MARACAS: A Real-Time Multicore VCPU Scheduling Framework

Ying Ye, Richard West, Jingyi Zhang, Zhuoqun Cheng

Computer Science Department
Boston University
Overview

1. Introduction
2. Quest RTOS
3. Background Scheduling
4. Memory-Aware Scheduling
5. Multicore VCPU Scheduling
6. Evaluation
7. Conclusion
Motivation

- Multicore platforms are gaining popularity in embedded and real-time systems
  - concurrent workload support
  - less circuit area
  - lower power consumption
  - lower cost
Motivation

- Multicore platforms are gaining popularity in embedded and real-time systems
  - concurrent workload support
  - less circuit area
  - lower power consumption
  - lower cost

- Complex on-chip memory hierarchies pose significant challenges for applications with real-time requirements
Motivation

- Shared cache contention:
  - page coloring
  - hardware cache partitioning (Intel CAT)
  - static VS dynamic
Motivation

- **Shared cache contention:**
  - page coloring
  - hardware cache partitioning (Intel CAT)
  - static VS dynamic

- **Memory bus contention:**
  - bank-aware memory management
  - memory throttling
We proposed the use of foreground (reservation) + background (surplus) scheduling model
- improves application performance
- effectively reduces resource contention
- well-integrated with real-time scheduling algorithms
Contribution

- We proposed the use of foreground (reservation) + background (surplus) scheduling model
  - improves application performance
  - effectively reduces resource contention
  - well-integrated with real-time scheduling algorithms

- We proposed a new bus monitoring metric that accurately detects traffic
Application

- Imprecise computation/Numeric integration
  - MPEG video decoding: mandatory to process I-frames, optional to process B- and P-frames to improve frame rate
Application

- Imprecise computation/Numeric integration
  - MPEG video decoding: mandatory to process I-frames, optional to process B- and P-frames to improve frame rate

- Mixed-criticality systems running performance-demanding applications
  - machine learning
  - computer vision
Quest RTOS

VCPU model \((C, T)\) in Quest RTOS
- \(C\): Capacity
- \(T\): Period
**Quest RTOS**

- VCPU model \((C, T)\) in Quest RTOS
  - \(C\): Capacity
  - \(T\): Period

- Partitioned scheduling using RMS
MARACAS
Ying Ye, Richard West, Jingyi Zhang, Zhuoqun Cheng

Introduction

Quest RTOS

Background
Scheduling

Memory-Aware Scheduling

Multicore VCPU Scheduling

Evaluation
Conclusion

VCPU model (C, T) in Quest RTOS
- C: Capacity
- T: Period

Partitioned scheduling using RMS

Schedulability test
\[ \sum_{i=1}^{n} \left( \frac{C_i}{T_i} \right) \leq n(\sqrt{2} - 1) \]
**Background Scheduling**

- VCPU enters background mode upon depleting its budget (C)

![Diagram showing foreground and background scheduling]

---

**Background Scheduling**

- VCPU enters background mode upon depleting its budget (C)

![Diagram showing foreground and background scheduling]

- Core enters background mode when all VCPUs are in background mode.

**Background CPU Time (BGT):** time a VCPU runs when core is in background mode.

**Background scheduling:** schedule VCPUs when core is in background mode with a fair share of BGT amongst VCPUs on core.
Background Scheduling

- VCPU enters background mode upon depleting its budget (C)

- Core enters background mode when all VCPUs are in background mode

\[ \begin{align*}
\text{Foreground} & \quad \text{Background} \\
C & \quad T - C \\
& \quad T
\end{align*} \]
Background Scheduling

- VCPU enters background mode upon depleting its budget (C)

- Core enters background mode when all VCPUs are in background mode

- Background CPU Time (BGT): time a VCPU runs when core in background mode
Background Scheduling

- VCPU enters background mode upon depleting its budget (C)

- Core enters background mode when all VCPUs are in background mode

- Background CPU Time (BGT): time a VCPU runs when core in background mode

- Background scheduling: schedule VCPUs when core is in background mode
  - fair share of BGT amongst VCPUs on core
DRAM structure

- MARACAS
- Ying Ye,
- Richard West,
- Jingyi Zhang,
- Zhuoqun
- Cheng

Introduction
Quest RTOS
Background
Scheduling
Memory-Aware
Scheduling
Multicore
VCPU
Scheduling
Evaluation
Conclusion
Memory-Aware Scheduling

- Prior work [MemGuard] uses "Rate Metric": number of DRAM accesses over a certain period
Memory-Aware Scheduling

- Prior work [MemGuard] uses "Rate Metric": number of DRAM accesses over a certain period
  - Bank-level parallelism
Memory-Aware Scheduling

- Prior work [MemGuard] uses "Rate Metric": number of DRAM accesses over a certain period
  - Bank-level parallelism
  - Row buffers
Memory-Aware Scheduling

