# Real-Time Cache Management for Multi-Core Virtualization

Hyoseung Kim [1,2]  Raj Rajkumar [2]

[1] University of Riverside, California

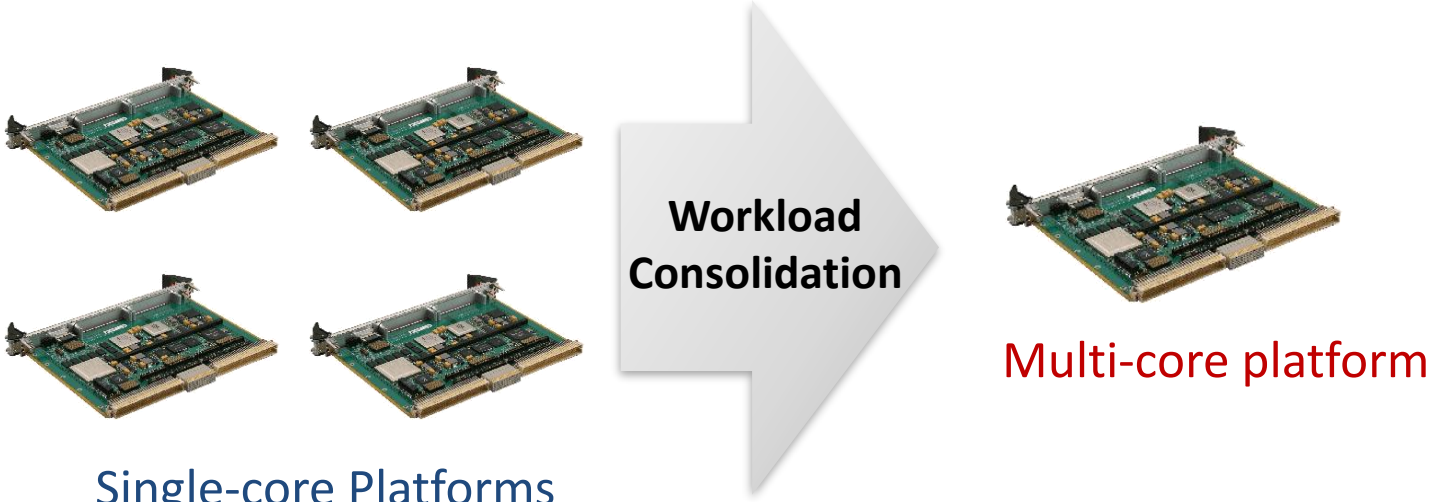
[2] Carnegie Mellon University

UC RIVERSIDE UNIVERSITY OF CALIFORNIA

Carnegie Mellon

# Benefits of Multi-Core Processors

- **Consolidation of real-time systems onto a single hardware platform**
  - Reduces the number of CPUs and wiring harness among them
  - Leads to a significant reduction in size, weight, and cost requirements

