Georgia Institute of Technology

A Prioritized Cache for Multi-tasking Real-time Systems

Yudong Tan* and Vincent J. Mooney III+

{ydtan, mooney}@ece.gatech.edu

+Assistant Professor, *School of Electrical & Computer Eng.

+Adjunct Assistant Professor, College of Computing

Georgia Institute of Technology

Atlanta, Georgia, USA

SASIMI Workshop, April 3-4, 2003

Outline

- Motivation
- Previous Work
- Cache Column Assignment Policy
- Experiments
- Conclusion
- Future Work

Motivation

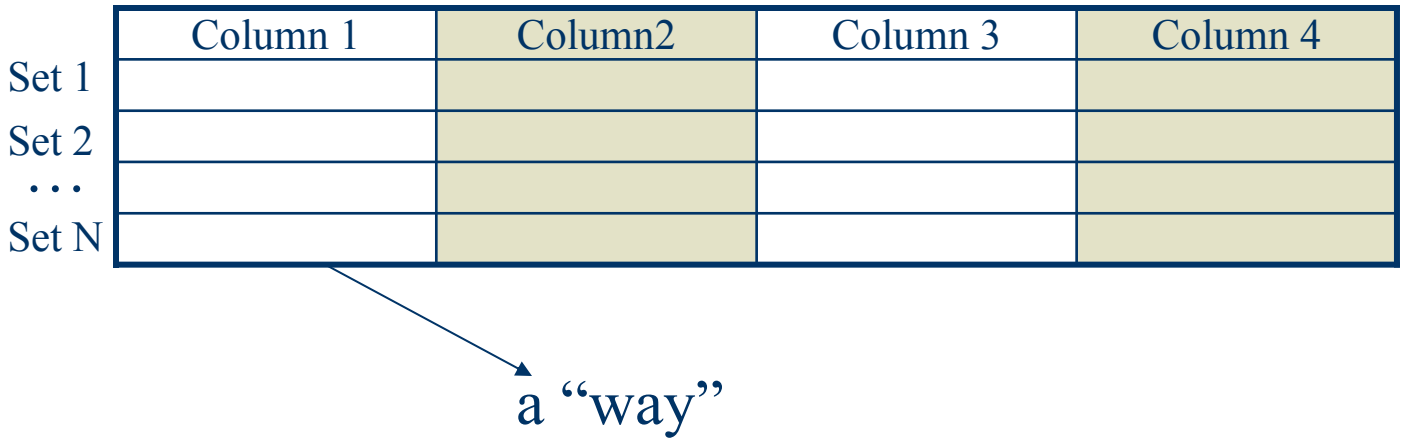
- Reduce the Worst Case Cache Miss Rate (WCCMR)
- Tighten the bounds of Worst Case Execution Time (WCET) estimates
- Guarantee the performance of high priority tasks in a multi-tasking real-time system

Previous Work

- Column Cache [Chiou D., Rudolph L etc., 2000]
  - Static Cache Allocation
- Data-replace-controlled Cache [N. Maki, etc., 1999]
  - Very fine-grain cache control
- OS-controlled Cache [J. Liedtke and H. Härtig, 1997]
- SMART (Strategic Memory Allocation for Real-Time) Cache [David B. Kirk, 1989]
  - Allocate Cache based on CPU utilization
- Cache Designs for Hard Real-Time Systems [S. M. Shahrier and J. C. Liu, 1997]
- Static timing analysis methods
  - Cinderella [Steven Li and S. Malik, 1997]
  - Symta, SPI [Fabian Wolf and Rolf Ernst, 1999]

