Colorama: Architectural Support for Data-Centric Synchronization

Luis Ceze, Pablo Montesinos, Christoph von Praun† and Josep Torrellas

University of Illinois at Urbana-Champaign
†IBM Research

http://iacoma.cs.uiuc.edu

HPCA, February 2007
Challenge: Simplify Parallel Programming
Challenge: Simplify Parallel Programming

• Thread synchronization is very challenging
Challenge: Simplify Parallel Programming

• Thread synchronization is very challenging

• Lock-based synchronization:
  • coarse-grain: low performance; fine-grain: error-prone
Challenge: Simplify Parallel Programming

• Thread synchronization is very challenging

• Lock-based synchronization:
  • coarse-grain: low performance; fine-grain: error-prone

• Transactional Memory-based synchronization:
  • programmability ~ coarse-grain locking; performance ~ fine-grain locking
Challenge: Simplify Parallel Programming

• Thread synchronization is very challenging

• Lock-based synchronization:
  • coarse-grain: low performance; fine-grain: error-prone

• Transactional Memory-based synchronization:
  • programmability ~ coarse-grain locking; performance ~ fine-grain locking

• Locks and TM shortcoming:
  • programmer needs to annotate code around every potential access to shared data
Challenge: Simplify Parallel Programming

- Thread synchronization is very challenging

- Lock-based synchronization:
  - coarse-grain: low performance; fine-grain: error

- Transactional Memory-based synchronization:
  - programmability ~ coarse-grain locking; performance ~ fine-grain locking

- Locks and TM shortcoming:
  - programmer needs to annotate code around every potential access to shared data

Need to make parallel programming much simpler:
Challenge: Simplify Parallel Programming

• Thread synchronization is very challenging

• Lock-based synchronization:
  - coarse-grain: low performance; fine-grain: error prone

• Transactional Memory-based synchronization:
  - programmability ~ coarse-grain locking; performance ~ fine-grain locking

• Locks and TM shortcoming:
  • programmer needs to annotate code around every potential access to shared data

Need to make parallel programming much simpler:

Data-Centric Synchronization (DCS)
<table>
<thead>
<tr>
<th>Conventional: Code-Centric Synchronization (CCS)</th>
<th>Colorama: Data-Centric Synchronization (DCS)</th>
</tr>
</thead>
</table>

Luis Ceze

Colorama

February 2007
<table>
<thead>
<tr>
<th>Conventional: Code-Centric Synchronization (CCS)</th>
<th>Colorama: Data-Centric Synchronization (DCS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare A,B,C</td>
<td></td>
</tr>
</tbody>
</table>
### Conventional: Code-Centric Synchronization (CCS)

```plaintext
declare A, B, C
```

![A B C](image)

### Colorama: Data-Centric Synchronization (DCS)
<table>
<thead>
<tr>
<th>Conventional: Code-Centric Synchronization (CCS)</th>
<th>Colorama: Data-Centric Synchronization (DCS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare A,B,C</td>
<td></td>
</tr>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

*Luis Ceze*  
*February 2007*
Conventional: Code-Centric Synchronization (CCS)

- declare A, B, C
- lock L1
- ld A
- st A
- st B
- unlock L1

Colorama: Data-Centric Synchronization (DCS)
<table>
<thead>
<tr>
<th>Conventional: Code-Centric Synchronization (CCS)</th>
<th>Colorama: Data-Centric Synchronization (DCS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>declare A, B, C</code></td>
<td></td>
</tr>
<tr>
<td><code>lock L1</code></td>
<td></td>
</tr>
<tr>
<td><code>ld A</code></td>
<td></td>
</tr>
<tr>
<td><code>st A</code></td>
<td></td>
</tr>
<tr>
<td><code>st B</code></td>
<td></td>
</tr>
<tr>
<td><code>unlock L1</code></td>
<td></td>
</tr>
<tr>
<td><code>lock L2</code></td>
<td></td>
</tr>
<tr>
<td><code>ld C</code></td>
<td></td>
</tr>
<tr>
<td><code>st C</code></td>
<td></td>
</tr>
<tr>
<td><code>unlock L2</code></td>
<td></td>
</tr>
</tbody>
</table>
Conventional: Code-Centric Synchronization (CCS)

- declare A, B, C
- lock L1
- ld A
- st A
- st B
- unlock L1
- lock L2
- ld C
- st C
- unlock L2