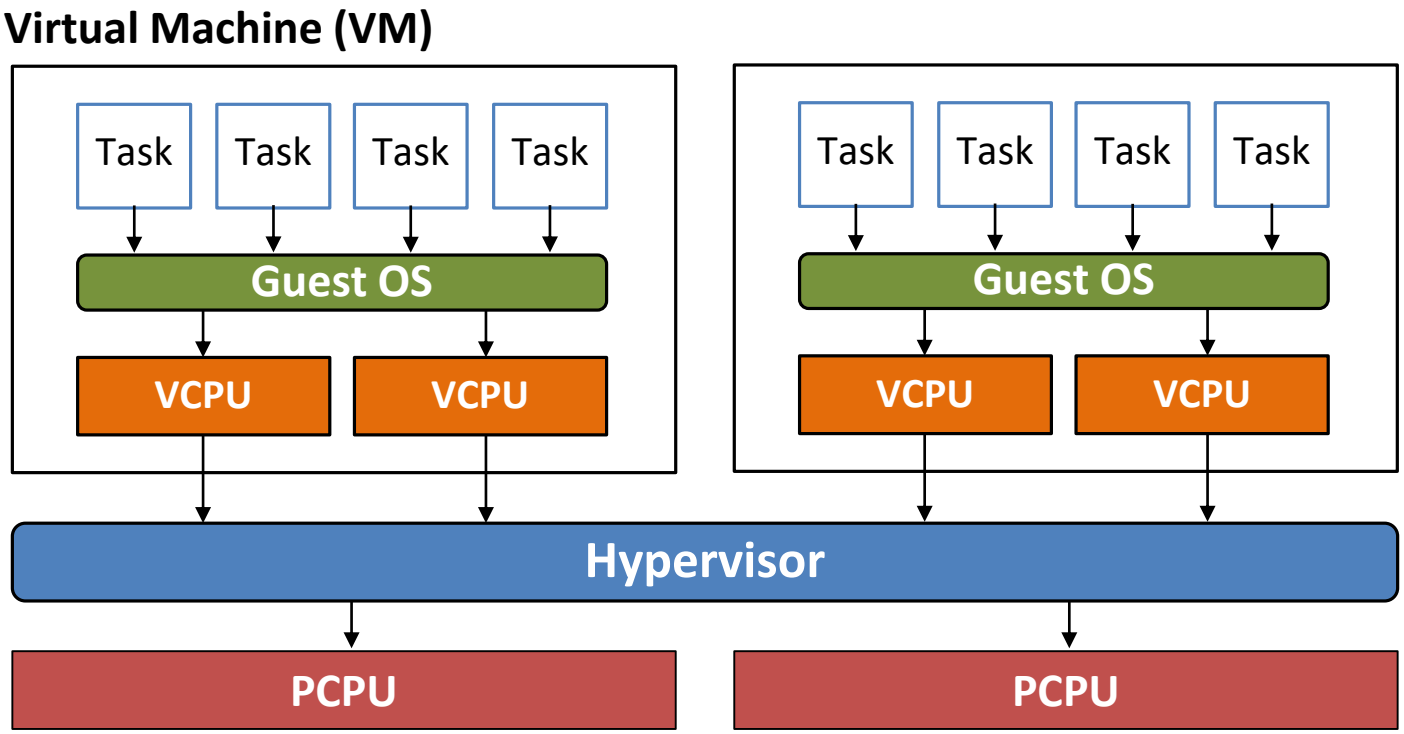


# Virtualization of Real-Time Systems

- **Barriers to consolidation**
  - Each app. could have been developed independently by different vendors
    - Bare-metal / Proprietary OS
    - Linux / Android
  - Different license issues
- **Consolidation via virtualization**
  - Each application can maintain its own implementation
  - Minimizes re-certification process
  - Fault isolation
  - IP protection, license segregation

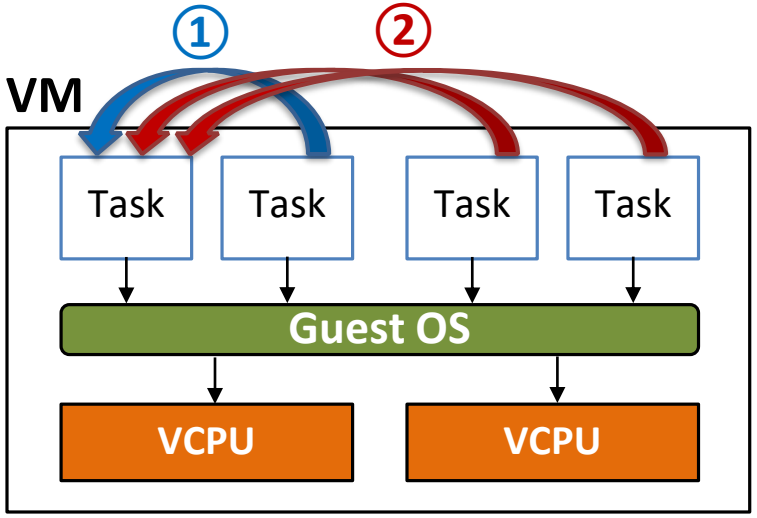
# Virtual Machines and Hypervisor

- Two-level hierarchical scheduling structure
  - Task scheduling on virtual CPUs (VCPUs) by Guest OSs
  - VCPU scheduling on physical CPUs (PCPUs) by the hypervisor



# Shared Cache Interference

- **Shared last-level cache (LLC)**
  - Reduces task execution time
  - Allows consolidating more tasks onto a single hardware platform
- **Cache interference in multi-core virtualization**




