# COLORIS: A Dynamic Cache Partitioning System Using Page Coloring

Ying Ye, Richard West, Zhuoqun Cheng, Ye Li

Computer Science Department
Boston University

# Overview

# Background

- For multicore platforms, tightly-coupled on-chip resources allow faster data sharing between processing cores, at the same time, suffering from potentially heavy resource contention

# Background

- For multicore platforms, tightly-coupled on-chip resources allow faster data sharing between processing cores, at the same time, suffering from potentially heavy resource contention
- Most commercial off-the-shelf systems only provide best effort service for accessing the shared LLC

# Background

- For multicore platforms, tightly-coupled on-chip resources allow faster data sharing between processing cores, at the same time, suffering from potentially heavy resource contention
- Most commercial off-the-shelf systems only provide best effort service for accessing the shared LLC
    - unpredictable caching behaviors
    - severe performance degradation
    - compromised QoS

# Background

- For multicore platforms, tightly-coupled on-chip resources allow faster data sharing between processing cores, at the same time, suffering from potentially heavy resource contention
- Most commercial off-the-shelf systems only provide best effort service for accessing the shared LLC
    - unpredictable caching behaviors
    - severe performance degradation
    - compromised QoS
- Performance isolation needed for QoS-demanding systems

# Page Coloring

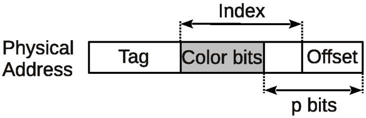



Figure : Page Color Bits

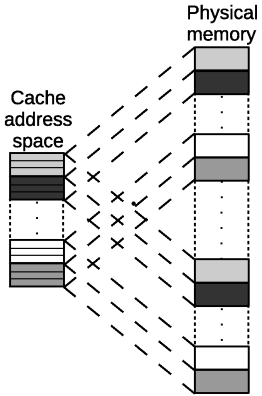
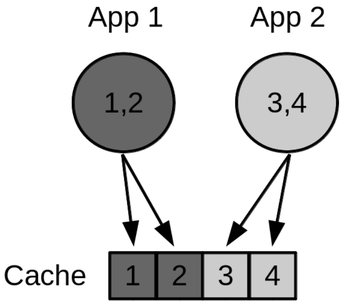
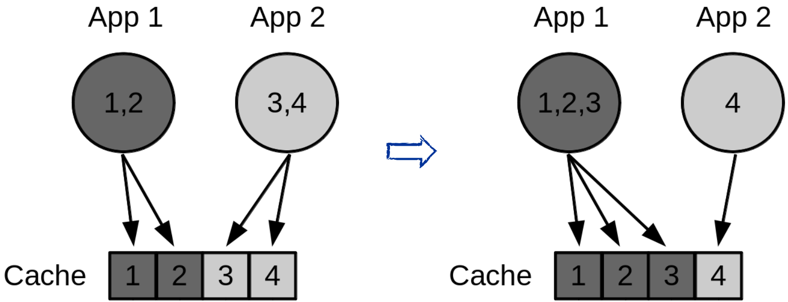



Figure : Mapping Between Memory Pages and Cache Space

# Page Coloring



# Page Coloring



# Dynamic Partitioning

# Dynamic Partitioning

- When to re-partition LLC?

# Dynamic Partitioning

- When to re-partition LLC?
  - phase change; absent of a-priori knowledge

# Dynamic Partitioning

- When to re-partition LLC?
  - phase change; absent of a-priori knowledge
- What is the right partition size?

# Dynamic Partitioning

- When to re-partition LLC?
  - phase change; absent of a-priori knowledge
- What is the right partition size?
- How to recolor memory?

# Dynamic Partitioning

- When to re-partition LLC?
  - phase change; absent of a-priori knowledge
- What is the right partition size?
- How to recolor memory?
  - heavy overhead; inefficient use

# Dynamic Partitioning

- When to re-partition LLC?
  - phase change; absent of a-priori knowledge
- What is the right partition size?
- How to recolor memory?
  - heavy overhead; inefficient use
- How to work with over-committed systems?

