reserved banks. We formulate a cost function based on some known task parameters and schedule tasks based on that cost function. Cost function, $Cost_{ij}$, is the cost to execute i^{th} instruction of j^{th} task. We define another variable, C_{ijk}, which is the remaining time after execution of i^{th} instruction of j^{th} task in k^{th} bank (where it is actually mapped) and is given by-

$$C_{ijk} = D_j - (t_{ik} + e_{iw} + e_{ik} + e_{rem})$$

TABLE II
A SNAPSHOT OF THE ENTIRE SYSTEM AT DIFFERENT CYCLES

Cycle	Waiting Queue	Processor	Memory Buffer	Memory	Status
0	T1	T2			
99	T1	T2			
100		T1		T2 <M 30> Bank1 Row 3	T2-Row Miss
199		T1		T2 <M 30> Bank1 Row3	
200			T1	T2 <M 30> Bank1 Row3	
399			T1	T2 <M 30> Bank1 Row3	
400			T1	T2 <M 29> Bank1 Row3	T2-Row Hit
599			T1	T2 <M 29> Bank1 Row3	
600		T2		T1 <M 25> Bank1 Row2	T1-Row Miss
899			T2	T1 <M 25> Bank1 Row2	
900			T1	T2 <M 24> Bank1 Row2	T2-Row Hit
1099			T1	T2 <M 24> Bank1 Row2	T2-complete execution
1100				T1 <M 20> Bank1 Row1	T1-Row Miss
1399				T1 <M 20> Bank1 Row1	
1400		T1			
1499		T1			
1500				T1 <M 30> Bank1 Row3	T1-Row Miss
1799				T1 <M 30> Bank1 Row3	
1800				T1 <M 28> Bank1 Row3	T1-Deadline Miss

Algorithm 1 Scheduling Policy at the Memory Controller

```
 1: function SchedMemReq($Task\ t$, Instruction $I$)   ▷ $t$ - Task, $I$ - $i^{th}$ memory instruction of task $t$
 2:     $t_{ik}$ - time to copy $i^{th}$ instruction in $k^{th}$ bank
 3:     $e_{iw}$ = maximum waiting time of $t$ due to tasks which are in the waiting queue ahead of $t$
 4:     $e_{ik}$ = execution time of $i^{th}$ instruction in $k^{th}$ bank in case of a row hit/miss
 5:     $e_{ik'}$ = execution time of $i^{th}$ instruction in its own bank in case of a row hit/miss
 6:     $e_{rem}$ = worst case execution time of all the remaining instructions
 7:     $C_{itk}$ = $t.deadline$ - ($t_{ik}$ + $e_{iw}$ + $e_{ik}$ + $e_{rem}$)   ▷ $C_{itk}$ denotes the total time remaining after executing $i^{th}$ instruction of task $t$ in $k^{th}$ bank
 8:     $C_{itk'}$ = $t.deadline$ - ($e_{iw}$ + $e_{ik'}$ + $e_{rem}$)   ▷ $C_{itk}$ denotes the total time remaining after executing $i^{th}$ instruction of task $t$ in its own bank
 9:     $Cost_{ij}$ = $C_{ijk}$ - $C_{ijk'}$   ▷ $Cost_{ijk}$ denotes the cost incurred in the process
10:     if $Cost_{ij} > 0$ then
11:         Schedule $t$ in its own bank
12:     else
13:         Schedule $t$ in some reserved banks
14:     end if
15: end function
```

where D_j : the deadline of j^{th} task,
t_{ik} : the time to copy the i^{th} instruction in k^{th} bank,
e_{iw} : the maximum waiting time of i^{th} task in the memory buffer due to presence of other tasks infront of the i^{th} instruction in the memory buffer,
e_{ik} : the execution time of i^{th} instruction in k^{th} bank if there is a row miss/hit,
e_{rem} : the worst case execution time required by the remaining instructions of j^{th} task.

Similarly, $C_{ijk'}$ is defined as the remaining time after execution of i^{th} instruction of j^{th} task in one of the extra memory banks or the reserved banks, say k'. Our aim is to execute the i^{th} instruction of j^{th} task instance in the bank that allows us to have greater remaining time so that there is a lesser chance of having a deadline miss. This is determined with the help of the cost function C_{ij} which is given by-

$$Cost_{ij} = C_{ijk} - C_{ijk'}$$

If $Cost_{ij}$ is greater than 0, we allow the task to execute in its own bank, otherwise we schedule it to execute in any one of the reserved set of banks. Algorithm 1 explains our proposed method.

IV. Implementation and Results

Our end to end implementation consists of two main parts. The first part consists of trace generation from benchmark programs of the Malardalen WCET benchmark [5]. We executed our benchmark programs on an X-86 architectural platform to generate X-86 instructions. Table III gives a description of the benchmark programs in detail. In our implementation, the arrival time (A) of each task is randomly generated. The execution time of a task(E) is actually the sum of execution time of all the instructions in the worst case. The period of each task (P) is randomly generated with the condition that it should be greater than the worst case execution time. We have considered the deadline of a task instance (D) to be equal to the period after which the next task instance arrives. Thus at a particular time instant only one task instance of a task actually executes in the system. The final part deals with the end to end simulation. We have considered a uni-processor system, where tasks are scheduled according to EDF at the processor. Whenever a memory request arrives, a context switch occurs at the CPU and the memory request waits in the memory buffer. The memory controller then selects a request from the memory buffer based on our proposed algorithm and executes a task in memory. Our results were generated by varying the number of banks and the size of row in each bank. In all the cases, we have assumed that we have a set of two extra banks which do not participate in address mapping. We have implemented our own end-to-end simulator and have compared our results with existing state-of-the-art DRAM controllers. Table IV shows a comparative analysis of existing DRAM controllers following