① **Intra-VCPU cache interference:** tasks running on the same VCPU

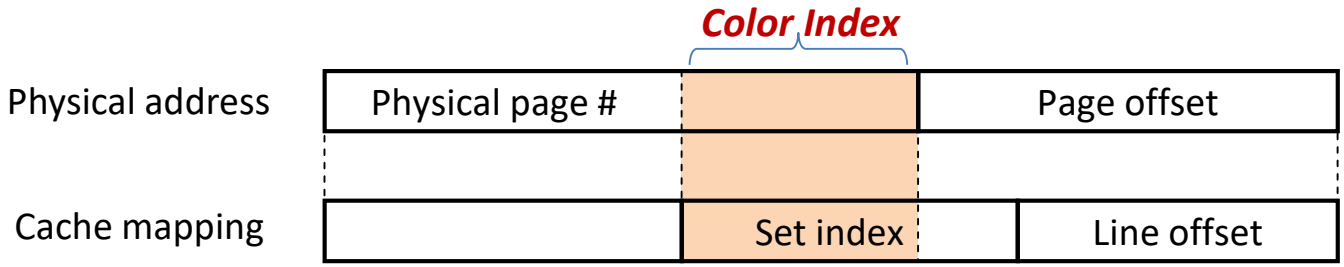
② **Inter-VCPU cache interference:** tasks running on different VCPUs

*Cache interference must be addressed for real-time predictability*

# Page Coloring for S/W Cache Control

- **Page coloring**
  - Software-based, OS-level cache partitioning mechanism
  - Used by many prior cache management schemes developed for ***non-virtualized*** multi-core systems [1, 2, 3, 4]

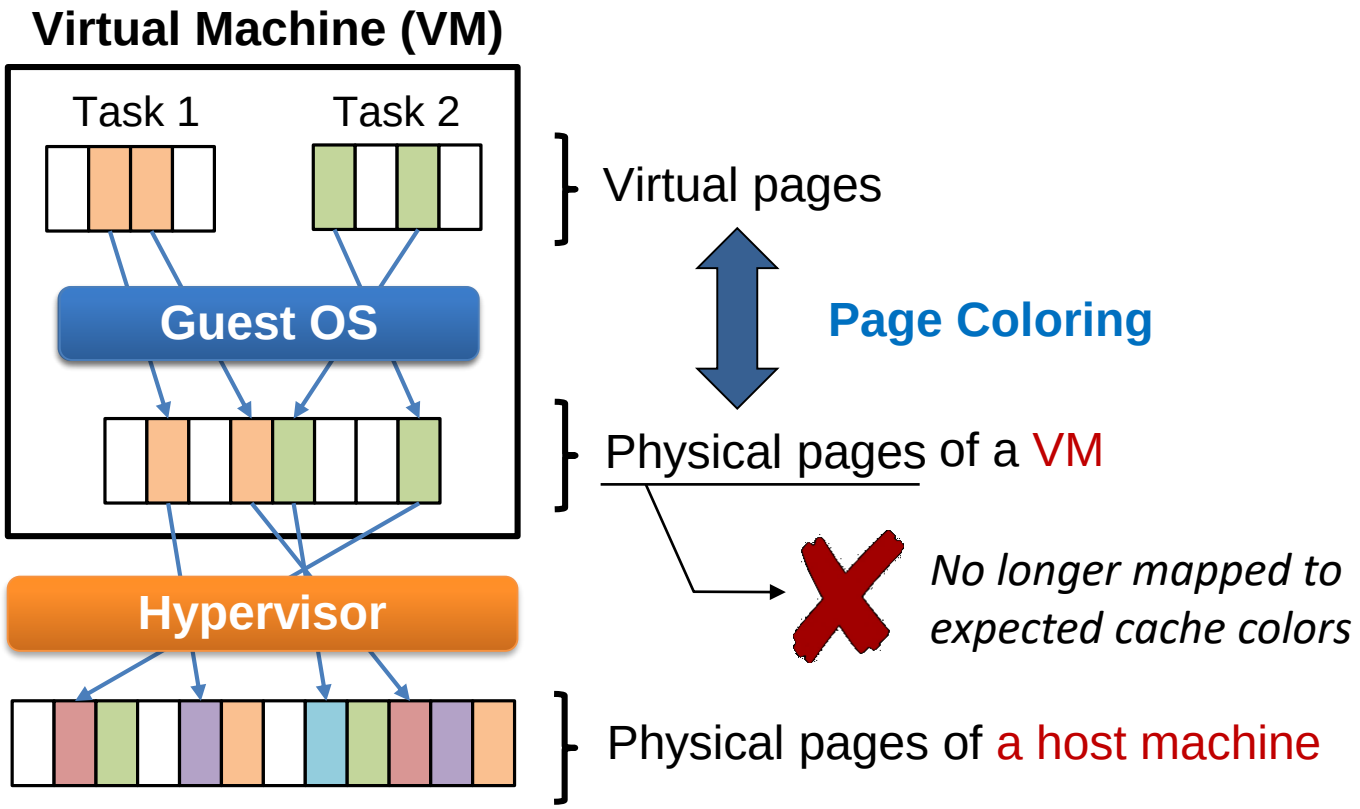
[ Physically-indexed, set-associative cache ]