Cache Columns

- 4-way set-associative cache

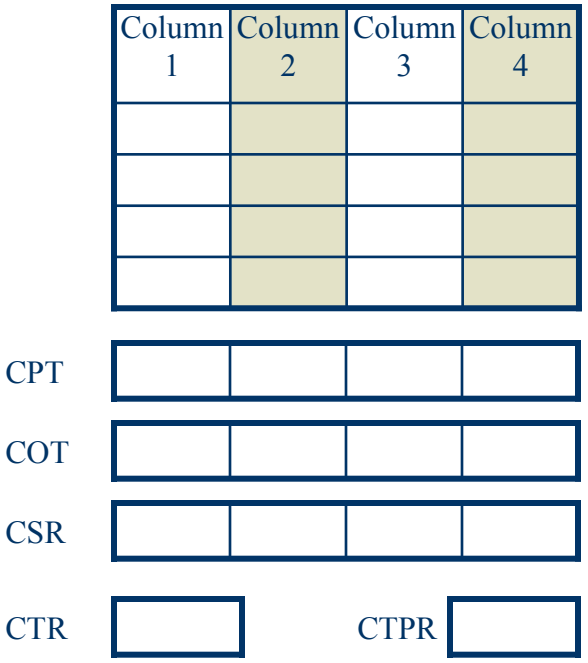


Approach

- Assumption
  - Each task has a unique priority.
- Owner of a Column
  - The task which is using the column
- Priority of a Column
  - The priority of the owner

Approach (cont.)

- Registers in a *k*-way set associative prioritized cache
  - Column Priority Table (CPT) – *kN* bits
  - Column Owner Table (COT) – *kM* bits
  - Column Status Reg. (CSR) – *k* bits
  - Current Task Reg. (CTR) – *M* bits
  - Current Task's Priority Reg. (CTPR) – *N* bits
- For a *k*-way set associative cache
  - *(k+1)(N+M)+k* bits in all registers.
  - Up to 2^N priorities and 2^M tasks are supported.
  - For *k=4,M=N=8,*
    - 84 bits in total in CPT, COT, CSR, CTR and CTPR registers,
    - 256 levels of priorities and 256 tasks.



Example

- Task 1
  - Has a priority of 7
  - Has an ID of 1
  - Owns Column 1
- Task 2
  - Has a priority of 4
  - Has an ID of 2
  - Owns Column 3
- Column 2 is not in use.
- Column 4 is shared.
- Task 2 is running.
- The lowest priority is *0xff*.

	Column 1	Column 2	Column 3	Column 4
CPT	7	0xff	4	0xff
COT	1	0	2	0
CSR	0	0	0	1

CTR: 2 CTPR: 4

Column Assignment Policy

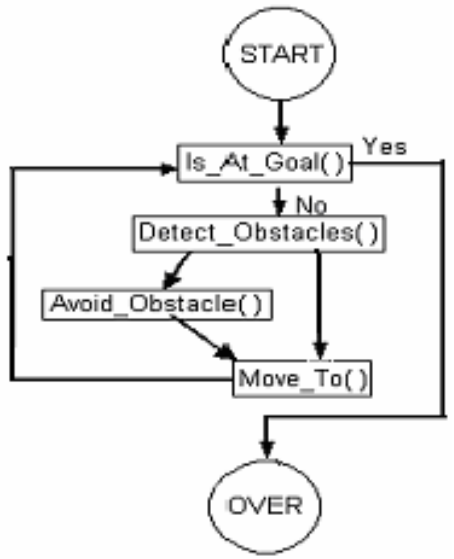
- Each cache column has a priority which is the same as the task that is using the column.
- A column can be used by a task which has a higher priority than the column.
- When a block is missing, the cache lines in the columns owed by the current task are considered, then the columns with lower priorities than the priority of the current task are considered.
- The priorities of columns used by a task are set to the lowest priority when the task is completed.

Shared Column

- Reason -- Low priority tasks may not have cache columns to use.
- Shared Column -- When a column is shared, its priority is always fixed to the lowest priority. It is not owned by any task.
- CSR is used to indicate if a column is shared or not.
- If CSR is set for all columns, the prioritized cache is the same as a normal set-associative cache.

Example

- Two applications
  - MPEG decoder (MPEG)
  - A Mobile Robot Control Application (MR)
- A 4-way set-associative cache



CDFG of MR

Example (cont.)