- Prior work [MemGuard] uses "Rate Metric": number of DRAM accesses over a certain period
  - Bank-level parallelism
  - Row buffers
  - Sync Effect
Sync Effect

Each task reduces its access rate by a factor of $(T-t)/T$.
Sync Effect

- Each task reduces its access rate by a factor of \((T-t)/T\)
- Contention in \([0, t]\) remains the same
**Latency Metric**

- $\text{requests} = 3$, $\text{occupancy} = 10$
Latency Metric

<table>
<thead>
<tr>
<th>Time (cycles)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue Length</td>
<td>r_1</td>
<td>r_1</td>
<td>r_2</td>
<td>r_2</td>
<td>r_2</td>
<td>r_3</td>
</tr>
</tbody>
</table>

- \( \text{requests} = 3, \ \text{occupancy} = 10 \)
- \( \text{latency} = \frac{10}{3} = 3.3 \)
Latency Metric

- **UNC_ARB_TRK_REQUEST.ALL (requests)**: counts all memory requests going to the memory controller request queue

- **UNC_ARB_TRK_OCCUPANCY.ALL (occupancy)**: counts cycles weighted by the number of pending requests in the queue
Latency Metric

- UNC_ARB_TRK_REQUEST.ALL (requests): counts all memory requests going to the memory controller request queue

- UNC_ARB_TRK_OCCUPANCY.ALL (occupancy): counts cycles weighted by the number of pending requests in the queue

Average latency: 
\[ \text{latency} = \frac{\text{occupancy}}{\text{requests}} \]
Memory Throttling

- When core gets throttled, background scheduling is disabled
Memory Throttling

- When core gets throttled, background scheduling is disabled

- Latency threshold: MAX_MEM_LAT
  
  \[
  \text{if } \text{latency} \geq \text{MAX\_MEM\_LAT} \text{ then} \quad \text{num\_throttle} \quad ++
  \]
Memory Throttling

- When core gets throttled, background scheduling is disabled

- Latency threshold: \( \text{MAX\_MEM\_LAT} \)
  
  \[
  \text{if } \text{latency} \geq \text{MAX\_MEM\_LAT} \text{ then } \num_{throttle} + +
  \]

- Proportional throttling
  - Every core is throttled at some point
  - Throttled time proportional to core’s DRAM access rate
Predictable Migration

MARACAS
Ying Ye, Richard West, Jingyi Zhang, Zhuoqun Cheng

Introduction
Quest RTOS
Background Scheduling
Memory-Aware Scheduling
Multicore VCPU Scheduling
Evaluation
Conclusion
Predictable Migration

- Run migration thread with highest priority on each core: pushing local VCPUs to other cores (starts from highest utilization ones)
Predictable Migration

- Run migration thread with highest priority on each core: pushing local VCPUs to other cores (starts from highest utilization ones)

- Only one migration thread active during a migration period
Predictable Migration

- Run migration thread with highest priority on each core: pushing local VCPUs to other cores (starts from highest utilization ones)

- Only one migration thread active during a migration period

- Its execution of its entire capacity $C$ does not lead to any other local VCPUs missing their deadlines
Predictable Migration

- Run migration thread with highest priority on each core: pushing local VCPUs to other cores (starts from highest utilization ones)

- Only one migration thread active during a migration period

- Its execution of its entire capacity $C$ does not lead to any other local VCPUs missing their deadlines

- Constraint on $C$:

$$C \geq 2 \times E_{lock} + E_{struct}$$
For every core, define Slack-Per-VCPU (SPV):

$$SPV = 1 - \frac{1}{n} \sum_{i=1}^{n} \left( \frac{C_i}{T_i} \right)$$
For every core, define Slack-Per-VCPU (SPV):

$$SPV = \frac{1 - \sum_{i=1}^{n} (C_i/T_i)}{n}$$

- Core
  - 10%
  - 30%
  - 60%

VCPU1  VCPU2  Slack

$$SPV = \frac{1 - (10\% + 30\%)}{2} = 30\%$$
VCPU Load Balancing

- Balance Background CPU Time (BGT) used by every VCPU across cores: equalize SPVs of all cores
### VCPU Load Balancing

- Balance Background CPU Time (BGT) used by every VCPU across cores: equalize SPVs of all cores
  - **BGT** fair sharing
VCPU Load Balancing

- Balance Background CPU Time ($BGT$) used by every VCPU across cores: equalize SPVs of all cores
  - $BGT$ fair sharing
  - balanced memory throttling capability on each core
**VCPU Load Balancing**

- Balance Background CPU Time (**BGT**) used by every VCPU across cores: equalize **SPVs** of all cores
  - **BGT** fair sharing
  - balanced memory throttling capability on each core

![Diagram showing load balancing](image)
Cache-Aware Scheduling

- Static cache partitioning amongst cores
  - page coloring
Cache-Aware Scheduling

- Static cache partitioning amongst cores
  - page coloring

- New API:
  