[1] H. Kim et al. A coordinated approach for practical OS-level cache management in multi-core real-time systems. In *ECRTS*, 2013.
[2] R. Mancuso et al. Real-time cache management framework for multi-core architectures. In *RTAS*, 2013.
[3] N. Suzuki et al. Coordinated bank and cache coloring for temporal protection of memory accesses. In *ICESS*, 2013.
[4] B. C. Ward et al. Making shared caches more predictable on multicore platforms. In *ECRTS*, 2013..

# Challenges in Virtualization (1/2)

1. Page coloring and algorithms based on it do ***not*** work in a VM due to the additional address layer at the hypervisor



# Challenges in Virtualization (2/2)

2. Even if page coloring works in a VM, *legacy systems* to be virtualized may not have page coloring support
   - Will suffer from cache interference
   - Need a support for closed-source guest OSs

3. Prior real-time cache management schemes cannot answer:
   - How to find a VM's resource requirement in the presence of cache interference?
   - How to allocate the host machine's cache to VMs to be consolidated?

# Our Contributions

- Real-time cache management for multi-core virtualization
- **vLLC and vColoring**
  - Provide a way to allocate host cache colors to individual tasks running in a virtual machine → *First software-based techniques*
  - Prototype implemented in KVM running on x86 and ARM platforms
- **Cache management scheme**
  - Allocates cache colors to tasks in a VM while satisfying timing constraints
  - Finds a VM's CPU demand w.r.t. the number of cache colors assigned to it
  - Minimizes the total utilization of VMs to be consolidated → *First approach*

# Outline

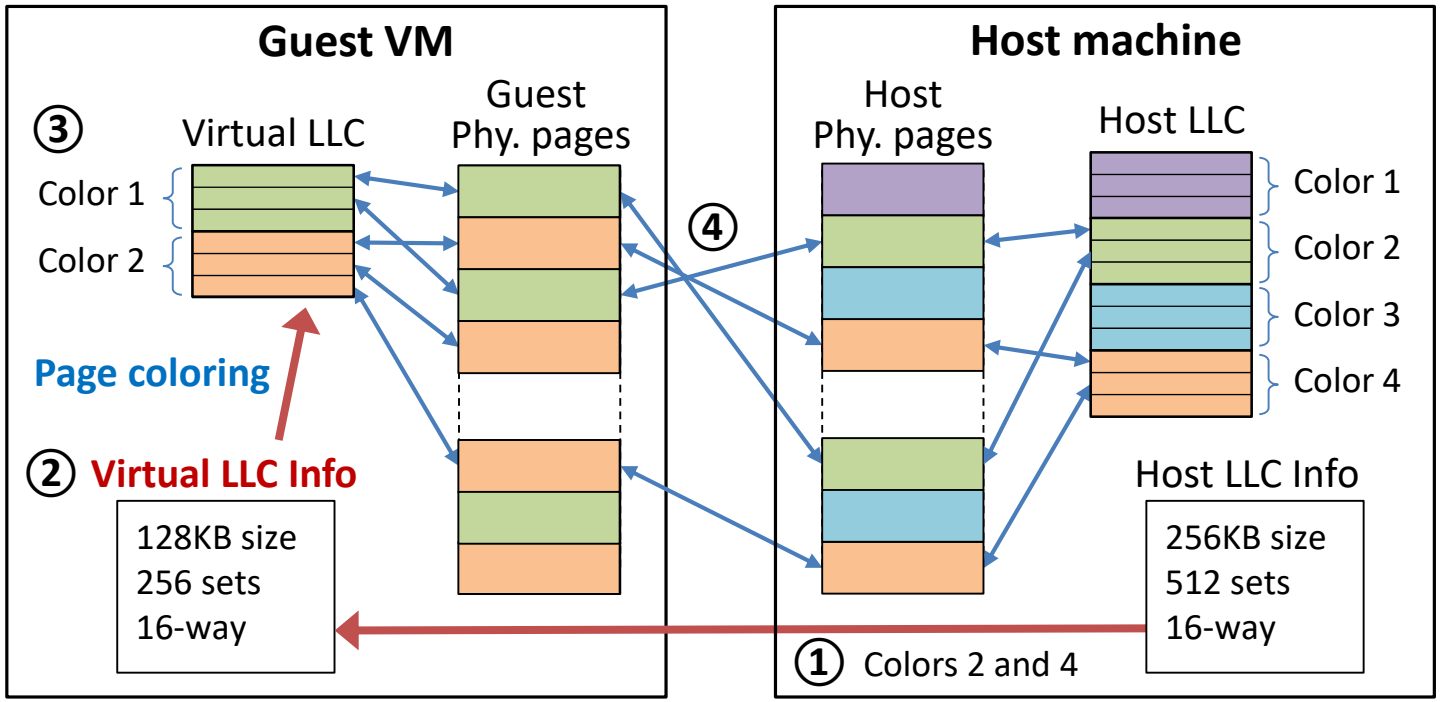
- Introduction and Motivation
- Real-Time Cache Management for Multi-Core Virtualization
  - System model
  - vLLC and vColoring
  - Cache management scheme
- Evaluation
- Conclusions