# Contribution

- Our work tries to solve all problems above associated with implementing dynamic page coloring in production systems
- We proposes an efficient page recoloring framework in the Linux kernel, called COLORIS (COLOR ISolation)

# Page COLOR ISolation Architecture

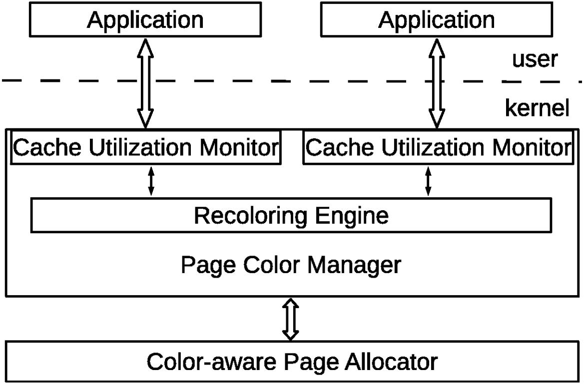



Figure : COLORIS Architecture

# Color-aware Page Allocator

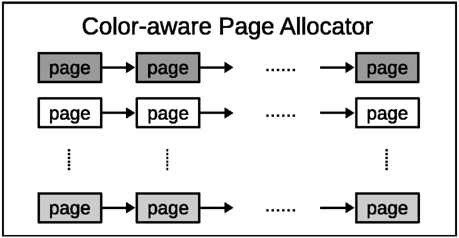
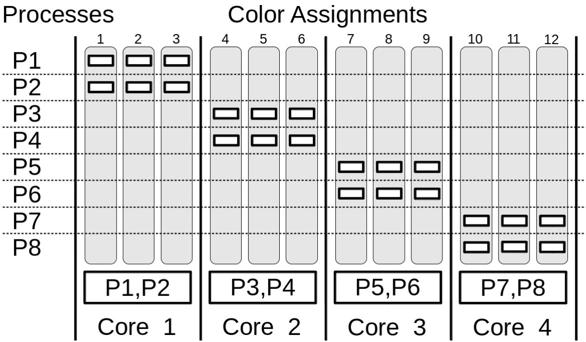



Figure : Page Allocator

# Page Color Manager

- Static color assignment
  - Cache is divided into N sections of contiguous colors
  - Each cache section is statically assigned to a core
    - local core; remote core
  - Each process is assigned a section of page colors and runs on the corresponding core
    - local color; remote color

# Static Color Assignment

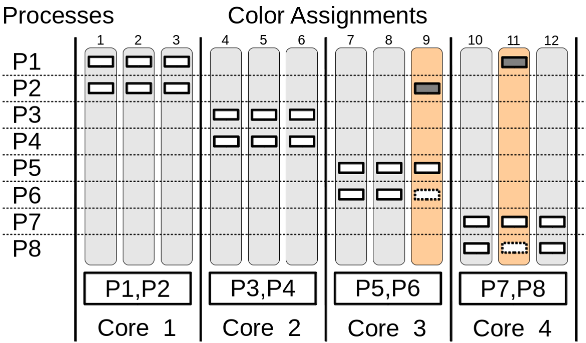


# Dynamic Color Assignment

- Dynamic color assignment:
  - Applications with low cache demand may give up page colors
  - Applications needing more cache may acquire page colors from other cache sections

# Dynamic Color Assignment

- Dynamic color assignment:
  - Applications with low cache demand may give up page colors
  - Applications needing more cache may acquire page colors from other cache sections



# Cache Utilization Monitor

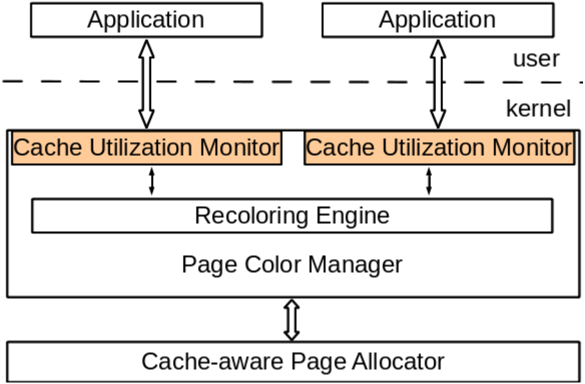



Figure : COLORIS Architecture

# Cache Utilization Monitor

- Measures cache usage of individual applications:
  - $cache\ miss\ rate = \frac{misses}{accesses}$

# Cache Utilization Monitor

- Measures cache usage of individual applications:
    - $cache\ miss\ rate = \frac{misses}{accesses}$
- Triggers cache re-partitioning:
    - miss rate higher than *HighThreshold*
    - miss rate lower than *LowThreshold*

# Cache Re-partitioning

## Color Hotness

The number of processes sharing the color

# Cache Re-partitioning

**Color Hotness**

The number of processes sharing the color

- Global Hotness: number of owners on all cores
- Remote Hotness: number of owners on remote cores