  ```
  bool vcpu_create(uint C, uint T, uint cache);
  ```
Cache-Aware Scheduling

- Static cache partitioning amongst cores
  - page coloring

- New API:
  ```
  bool vcpu_create(uint C, uint T, uint cache);
  ```

- Extension of VCPU Load Balancing:
  destination core meets the cache requirement
MARACAS running on the following hardware platform:

<table>
<thead>
<tr>
<th></th>
<th>Intel Core i5-2500k quad-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td></td>
</tr>
<tr>
<td>Caches</td>
<td>6MB L3 cache, 12-way set associative, 4 cache slices</td>
</tr>
<tr>
<td>Memory</td>
<td>8GB 1333MHz DDR3, 1 channel, 2 ranks, 8KB row buffers</td>
</tr>
</tbody>
</table>
Rate VS Latency

- Micro-benchmark m_jump:
  
  ```
  byte array[6M];
  for (uint32 j = 0; j < 8K; j += 64)
      for (uint32 i = j; i < 6M; i += 8K)
          <Variable delay added here>
          (uint32)array[i] = i;
  ```
Rate VS Latency

- Micro-benchmark m_jump:
  ```c
  byte array[6M];
  for (uint32 j = 0; j < 8K; j += 64)
      for (uint32 i = j; i < 6M; i += 8K)
          // Variable delay added here
          (uint32)array[i] = i;
  ```

- Three m_jump (task 1, 2, 3) running on separate cores without memory throttling, utilization (C/T) 50%
Rate VS Latency

- Micro-benchmark `m_jump`:
  ```c
  byte array[6M];
  for (uint32 j = 0; j < 8K; j += 64)
    for (uint32 i = j; i < 6M; i += 8K)
      (uint32)array[i] = i;
  ``

- Three `m_jump` (task 1,2,3) running on separate cores without memory throttling, utilization (C/T) 50%

- Each run, insert a different time delay in task1 and task2, task3 has no delay
Rate VS Latency

- Record the total memory bus traffic, average memory request latency and task3’s instructions retired in foreground mode:
Record the total memory bus traffic, average memory request latency and task3’s instructions retired in foreground mode:

<table>
<thead>
<tr>
<th></th>
<th>Bus Traffic (GB)</th>
<th>Latency</th>
<th>task3 Instructions Retired ($\times 10^8$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1128</td>
<td>228</td>
<td>249</td>
</tr>
<tr>
<td>M</td>
<td>1049</td>
<td>183</td>
<td>304</td>
</tr>
<tr>
<td>L</td>
<td>976</td>
<td>157</td>
<td>357</td>
</tr>
</tbody>
</table>
Rate VS Latency

- Record the total memory bus traffic, average memory request latency and task3’s instructions retired in foreground mode:

<table>
<thead>
<tr>
<th></th>
<th>Bus Traffic (GB)</th>
<th>Latency</th>
<th>task3 Instructions Retired (×10^8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1128</td>
<td>228</td>
<td>249</td>
</tr>
<tr>
<td>M</td>
<td>1049</td>
<td>183</td>
<td>304</td>
</tr>
<tr>
<td>L</td>
<td>976</td>
<td>157</td>
<td>357</td>
</tr>
</tbody>
</table>

- Setting comparable thresholds:
  - rate-based: derived from Bus Traffic (1128/time)
  - latency-based: from Latency (228)
Rate VS Latency

- Record the total memory bus traffic, average memory request latency and task3’s instructions retired in foreground mode:

<table>
<thead>
<tr>
<th></th>
<th>Bus Traffic (GB)</th>
<th>Latency</th>
<th>task3 Instructions Retired ($\times 10^8$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1128</td>
<td>228</td>
<td>249</td>
</tr>
<tr>
<td>M</td>
<td>1049</td>
<td>183</td>
<td>304</td>
</tr>
<tr>
<td>L</td>
<td>976</td>
<td>157</td>
<td>357</td>
</tr>
</tbody>
</table>

- Setting comparable thresholds:
  - rate-based: derived from Bus Traffic (1128/time)
  - latency-based: from Latency (228)

- Last column serves as reference, showing the expected performance of task3 using the corresponding thresholds
Repeat experiment with memory throttling enabled and fixed delay for task1/task2
Repeat experiment with memory throttling enabled and fixed delay for task1/task2
Conclusion

- MARACAS uses background time to improve task performance; when memory bus is contended, it gets disabled through throttling.
Conclusion

- MARACAS uses background time to improve task performance; when memory bus is contended, it gets disabled through throttling.

- MARACAS uses a latency metric to trigger throttling, outperforming prior rate-based approach.
Conclusion

- MARACAS uses background time to improve task performance; when memory bus is contended, it gets disabled through throttling.

- MARACAS uses a latency metric to trigger throttling, outperforming prior rate-based approach.

- MARACAS fairly distributes background time across cores, for both fairness and better throttling.