# System Model

- Hypervisor: implements page coloring
- Guest OSs: may or may not have page coloring
- Partitioned fixed-priority scheduling for both the hypervisor & guest OSs
- $\text{VM} \coloneqq (v_1, v_2, \ldots, v_{N_{vcpu}})$
- VCPU $v_i \coloneqq (C_i^v(k), T_i^v)$
  - $C_i^v(k)$: Execution budget with $k$ cache colors assigned to it
  - $T_i^v$: Budget replenishment period
- Task $\tau_i \coloneqq (C_i(k), T_i, D_i)$
  - $C_i(k)$: Worst-case execution time (WCET) with $k$ cache colors assigned to it
  - $T_i$: Period
  - $D_i$: Relative deadline

# vLLC: Virtual Last-Level Cache

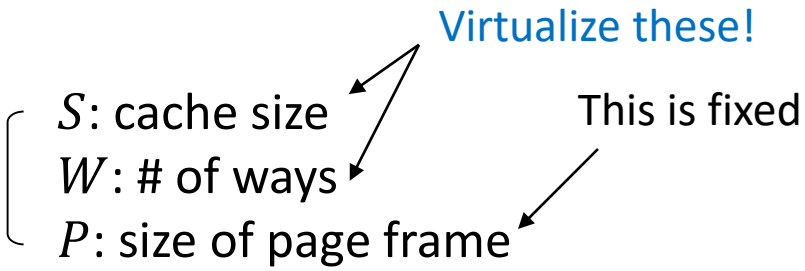
- Technique for guest OSs with page coloring (e.g., Linux/RK)
  - Provides
    - Virtual LLC (Last-level cache) information
    - Host physical pages corresponding to the virtual LLC




Guest Cache Color 1 = Host Cache Color 2, Guest Cache Color 2 = Host Cache Color 4