- Initialization
  - 256 priorities (0-255)
  - IDLE task with an ID of 0 and a priority of 255
- MPEG and MR run alternatively in a co-routine like fashion
  - Priority of MPEG 2, Task ID of MPEG 1
  - Priority of MR 1, Task ID of MR 2
  - MPEG is started first and completed earlier than MR.
- Column 4 is set as “shared”.

Column 1	Column2	Column 3	Column 4

CPT	0xff	0xff	0xff	0xff
COT	0	0	0	0
CSR	0	0	0	1

CTR	0	CTPR	0xff

Example (cont.)

- MPEG runs.
  - Column 1 is assigned to MPEG.

Column 1	Column2	Column 3	Column 4

CPT	2	0xff	0xff	0xff
COT	1	0	0	0
CSR	0	0	0	1

CTR 1 CTPR 2

Example (cont.)

- MPEG runs.
  - Column 1 is assigned to MPEG.
  - Column 2 is assigned to MPEG.

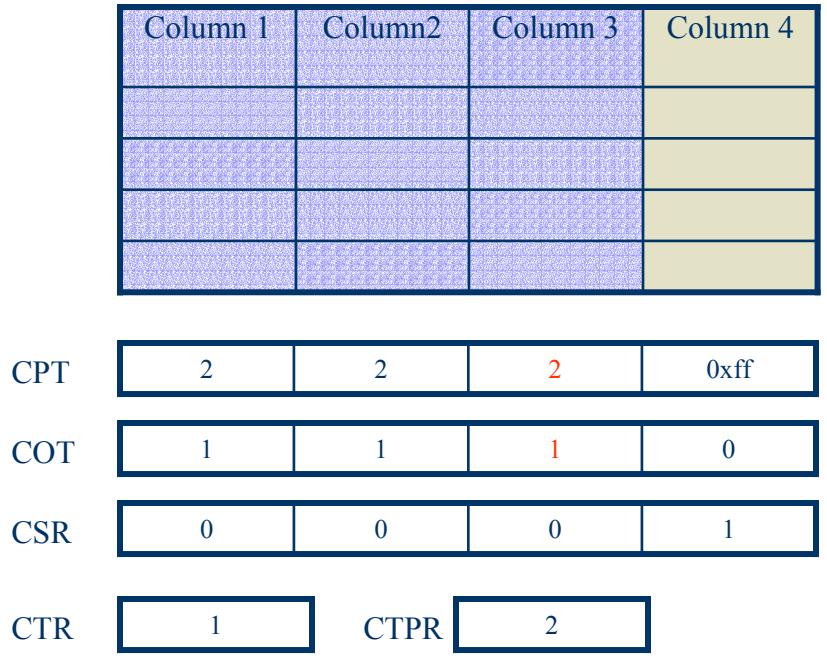
Column 1	Column2	Column 3	Column 4

CPT	2	2	0xff	0xff
COT	1	1	0	0
CSR	0	0	0	1

CTR 1 CTPR 2

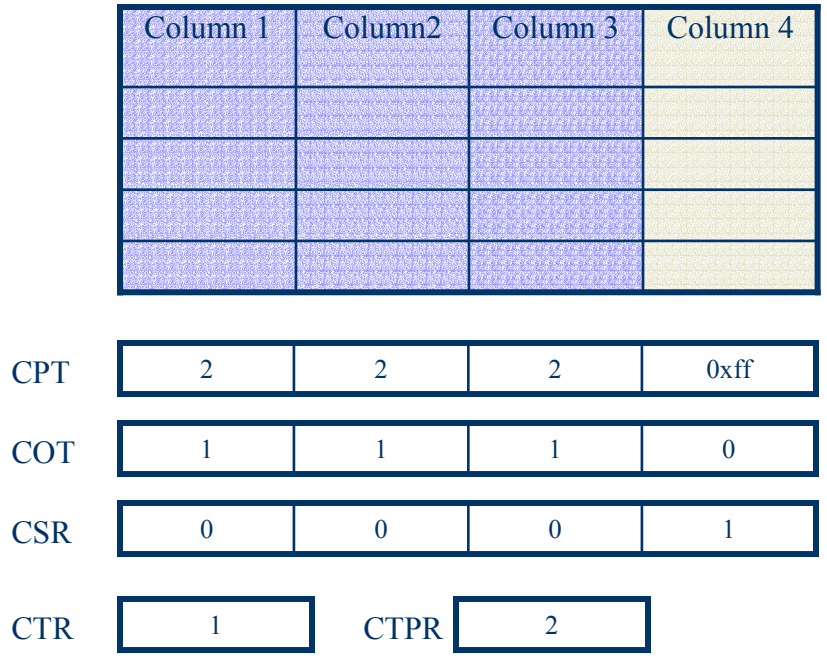
Example (cont.)