# Cache Re-partitioning

## Color Hotness

The number of processes sharing the color

- Global Hotness: number of owners on all cores
- Remote Hotness: number of owners on remote cores
  - if color A is in the cache section statically assigned to core X, all other cores are called remote cores with respect to A

# Cache Re-partitioning

```
procedure alloc_colors(num)
new ← φ
while num > 0
   if needRemote()
     new +=
       pick_coldest_remote()
   else
     new +=
       pick_coldest_local()
   num ← num − 1
return new
end procedure
```

# Cache Re-partitioning

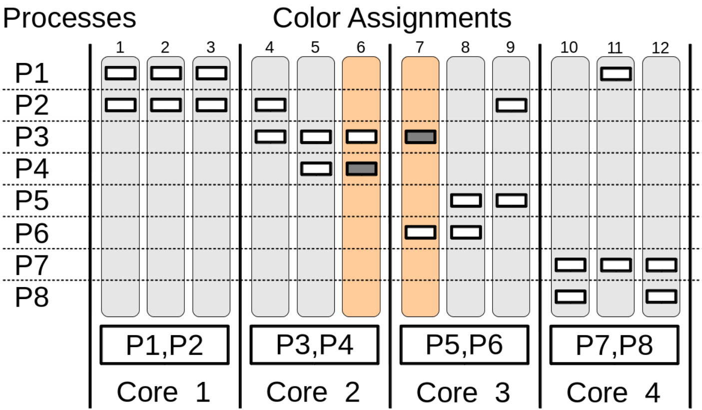
**procedure** alloc_colors(num)
$new \leftarrow \phi$
**while** $num > 0$
  **if** needRemote()
    $new += pick\_coldest\_remote()$
  **else**
    $new += pick\_coldest\_local()$
  $num \leftarrow num - 1$
**return** $new$
**end procedure**

- pick_coldest_remote: pick a color in a remote cache section, with the smallest global hotness

# Cache Re-partitioning

**procedure** alloc_colors(num)
*new* $\leftarrow \phi$
**while** *num* $> 0$
  **if** needRemote()
    *new* $+ =$
      *pick_coldest_remote*()
  **else**
    *new* $+ =$
      *pick_coldest_local*()
  *num* $\leftarrow$ *num* $- 1$
**return** *new*
**end procedure**

- pick_coldest_remote:
  pick a color in a remote cache section, with the smallest global hotness
- pick_coldest_local:
  pick a color in the local cache section, with the smallest remote hotness


Processes
Color Assignments
1
2
3
4
5
6
7
8
9
10
11
12
P1
P2
P3
P4
P5
P6
P7
P8
P1,P2
P3,P4
P5,P6
P7,P8
Core 1
Core 2
Core 3
Core 4

# Cache Re-partitioning

```
procedure pick_victims(num)
victims ← φ
while num > 0
   if hasRemote()
      victims += 
        pick_hottest_remote()
   else
      victims += 
        pick_hottest_local()
   num ← num − 1
return victims
end procedure
```

# Cache Re-partitioning

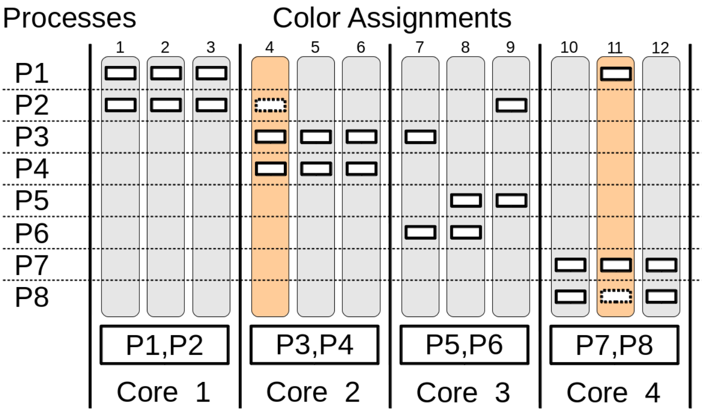
**procedure** pick_victims(num)
$victims \leftarrow \phi$
**while** $num > 0$
  **if** hasRemote()
    $victims += pick\_hottest\_remote()$
  **else**
    $victims += pick\_hottest\_local()$
  $num \leftarrow num - 1$
**return** $victims$
**end procedure**

- pick_hottest_remote: pick a color in a remote cache section, with the largest global hotness

# Cache Re-partitioning

**procedure** pick_victims(num)
*victims* $\leftarrow \phi$
**while** *num* $> 0$
  **if** hasRemote()
    *victims* $+=$
      *pick_hottest_remote*()
  **else**
    *victims* $+=$
      *pick_hottest_local*()
  *num* $\leftarrow$ *num* $- 1$
**return** *victims*
**end procedure**