# vLLC: Virtual Last-Level Cache

- Virtual LLC information
  - # of cache colors $n = S/(W \cdot P)$

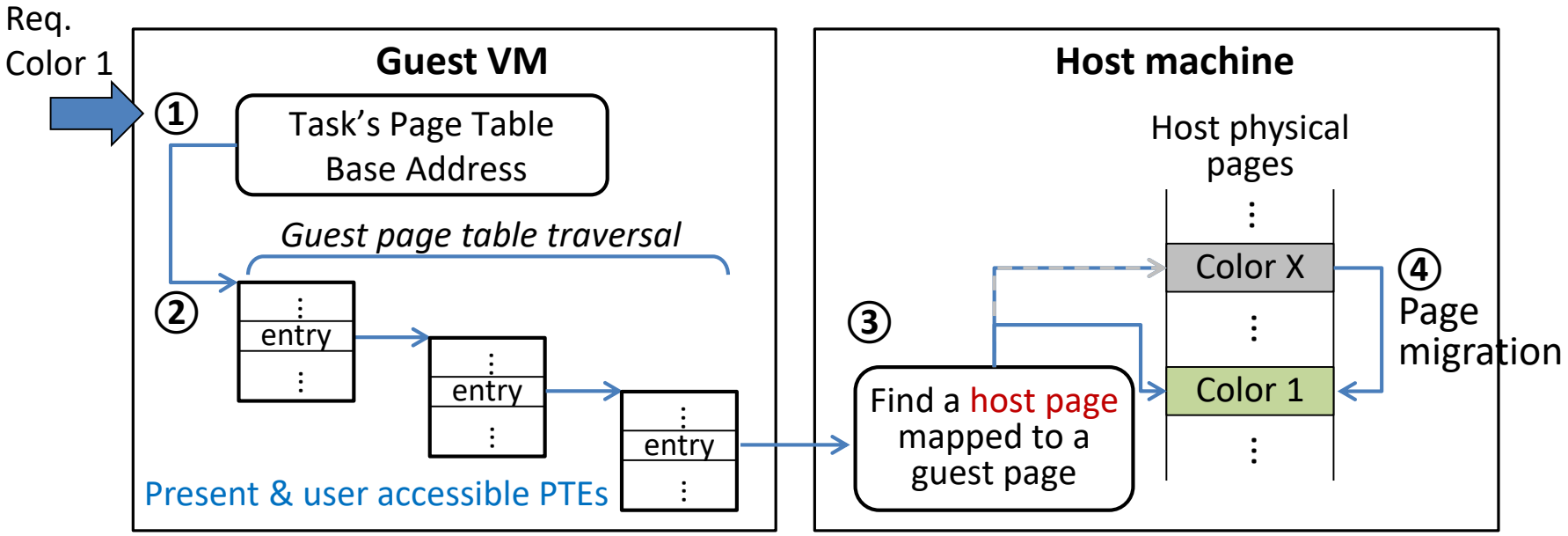



- Trapping and emulating cache-related operations
  - x86: executions of a CPUID instruction
  - ARM Cortex-A15: accesses to CCSIDR and CSSERR registers

- Limitations
  - The number of cache colors is restricted to a power of two
  - Cannot support a guest OS where page coloring is hard-coded

# vColoring: Virtual Coloring of Cache

- Technique for guest OSs without page coloring support
  - Re-maps guest pages to host pages for the requested cache colors
  - Applicable to VMs running closed-source, proprietary guest OSs




Guest page tables are *not* changed at all

→ Cache allocation is transparent to the guest OS

# Outline

- Introduction and Motivation
- Real-Time Cache Management for Multi-Core Virtualization
  - System model
  - vLLC and vColoring
  - Cache management scheme
- Evaluation
- Conclusions

# Allocating Cache Colors to Tasks

- Two types of cache interference: **Inter-VCPU** & **Intra-VCPU**
- **Simple approach 1:** Complete cache partitioning (CCP)
  - No cache sharing at all
  - May result in poor performance due to smaller cache size
- **Simple approach 2:** Complete cache sharing (CCS) among tasks on the same VCPU
  - No cache sharing between tasks on different VCPUs
  - Bounds intra-VCPU interference with Cache-Related Preemption Delay (CRPD)
  - May suffer from high CRPD
- **Our approach:** Controlled sharing of cache colors on each VCPU
  - Goal: finds a cache-to-task allocation that minimizes taskset utilization → NP-hard
  - Approximates CRPD caused by task $\tau_i$ to reduce the complexity

$$S_i \leftarrow \text{argmin}_{1 \le k \le N_{cache}} \left( \frac{C_i(k)}{T_i} + \frac{\gamma_{i,n}}{T_i} \right)$$

*Assuming all other tasks have been assigned all cache colors*

# Designing a Cache-Aware VM

- **VM's CPU demand**
  - The sum of the CPU demands of VCPUs in the VM
    - Affected by the allocation of tasks and cache colors to VCPUs
- **Our approach**: Cache-aware VM designing algorithm (CAVM)
  - **Phase 1:** Allocates cache-sensitive tasks to the same VCPU so that they can benefit from cache sharing
    - After Phase 1, each VCPU has its own taskset
  - **Phase 2:** Derives each VCPU's CPU demands w.r.t. the number of cache colors assigned to it
    - Determines the minimum budget $C_i^v(k)$ for all possible $k$ values

# Allocating Host Cache Colors to VMs

- Goal: determines the number of cache colors for each VCPU of the VMs to be consolidated, while minimizing the total VM utilization