- MPEG runs.
  - Column 1 is assigned to MPEG.
  - Column 2 is assigned to MPEG.
  - Column 3 is assigned to MPEG.



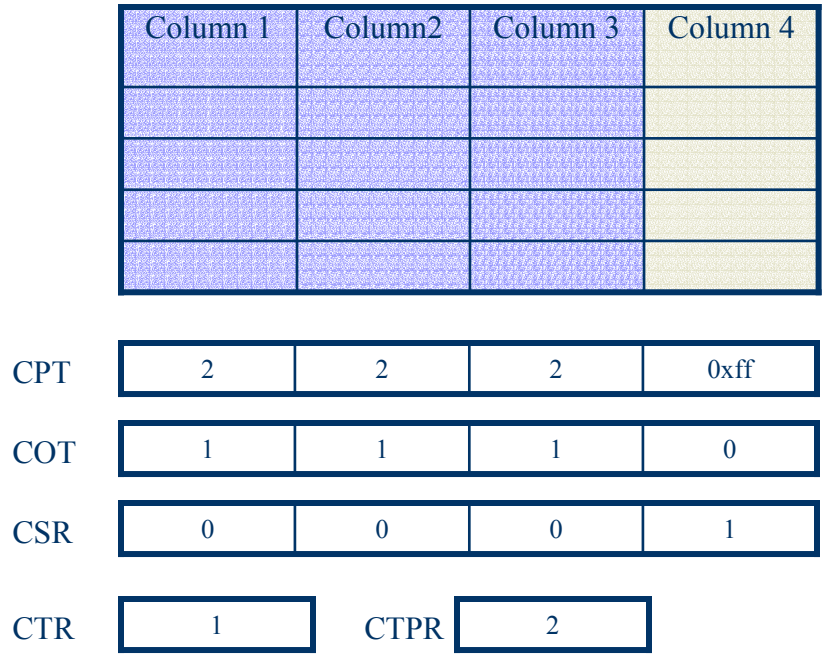
Example (cont.)

- MPEG runs.
  - Column 1 is assigned to MPEG.
  - Column 2 is assigned to MPEG.
  - Column 3 is assigned to MPEG.
  - Column 4 is used by MPEG.



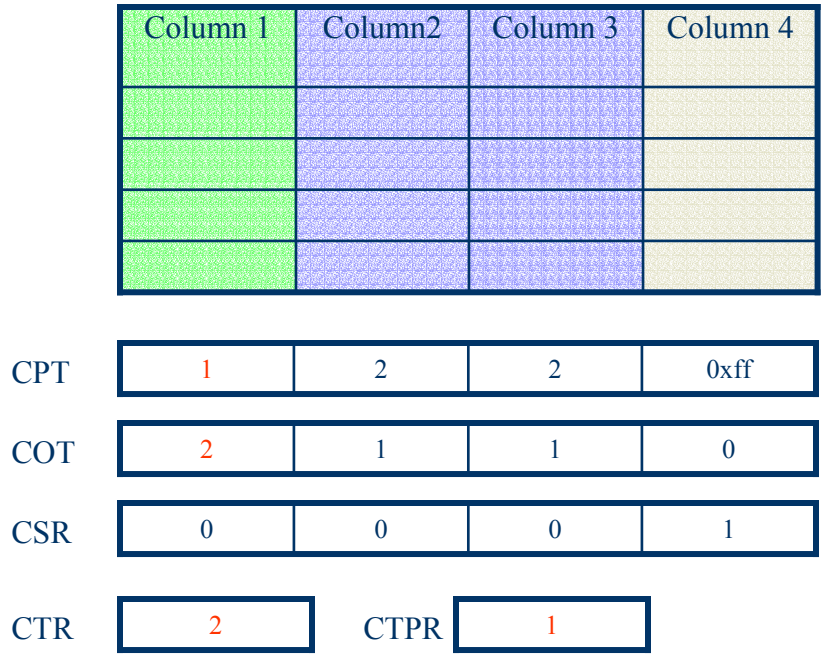
Example (cont.)

