COLORIS: A Dynamic Cache Partitioning System Using Page Coloring

Ying Ye, Richard West, Zhuoqun Cheng, Ye Li

Computer Science Department
Boston University
Overview

1 Background
2 Contribution
3 COLORIS Design
4 Evaluation
5 Conclusion
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.
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

Performance isolation needed for QoS-demanding systems.
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

App 1
1,2

App 2
3,4

Cache
1 2 3 4
Page Coloring

App 1

1,2

App 2

3,4

Cache

1 2 3 4

App 1

1,2,3

App 2

4

Cache

1 2 3 4

Arrow: from App 1 to App 2
Dynamic Partitioning

- When to re-partition LLC?
- Adjusting initial partition
- What is the right partition size?
- How to recolor memory?
- Heavy overhead; inefficient use
- How to work with over-committed systems?
Dynamic Partitioning

- When to re-partition LLC?
Dynamic Partitioning

- When to re-partition LLC?
  - phase change; adjusting initial partition

- How to recolor memory?

- heavy overhead; inefficient use

- How to work with over-committed systems?
Dynamic Partitioning

- When to re-partition LLC?
  - phase change; adjusting initial partition

- What is the right partition size?
Dynamic Partitioning

- When to re-partition LLC?
  - phase change; adjusting initial partition

- What is the right partition size?

- How to recolor memory?
Dynamic Partitioning

- When to re-partition LLC?
  - phase change; adjusting initial partition

- What is the right partition size?

- How to recolor memory?
  - heavy overhead; inefficient use
Dynamic Partitioning

- When to re-partition LLC?
  - phase change; adjusting initial partition

- 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 propose an efficient page recoloring framework in the Linux kernel, called COLORIS (COLOR ISolation).
Figure: COLORIS Architecture
Figure: Color-aware 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
<table>
<thead>
<tr>
<th>Processes</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1, P2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Color Assignments</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3, P4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Core 3            |    |    |    |    |    |    |    |    |    |    |    |    |
| P5, P6            |    |    |    |    |    |    |    |    |    |    |    |    |

| Core 4            |    |    |    |    |    |    |    |    |    |    |    |    |
| P7, P8            |    |    |    |    |    |    |    |    |    |    |    |    |
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

![Diagram showing color assignments for processes P1 to P8 across cores 1 to 4]
Cache Utilization Monitor

Figure: COLORIS Architecture
Cache Utilization Monitor

- Measures cache usage of individual applications:
  - \( \text{cache miss rate} = \frac{\text{misses}}{\text{accesses}} \)
Cache Utilization Monitor

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

- 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
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
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

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

- **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
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)

\[
\text{victims} \leftarrow \emptyset
\]

**while** num > 0

**if** hasRemote()

\[
\text{victims} \leftarrow \text{pick_hottest_remote}()
\]

**else**

\[
\text{victims} \leftarrow \text{pick_hottest_local}()
\]

\[
\text{num} \leftarrow \text{num} - 1
\]

**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)

\[ \textit{victims} \leftarrow \emptyset \]

**while** num > 0

\[ \textit{if} \ \text{hasRemote()} \]

\[ \textit{victims} + = \]

\[ \text{pick_hottest_remote()} \]

\[ \text{else} \]

\[ \textit{victims} + = \]

\[ \text{pick_hottest_local()} \]

\[ \text{num} \leftarrow \text{num} - 1 \]

**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

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
<th>Core 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1, P2</td>
<td>P3, P4</td>
<td>P5, P6</td>
<td>P7, P8</td>
</tr>
</tbody>
</table>

Color Assignments

1  2  3  4  5  6  7  8  9  10  11  12
Figure: COLORIS Architecture
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

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
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 and apply lazy recoloring to entries with access bits set.
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
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

In C1 and C2, HighThreshold is 65% and 75% respectively.
Dynamic partitioning for QoS

Four benchmarks run together for an hour
In C1 and C2, HighThreshold is 65% and 75% respectively
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)

![Graph showing Instructions retired and LLC miss rate](image-url)
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!