- Our approach: Dynamic programming

Minimum number of cache colors to satisfy timing constraints

$$U(k) = \begin{cases} \infty \text{ (unschedulable)} & : k < z \\ \sum_{v_i \in \mathcal{V}} \frac{C_i^v(\sigma_{i,k})}{T_i} & : k = z \\ \min_{z \le k' < k} \left( U(k') - \max_{v_i \in \mathcal{V}} \frac{C_i^v(\sigma_{i,k'}) - C_i^v(\sigma_{i,k'+(k-k')})}{T_i^v} \right) & : k > z \end{cases}$$

Finds the maximum utilization gain made by additional cache colors

# Outline

- Introduction and Motivation
- Real-Time Cache Management for Multi-Core Virtualization
  - System model
  - vLLC and vColoring
  - Cache management scheme
- Evaluation
- Conclusions

# Implementation

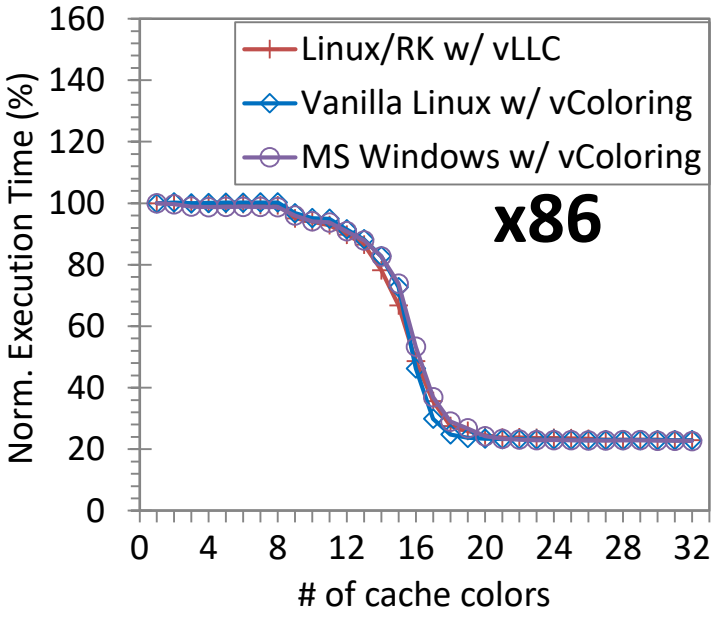
- Experimental setup
  - **x86**: Intel i7-2600 four cores @ 3.4 GHz → 8 MB LLC, 32 colors
  - **ARM**: Exynos 5422 (four Cortex-A15 cores @ 2 GHz) → 2 MB LLC, 32 colors
  - **Hypervisor**: Implemented in KVM, but applicable to other hypervisors
  - **Guest OSs**: Linux/RK, Vanilla Linux, MS Windows Embedded (x86 only)

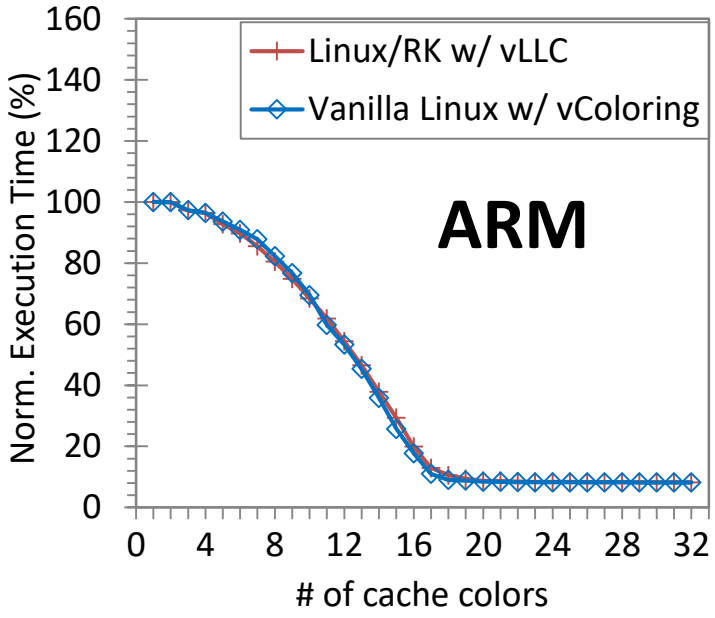
- Implementation overhead

| Name | Items | Cost (nsec) | |
|---|---|---|---|
| | | x86 | ARM |
| vLLC | Virtual LLC emulation | 787 | 12212 |
| | Color check in GPP-to-HPP mapping | 34 | 921 |
| vColoring | Page migration for GPP re-mapping | 2359 | 31864 |

