vMPCP: A Synchronization Framework for Multi-Core Virtual Machines

Hyoseung Kim*  
hyoseung@cmu.edu

Shige Wang†  
shige.wang@gm.com

Raj Rajkumar *  
raj@ece.cmu.edu

* Carnegie Mellon University
† General Motors R&D
Benefits of Multi-Core Processors

• Multi-core CPUs for embedded real-time systems
  • Automotive:
    – Freescale i.MX6 4-core CPU
    – NVIDIA Tegra K1 platform
  • Avionics and defense:
    – Rugged Intel i7 single board computers
    – Freescale P4080 8-core CPU

• Consolidation of real-time applications onto a single multi-core CPU
  – Reduces the number of CPUs and wiring harnesses among them
  – Leads to a significant reduction in space and power requirements
Virtualization of Real-Time Systems

• Barrier to consolidation
  – Each app. could have been developed independently by different vendors
    • Heterogeneous S/W infrastructure
    • Bare-metal / Proprietary OS
    • Linux / Android
  – Different license issues

• Consolidation via virtualization
  – Each application can maintain its own implementation
  – Minimizes re-certification process
  – IP protection, license segregation
  – Fault isolation
Virtual Machines and Hypervisor

- Two-level hierarchical scheduling structure
  - Task scheduling and VCPU scheduling

```
<table>
<thead>
<tr>
<th>VM1</th>
<th>VM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task (\tau_1)</td>
<td>Task (\tau_5)</td>
</tr>
<tr>
<td>Task (\tau_2)</td>
<td>Task (\tau_6)</td>
</tr>
<tr>
<td>Task (\tau_3)</td>
<td>Task (\tau_7)</td>
</tr>
<tr>
<td>Task (\tau_4)</td>
<td>Task (\tau_8)</td>
</tr>
<tr>
<td>VCPU1</td>
<td>VCPU3</td>
</tr>
<tr>
<td>VCPU2</td>
<td>VCPU4</td>
</tr>
</tbody>
</table>

Hypervisor

VCPU Scheduler

Physical Core 1 (PCPU1)

Physical Core 2 (PCPU2)
```
Resource Sharing

- Consolidation inevitably causes the sharing of physical and logical resources
  - Sensors
  - Network interfaces
  - I/O devices
  - Shared memory
  Requires mutually-exclusive locks to avoid race conditions
- Increase in processor core count
  - More tasks can be consolidated
  - More resource sharing is expected

We need a synchronization mechanism with bounded blocking times for multi-core real-time virtualization
## Previous Work

<table>
<thead>
<tr>
<th>Context</th>
<th>Synch. protocols</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-core scheduling</td>
<td>MPCP [1]</td>
<td>• Designed for <strong>non-hierarchical scheduling</strong></td>
</tr>
<tr>
<td></td>
<td>MSRP [2]</td>
<td>• <strong>Unbounded blocking time</strong> in a multi-core virtualization environment (VCPU preemption / budget depletion)</td>
</tr>
<tr>
<td></td>
<td>FMLP [3]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MSOS [4]</td>
<td></td>
</tr>
<tr>
<td>Hierarchical scheduling</td>
<td>HSRP [5]</td>
<td>• Designed for <strong>single-core systems</strong></td>
</tr>
<tr>
<td></td>
<td>SIRAP [6]</td>
<td>• <strong>Not</strong> extended to multi-core systems</td>
</tr>
<tr>
<td></td>
<td>RRP [7]</td>
<td>• No software mechanism for virtualization</td>
</tr>
</tbody>
</table>

Our Approach

- **vMPCP**: a virtualization-aware multiprocessor priority ceiling protocol
  - Provides *bounded blocking time* on accessing shared resources in multi-core virtualization
    - Two-level hierarchical priority ceilings
    - Para-virtualization interface
  - VCPU budget replenishment policies
    - Periodic server
    - Deferrable server
  - Optional VCPU budget overrun
  - Implemented on the KVM hypervisor of Linux/RK
Outline

• Introduction

• vMPCP Framework
  – System model
  – Penalties from shared resources
  – vMPCP details
  – Analysis

• Evaluation

• Conclusion
System Model (1)

- Partitioned fixed-priority scheduling for both VCPUs and tasks

- VCPU $\nu_i: (C_i^\nu, T_i^\nu)$
  - $C_i^\nu$: Maximum execution budget
  - $T_i^\nu$: Budget replenishment period