TABLE III
LIST OF BENCHMARK PROGRAMS USED AS REAL TIME TASKS

Benchmark programs	Total instructions	Memory Reads	Memory Writes
bs	179	32	32
cnt	295	52	43
duff	257	47	64
fac	164	24	24
fibcall	163	27	28
insertsort	189	36	30
lcdnum	198	26	23
ns	245	31	40
prime	226	28	41
res	179	91	88

TABLE IV
RESULTS ON SOME OF THE DATASETS

Dataset Number	Number of cycles	Number of Banks	Rowsize	Number of Task Miss (open-page policy)	Number of Task Miss (our method)
Dataset I (fibcall.c, lcdnum.c, fac.c)	10,000	10 banks	64B	1	1
	20,000			3	2
	30,000			4	3
	40,000			6	4
	50,000			8	5
	60,000			9	6
	70,000			11	7
	10,000	10 banks	32B	2	1
	20,000			4	2
	30,000			5	3
	40,000			8	5
	50,000			11	6
	60,000			12	7
	70,000			14	9
Dataset II (bs.c, count.c duff.c)	10,000	19 banks	64B	3	0
	20,000			6	1
	30,000			9	1
	40,000			14	4
	10,000	19 banks	32B	3	2
	20,000			5	3
	30,000			8	6
	40,000			13	11
Dataset III (prime.c, ns.c, insertsort.c)	10,000	15 banks	64B	2	1
	20,000			4	1
	30,000			5	2
	40,000			8	3
	10,000	15 banks	32B	3	2
	20,000			5	3
	30,000			6	4
	40,000			9	6
Dataset IV (lcdnum.c, insertsort.c, fibcall.c)	10,000	12 banks	64B	2	1
	20,000			5	2
	30,000			7	4
	40,000			12	4
	10,000	12 banks	32B	3	2
	20,000			6	4
	30,000			9	6
	40,000			14	11

open-page policy vs our proposed DRAM controller. We have generated some dataset from the benchmark programs of the Malardalen WCET benchmark. We have run each dataset upto certain number of cycles by varying the number of memory banks. We have also varied the size of each row in a memory bank. In Table IV, column I shows the programs in each dataset, column II shows the number of cycles upto which we ran our experiments, column III shows the number of banks in the memory module we worked on, column IV shows the size of row in each bank, column V shows the number of misses in DRAM with open-page policy and column VI shows the number of misses we achieved by using our scheduling algorithm at the DRAM controller.

V. CONCLUSION

In this paper, we present the problem of open page policy in state-of-the-art DRAM controllers and the problem in using these DRAMs in real time systems. In real time systems, though the real time tasks are scheduled based on some priority based scheduling algorithms at the processor, the tasks are not executed in the same order in the memory due to prioritization of some other tasks at the DRAM controller. This becomes a bottleneck to guarantee completion of all real time tasks within their deadline. Finally, we proposed a scheduling algorithm at the DRAM which schedules the tasks based on a cost function evaluated on the basis of some local task parameters. We implemented an end-to-end simulator for the same and generated our results on the Malardalen WCET benchmark programs.

REFERENCES

[1] B. Akesson and K. Goossens. Architectures and modeling of predictable memory controllers for improved system integration. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –6, March 2011.

[2] B. Akesson, L. Steffens, E. Strooisma, and K. Goossens. Real-time scheduling using credit-controlled static-priority arbitration. In *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA'08. 14th IEEE International Conference on*, pages 3–14. IEEE, 2008.

[3] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13, pages 17:1–17:15, Piscataway, NJ, USA, 2013. IEEE Press.

[4] S. Goossens, J. Kuijsten, B. Akesson, and K. Goossens. A reconfigurable real-time SDRAM controller for mixed time-criticality systems. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, pages 1–10, September 2013.

[5] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The mälardalen wcet benchmarks: Past, present and future. In *OASIcs-OpenAccess Series in Informatics*, volume 15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[6] M. Hassan. Predictable shared memory resources for multi-core real-time systems. 2017.

[7] H. Kim, D. De Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar. Bounding memory interference delay in cots-based multi-core systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*, pages 145–154. IEEE, 2014.

[8] M. Paolieri, E. Quiñones, and F. J. Cazorla. Timing effects of ddr memory systems in hard real-time multicore architectures: Issues and solutions. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(1s):64, 2013.

[9] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee. Pret dram controller: Bank privatization for predictability and temporal isolation. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proceedings of the 9th International Conference on*, pages 99–108. IEEE, 2011.

[10] David Tawei Wang. *Modern Dram Memory Systems: Performance Analysis and Scheduling Algorithm.* PhD thesis, College Park, MD, USA, 2005. AAI3178628.

[11] Wikipedia. Dynamic random-access memory — wikipedia, the free encyclopedia, 2017. [Online; accessed 10-April-2017].

[12] Wikipedia. Earliest deadline first scheduling — wikipedia, the free encyclopedia, 2017. [Online; accessed 11-April-2017].

[13] Wikipedia. Rate-monotonic scheduling — wikipedia, the free encyclopedia, 2017. [Online; accessed 11-April-2017].

[14] H. Yun, R. Mancuso, Z. Wu, and R. Pellizzoni. Palloc: Dram bank-aware memory allocator for performance isolation on multicore platforms. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*, pages 155–166. IEEE, 2014.