- MR runs.



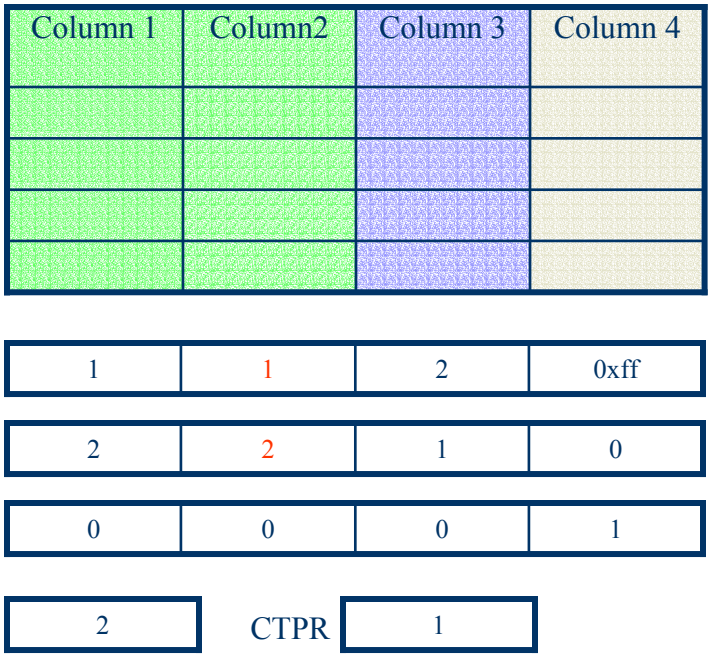
Example (cont.)

- MR runs.
  - Column 1 is assigned to MR.



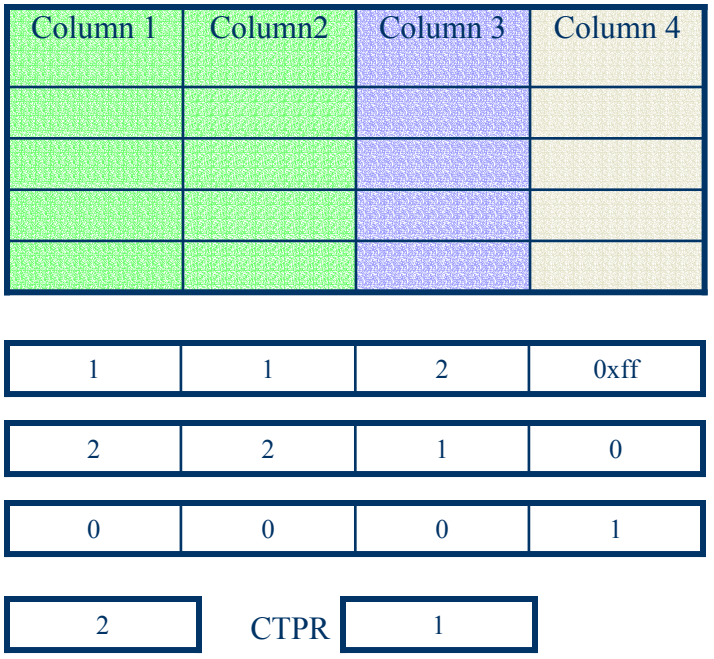
Example (cont.)