# vLLC and vColoring

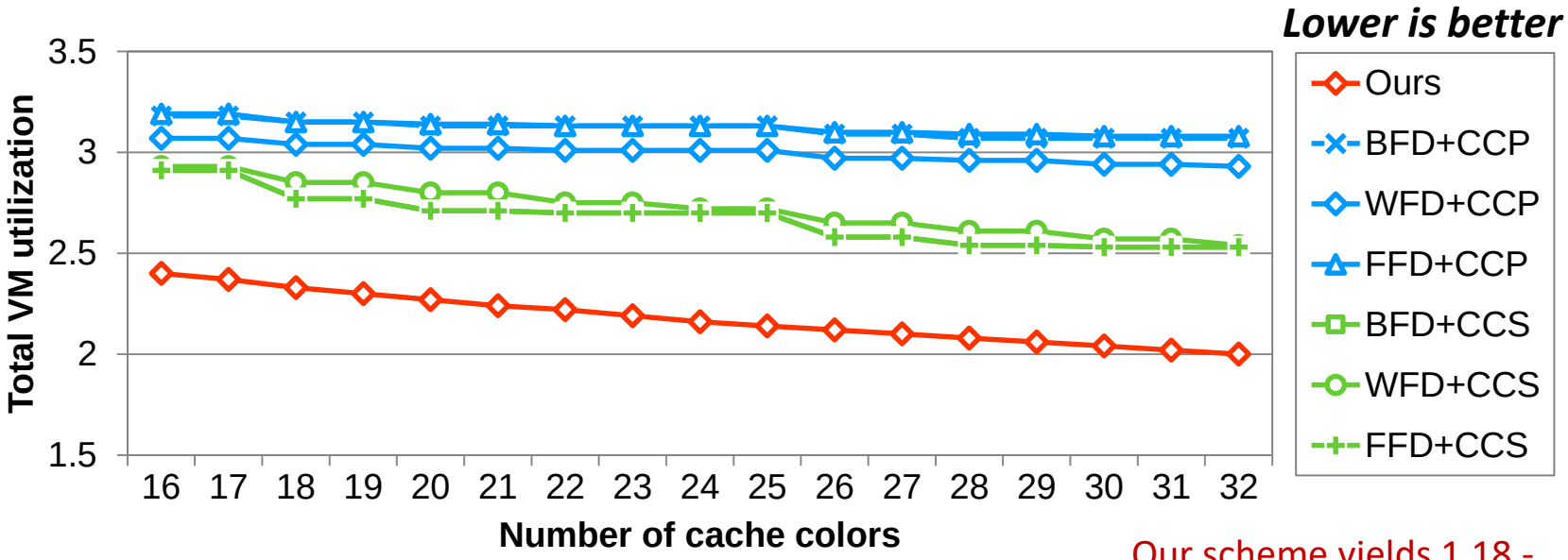
- Execution times of a synthetic task



# Cache Management Scheme

- Experimental results with random tasksets
  - Quad-core, 2 VMs, 4 VCPUs per VM, 2MB LLC, 10 – 15 tasks
  - Cache color reload time: 207 $\mu$sec (obtained from our ARM board)
- VM utilization w.r.t. the number of cache colors




Our scheme yields 1.18 - 1.54x lower utilization

# Conclusions

- Real-time cache management for multi-core virtualization
- vLLC and vColoring
  - *Hypervisor-level techniques* to control cache allocation to individual tasks running in a virtual machine
  - Evaluated with Linux/RK, vanilla Linux, and MS Embedded Windows
- Cache management scheme
  - Determines cache to task allocation
  - Designs a VM in the presence of cache interference
  - Minimizes the total utilization of VMs

  } *Up to 1.54x lower utilization*
- Future work: main memory interference in virtualization
  - vColoring: applicable to DRAM bank partitioning

# Thank You

## Real-Time Cache Management for Multi-Core Virtualization

Hyoseung Kim [1,2] Raj Rajkumar [2]

[1] University of Riverside, California

[2] Carnegie Mellon University