Colorama: Data-Centric Synchronization (DCS)
<table>
<thead>
<tr>
<th>Conventional: Code-Centric Synchronization (CCS)</th>
<th>Colorama: Data-Centric Synchronization (DCS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare A,B,C</td>
<td>declare A,B,C</td>
</tr>
<tr>
<td><strong>lock L1</strong></td>
<td></td>
</tr>
<tr>
<td>ld A</td>
<td></td>
</tr>
<tr>
<td>st A</td>
<td></td>
</tr>
<tr>
<td>st B</td>
<td></td>
</tr>
<tr>
<td><strong>unlock L1</strong></td>
<td></td>
</tr>
<tr>
<td><strong>lock L2</strong></td>
<td></td>
</tr>
<tr>
<td>ld C</td>
<td></td>
</tr>
<tr>
<td>st C</td>
<td></td>
</tr>
<tr>
<td><strong>unlock L2</strong></td>
<td></td>
</tr>
</tbody>
</table>
Conventional: Code-Centric Synchronization (CCS)

- declare A, B, C
- lock L1
- ld A
- st A
- st B
- unlock L1
- lock L2
- ld C
- st C
- unlock L2

Colorama: Data-Centric Synchronization (DCS)

- declare A, B, C
## Conventional: Code-Centric Synchronization (CCS)

1. `declare A, B, C`  
2. `lock L1`  
3. `ld A`  
4. `st A`  
5. `st B`  
6. `unlock L1`

## Colorama: Data-Centric Synchronization (DCS)

1. `declare A, B, C`  
2. `color red A, B`  
3. `lock L1`  
4. `ld A`  
5. `st A`  
6. `st B`  
7. `unlock L1`
Conventional: Code-Centric Synchronization (CCS)

```plaintext
declare A, B, C

lock L1
ld A
st A
st B
unlock L1

lock L2
ld C
st C
unlock L2
```

Colorama: Data-Centric Synchronization (DCS)

```plaintext
declare A, B, C

color red A, B

color blue C
```
Conventional: Code-Centric Synchronization (CCS)

```
declare A, B, C
lock L1
ld A
st A
st B
unlock L1
lock L2
ld C
st C
unlock L2
```

Colorama: Data-Centric Synchronization (DCS)

```
declare A, B, C
color red A, B
color blue C
```
Conventional: Code-Centric Synchronization (CCS)

- declare A, B, C
- lock L1
- ld A
- st A
- st B
- unlock L1
- lock L2
- ld C
- st C
- unlock L2

Colorama: Data-Centric Synchronization (DCS)

- declare A, B, C
- color red A, B
- color blue C
- ld A
- st A
- st B
Conventional: Code-Centric Synchronization (CCS)

```
declare A, B, C
lock L1
ld A
st A
st B
unlock L1

lock L2
ld C
st C
unlock L2
```

Colorama: Data-Centric Synchronization (DCS)

```
declare A, B, C
color red A, B

A B C
```

```
lock L2
ld C
st C
```

```
unlock L2
```

```
color blue C
```
Conventional: Code-Centric Synchronization (CCS)

- `declare A, B, C`
- `lock L1`  
  `ld A`
  `st A`
  `st B`
  `unlock L1`
- `lock L2`  
  `ld C`
  `st C`
  `unlock L2`

Colorama: Data-Centric Synchronization (DCS)

- `declare A, B, C`
- `color red A, B`
- `color blue C`
- `ld A`
  `st A`
  `st B`
- System infers and enforces critical sections
- `ld C`
  `st C`
DCS Reduces Programmer Effort
DCS Reduces Programmer Effort

code-centric | data-centric
# DCS Reduces Programmer Effort

<table>
<thead>
<tr>
<th></th>
<th>Code-centric</th>
<th>Data-centric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasoning</td>
<td>Non-local</td>
<td>Local</td>
</tr>
</tbody>
</table>
## DCS Reduces Programmer Effort

<table>
<thead>
<tr>
<th></th>
<th>code-centric</th>
<th>data-centric</th>
</tr>
</thead>
<tbody>
<tr>
<td>reasoning</td>
<td>non-local</td>
<td>local</td>
</tr>
<tr>
<td>critical sections</td>
<td>explicitly defined</td>
<td>inferred by system</td>
</tr>
</tbody>
</table>
Intuition: Local reasoning improves programmer productivity by reducing chances of mistakes.