- VCPU budget replenishment policy
  - Periodic server
  - Deferrable server

- Task $\tau_i: \left( (C_{i,1}, E_{i,1}, C_{i,2}, E_{i,2}, ..., E_{i,S_i}, C_{i,S_i+1}), T_i \right)$
  - $C_{i,j}$: WCET of j-th normal execution segment
  - $E_{i,j}$: WCET of j-th critical section segment
  - $T_i$: Period
  - $S_i$: The number of critical section segments

Alternating sequence of normal execution and critical section segments
Local shared resources
Resources shared among tasks on the same VCPU → Local blocking

Global shared resources
Resources shared among tasks on other VCPUs that may be located on other PCPUs → Remote blocking
Penalties from Shared Resources

- **Local blocking**
  - Task waiting on the executions of lower-priority tasks on the same VCPU

- **Remote blocking**
  - Task waiting on the executions of tasks on other VCPUs

**Additional timing penalties caused by remote blocking**
- Back-to-back execution
- Multiple priority inversions

**Remote blocking time in a virtualized environment**
- Preemptions by higher-priority VCPUs
- VCPU budget depletion

**Goal:** minimize and bound the remote blocking time in a multi-core virtualization environment
vMPCP Overview

• Local shared resource
  – Follows the uniprocessor PCP

• Global shared resource
  – Uses hierarchical priority ceilings (Task-level and VCPU-level)
  – Suppresses task-level and VCPU-level preemptions while accessing a global resource → Reduces remote blocking time
  – Two-level priority queue for a mutex protecting a global resource

No need to compare task priorities in one VPCU with those in other VCPUs → Good for different guest OSs (ex, μc/os-ii and Linux)
VCPU Budget Overrun

• vMPCP provides an option for VCPUs to overrun their budgets when their tasks are in global critical sections (gcs’s)
  – Allows tasks to complete their gcs’s, even though their VCPU has exhausted its budget
  – Pro: reducing remote blocking time
  – Con: more interference to lower-priority VCPUs

Periodic server with overrun
• Obeys the periodic-server’s property of having no back-to-back execution

Deferrable server with overrun
• Can overrun more flexibly than a periodic server

Leads to different remote blocking time in analysis
Para-virtualization Interface

• In current virtualization solutions, the hypervisor is unaware of the executions of critical sections within VCPUs

• Solution: vMPCP para-virtualization interface
  – What is para-virtualization?
    • Small modifications to guest OSs or device drivers to achieve high performance and efficiency
  – To let the hypervisor know the executions of global critical sections within VCPUs
  – Two hypercalls
    \[
    \text{vmpcp\_start\_gcs}() \quad \text{vmpcp\_finish\_gcs}()
    \]
vMPCP Analysis (1)

- **Scope of our analysis**
  - VCPU schedulability
  - Task schedulability
  - Considers four different use cases of vMPCP

<table>
<thead>
<tr>
<th>VCPU budget replenish policies</th>
<th>With overrun</th>
<th>With no overrun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic server</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Deferrable server</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
vMPCP Analysis (2)

- **VCPU Schedulability**
  - Worst-case response time of VCPU ≤ VCPU period
    
    $W_{i,v,n+1} = C_i^v + O_i^v + B_i^v(W_{i,v,n}^v) + \sum_{v_n \in P(v_i) \land h > i} \left[ \frac{W_{i,v,n}^v + J_h^v}{T_h^v} \right] \cdot (C_h^v + O_h^v)$

- **Task Schedulability**
  - Worst-case response time of task ≤ Task deadline
    
    $W_{i,n+1} = C_i + B_i^l + B_i^r + \sum_{\tau_h \in V(\tau_i) \land h > i} \left[ \frac{W_{i,n}^v + J_h + B_h^r}{T_h} \right] C_h$
    
    $+ \left[ \frac{W_{i,n}^v + C_k^v}{T_k^v} \right] (T_k^v - C_k^v)$

- VCPU budget overrun
- Blocking time
- Higher-priority VCPUs
- Local and remote blocking times
- Higher-priority tasks in the same VCPU
- VCPU budget and budget replenishment period
Outline

• Introduction

• vMPCP Framework

• Evaluation
  – Comparison of different configurations
  – Implementation
  – Case study

• Conclusion
Comparison of Different Configurations

- **Purpose**: to explore the impact of different uses of vMPCP on task schedulability