- pick_hottest_remote:
  pick a color in a remote cache section, with the largest global hotness
- pick_hottest_local:
  pick a color in the local cache section, with the largest remote hotness


Processes
Color Assignments
1
2
3
4
5
6
7
8
9
10
11
12
P1
P2
P3
P4
P5
P6
P7
P8
P1,P2
P3,P4
P5,P6
P7,P8
Core 1
Core 2
Core 3
Core 4

# Recoloring Engine

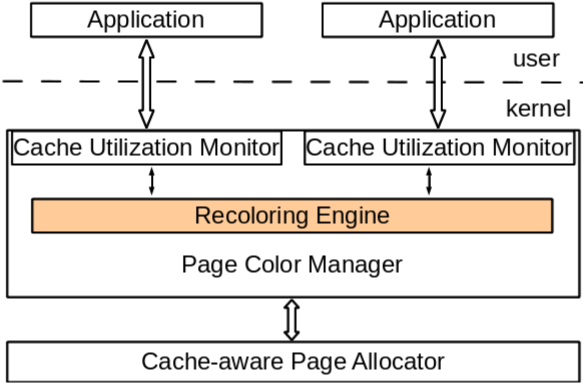



Figure : COLORIS Architecture

# Recoloring Engine

- Shrinkage: lazy recoloring [Lin et al:08]
  - look for pages of specific colors that are going to be taken away and clear the present bits of their page table entries
  - an unused bit is set to indicate recoloring needed
  - allocate new pages from assigned colors in a round-robin manner

# Recoloring Engine

- Expansion

# Recoloring Engine

- Expansion
  - Selective Moving:
    Assuming n-way set associative cache, scan the whole page table and recolor one in every $n + 1$ pages of the same color

# Recoloring Engine

- Expansion
  - Selective Moving:
    Assuming n-way set associative cache, scan the whole page table and recolor one in every $n + 1$ pages of the same color
  - Redistribution:
    - clear the access bit of every page table entry
    - after a fixed time window, scan the page table again
    - apply lazy recoloring to entries with access bits set

# Evaluation

- Experiment setup

  Dell PowerEdge T410 machine with quad-core Intel Xeon E5506 2.13GHz processor, 8GB RAM, shared 4MB 16-way set-associative L3 cache

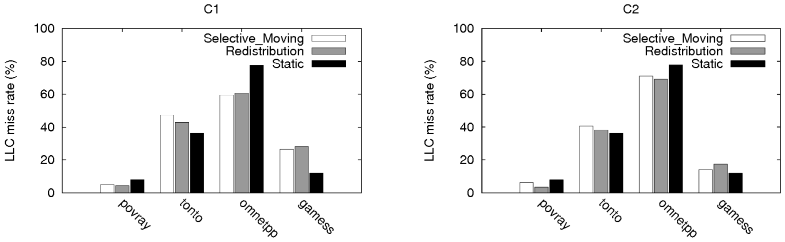
- Benchmark: SPEC CPU2006

# Evaluation

- Dynamic partitioning for QoS
- Four benchmarks run together for an hour

# Evaluation

- Dynamic partitioning for QoS
- Four benchmarks run together for an hour
  In C1 and C2, *HighThreshold* is 65% and 75% respectively

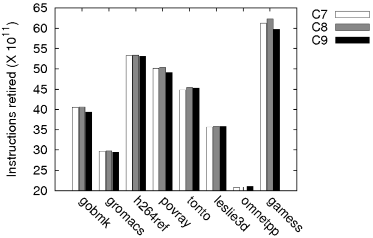


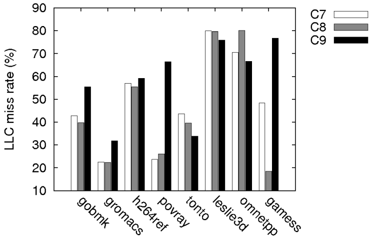
# Evaluation

- COLORIS in over-committed systems
- Eight applications run together, with each two pinned to a core

# Evaluation

- COLORIS in over-committed systems
- Eight applications run together, with each two pinned to a core
  C7: Dynamic; C8: Static; C9: None (Linux default)



# Conclusion

- Designed a memory sub-system that provides static/dynamic cache partitioning capabilities
- Proposed a scheme for managing page colors, which works for over-committed systems
- Studied two page selection policies for effective page recoloring

The End

Thank you!