- MR runs.
  - Column 1 is assigned to MR.
  - Column 2 is assigned to MR.



Example (cont.)

- MPEG is over.
  - Column 3 is released.



Example (cont.)

- MPEG is over.
  - Column 3 is released.

Column 1	Column2	Column 3	Column 4

CPT	1	1	0xff	0xff
COT	2	2	0	0
CSR	0	0	0	1

CTR 1 CTPR 2

Example (cont.)

- MR is over.
  - Column 1 and Column 2 are released.

Column 1	Column2	Column 3	Column 4

CPT	1	1	0xff	0xff
COT	2	2	0	0
CSR	0	0	0	1

CTR 2 CTPR 1

Example (cont.)

- MR is over.
  - Column 1 and Column 2 are released.

Column 1	Column2	Column 3	Column 4

CPT	0xff	0xff	0xff	0xff
COT	0	0	0	0
CSR	0	0	0	1

CTR: 0 CTPR: 0xff

API

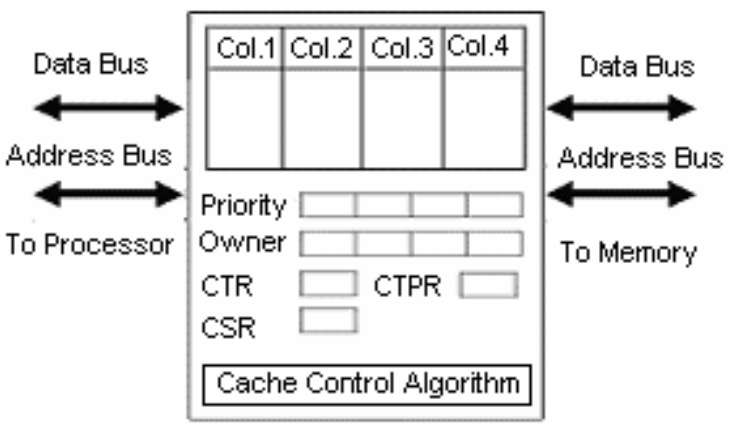
- *Set_tid_pri(tid,pri)*
- *Set_column_pri(col,pri)*
- *Release_column(tid)*
- *Set_ column_shared(col)*
- *Free_column_shared(col)*
- APIs can be embedded in OS so that users don't need to know the low-level details.
  - *Set_tid_pri()* can be embedded in context switch function.
  - *Release_column()* can be embedded in task destruction function.
  - Shared columns are set by default. Users can change them when it is necessary.