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSwO</td>
<td>Periodic Server with Overrun</td>
</tr>
<tr>
<td>DSwO</td>
<td>Deferrable Server with Overrun</td>
</tr>
<tr>
<td>PSnO</td>
<td>Periodic Server with no Overrun</td>
</tr>
<tr>
<td>DSnO</td>
<td>Deferrable Server with no Overrun</td>
</tr>
</tbody>
</table>

- **Experimental setup**
  - Used randomly-generated tasksets
  - **Metric**: the percentage of schedulable tasksets
  - Factors considered
    - Number of global critical sections per task
    - VCPU period
    - Size of a global critical section
    - Utilization of tasks within each VCPU
    - Number of lockers per mutex
In these two cases, \textbf{DSwO} outperforms the other schemes. \textbf{What about other cases?}
The schemes with no overrun (PSnO and DSnO) perform better than the schemes with overrun.

Findings:
(1) There is no single scheme that dominates the others.
(2) When overrun is used, a deferrable server outperforms a periodic server.
Implementation

- KVM Hypervisor + Linux/RK
  - KVM: A full open-source virtualization solution for Linux
  - Linux/RK: Resource kernel implementation based on the Linux kernel

- vMPCP implementation cost
  - Target system: Intel Core i7-2600 quad-core 3.4 GHz

<table>
<thead>
<tr>
<th>Types</th>
<th>Mutex APIs</th>
<th>Avg (μsec)</th>
<th>Max (μsec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra-VM</td>
<td>open (create new mutex)</td>
<td>4.16</td>
<td>7.14</td>
</tr>
<tr>
<td></td>
<td>open (existing mutex)</td>
<td>1.87</td>
<td>3.64</td>
</tr>
<tr>
<td></td>
<td>destroy</td>
<td>1.83</td>
<td>3.50</td>
</tr>
<tr>
<td></td>
<td>lock</td>
<td>3.51</td>
<td>5.69</td>
</tr>
<tr>
<td></td>
<td>trylock</td>
<td>2.75</td>
<td>5.15</td>
</tr>
<tr>
<td></td>
<td>unlock</td>
<td>2.26</td>
<td>2.68</td>
</tr>
<tr>
<td></td>
<td>*vmpcp_start_gcs</td>
<td>2.05</td>
<td>2.88</td>
</tr>
<tr>
<td></td>
<td>*vmpcp_finish_gcs</td>
<td>1.40</td>
<td>1.60</td>
</tr>
<tr>
<td>Inter-VM</td>
<td>open (create new mutex)</td>
<td>1.79</td>
<td>3.48</td>
</tr>
<tr>
<td></td>
<td>open (existing mutex)</td>
<td>1.76</td>
<td>3.35</td>
</tr>
<tr>
<td></td>
<td>destroy</td>
<td>1.49</td>
<td>1.78</td>
</tr>
<tr>
<td></td>
<td>lock</td>
<td>3.09</td>
<td>5.31</td>
</tr>
<tr>
<td></td>
<td>trylock</td>
<td>2.80</td>
<td>5.29</td>
</tr>
<tr>
<td></td>
<td>unlock</td>
<td>1.93</td>
<td>2.57</td>
</tr>
</tbody>
</table>
Case Study

- **Purpose:** compare vMPCP against a virtualization-unaware protocol (MPCP)
  - **Metric:** task response time
- **System configuration**
  - Hypervisor: Linux/RK + KVM
  - Guest OS: Linux/RK
  - VCPU budget replenish policy: **deferrable server**
Case Study Results

Virtualization-unaware synchronization protocol (MPCP)

Virtualization-aware synchronization protocol (vMPCP w/ overrun)

vMPCP yields 29% shorter response time on average
Conclusions

- vMPCP: a synchronization protocol for multi-core VMs
  - Bounded blocking time on accessing local/global shared resources
    - Hierarchical priority ceilings
    - Two-level priority queue for a mutex waiting list
    - Para-virtualization interface
  - Schedulability analysis and experimental results
    - Deferrable server outperforms periodic server when overrun is used
    - The use of overrun does not always yield better schedulability
  - KVM + Linux/RK: https://rtml.ece.cmu.edu/redmine/projects/rk/
    - In our case study, vMPCP yields 29% shorter task response time compared to a virtualization-unaware synchronization protocol

- Future Work
  - Memory interference, compositional framework