Example of using APIs

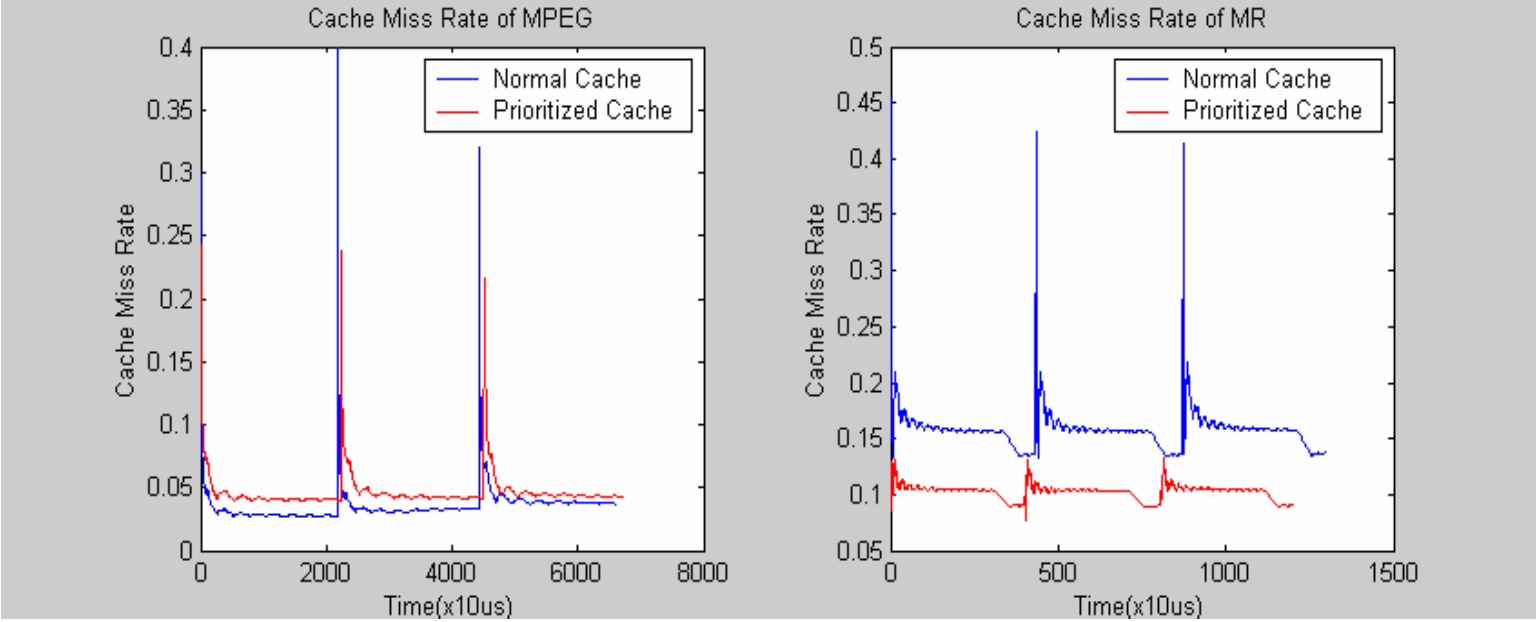
```
1. Set_column_shared(3);
2. while(!MPEG_over()&&!MR_over()){
3.    Set_tid_pri(MPEG_ID,2);
4.    MPEG_decode_one_slice();
5.    Set_tid_pri(MR_ID,1);
6.    MR_move_one_step();
7.    if(MPEG_over())
8.       Release_column(MPEG_ID);
9.    if(MR_over())
10.      Release_column(MR_ID);  }
```

Line	CTR	CTPR	CSR	Priority of Columns				Owner of Columns			
				0	1	2	3	0	1	2	3
Initial	0	0	0000	ff	ff	ff	ff	0	0	0	0
1	0	0	0001	ff	ff	ff	ff	0	0	0	0
3	1	2	0001	ff	ff	ff	ff	0	0	0	0
4	1	2	0001	2	2	2	ff	1	1	1	0
5	2	1	0001	2	2	2	ff	1	1	1	0
6	2	1	0001	1	1	2	ff	2	2	1	0
8	2	1	0001	1	1	ff	ff	2	2	0	0
10	2	1	0001	ff	ff	ff	ff	0	0	0	0

Experiment



Experimental Results



Increase of Area in Cache Controller < 1%
Increase of MPEG execution time = 3%
WCCMR of MR reduced from 45% to 13% , a 3X reduction

Estimate of MR WCET

	WCCMR	WCET estimates (clock cycles)
Normal Cache	0.45	731214
Prioritized Cache	0.13	425347

A 41% reduction in WCET estimate

Actual Execution time in Simulation = **409286**

Ideal CPI= 1.26

Cache missing penalty, P=4

# of instructions, M = 238959

Cache miss rate, *r*

$$WCET = M \times CPI_{ideal} + M \times r \times P$$

Conclusion

- Prioritized Cache can reduce the WCCMR.
- With a more stable WCCMR, we can estimate WCET with tighter bounds.
- Low-level details are transparent to users.
- Only need trivial modifications in OS.

Future Work

- Improve the method of setting shared columns to handle a group of tasks with the same priorities.
- Build an analytical model to estimate the WCET/BCET of tasks in a system using prioritized caches.
- Partition the cache at smaller granularities.