<table>
<thead>
<tr>
<th></th>
<th>code-centric</th>
<th>data-centric</th>
</tr>
</thead>
<tbody>
<tr>
<td>reasoning</td>
<td>non-local</td>
<td>local</td>
</tr>
<tr>
<td>critical sections</td>
<td>explicitly defined</td>
<td>inferred by system</td>
</tr>
</tbody>
</table>
Example from *mysql*
Example from *mysql*
Example from *mysql*

- All header protected by global lock
- Global lock used in 29 sites
Example from *mysql*

- all header protected by global lock
- global lock used in 29 sites
- info protected by per-record lock
- per-record lock used in 14 sites
Example from *mysql*

**CCS**
- all header protected by global lock
  - global lock used in 29 sites
- info protected by per-record lock
  - per-record lock used in 14 sites
Example from *mysql*

**CCS**
- All header protected by global lock
- Global lock used in 29 sites
- Info protected by per-record lock
- Per-record lock used in 14 sites

**DCS**
- Header fields same color
- Each info different color
Software DCS
Software DCS

- Software-only DCS concurrently developed [Vaziri PoPL’06]
  - for object-oriented languages (Java)
Software DCS

• Software-only DCS concurrently developed [Vaziri PoPL’06]
  • for object-oriented languages (Java)

• Needs whole-program analysis
Software DCS

• Software-only DCS concurrently developed [Vaziri PoPL’06]
  • for object-oriented languages (Java)

• Needs whole-program analysis

• Impractical in C/C++
  • Pointer aliasing
  • Compiler often does not have access to whole code
Colorama: Architecture Support for DCS
Colorama: Architecture Support for DCS

• Main advantage: HW can cheaply watch all memory refs
Colorama: Architecture Support for DCS

• Main advantage: HW can cheaply watch all memory refs
• *Enter* critical section when colored data is touched
Colorama: Architecture Support for DCS

• Main advantage: HW can cheaply watch all memory refs

• *Enter* critical section when colored data is touched
  • HW checks the color of every memory access
Colorama: Architecture Support for DCS

• Main advantage: HW can cheaply watch all memory refs
• Enter critical section when colored data is touched
  • HW checks the color of every memory access
  • Interface to assign colors to shared data (memory tagging)
Colorama: Architecture Support for DCS

• Main advantage: HW can cheaply watch all memory refs

• Enter critical section when colored data is touched
  • HW checks the color of every memory access
  • Interface to assign colors to shared data (memory tagging)

• Exit critical section using an Exit Policy
Colorama: Architecture Support for DCS

• Main advantage: HW can cheaply watch all memory refs

• *Enter* critical section when colored data is touched
  • HW checks the color of every memory access
  • Interface to assign colors to shared data (memory tagging)

• *Exit* critical section using an Exit Policy
  • HW provides flexible primitives to exit critical sections
Colorama: Architecture Support for DCS

- Main advantage: HW can cheaply watch all memory refs

- *Enter* critical section when colored data is touched
  - HW checks the color of every memory access
  - Interface to assign colors to shared data (memory tagging)

- *Exit* critical section using an Exit Policy
  - HW provides flexible primitives to exit critical sections
  - Compiler inserts the code that implements the policy
Colorama: Architecture Support for DCS

• Main advantage: HW can cheaply watch all memory refs

• Enter critical section when colored data is touched
  • HW checks the color of every memory access
  • Interface to assign colors to shared data (memory tagging)

• Exit critical section using an Exit Policy
  • HW provides flexible primitives to exit critical sections
  • Compiler inserts the code that implements the policy
  • Programmer must know (and follow) the Exit Policy
Colorama: Architecture Support for DCS

- **Main advantage**: HW can cheaply watch all memory refs

- **Enter critical section** when colored data is touched
  - HW checks the color of every memory access
  - Interface to assign colors to shared data (memory tagging)

- **Exit critical section** using an Exit Policy
  - HW provides flexible primitives to exit critical sections
  - Compiler inserts the code that implements the policy
  - Programmer must know (and follow) the Exit Policy

- **Structures to keep track of color ownership**
Architecture Components
Architecture Components
Architecture Components

Start Address | End Address | ColorID
---|---|---
Palette

Shared

ColorID\textsubscript{i}

Owned Colors Array

Color Acquire Bitmap Register (CAB)

Color Release Bitmap Register (CRB)

Thread Color Status

Per Thread
Colorama Operation Example

Palette

Owned Colors

Array
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

Palette
A red
B red
C red
E blue
F blue

Owned Colors
Array
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

thread 1

Palette
A red
B red
C red
E blue
F blue

Owned Colors Array
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

Palette
A red
B red
C red
E blue
F blue

Owned Colors
Array
thread 1

ld A
Colorama Operation Example

```
<table>
<thead>
<tr>
<th>Palette</th>
<th>Owned Colors Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>A red</td>
<td></td>
</tr>
<tr>
<td>B red</td>
<td></td>
</tr>
<tr>
<td>C red</td>
<td></td>
</tr>
<tr>
<td>E blue</td>
<td></td>
</tr>
<tr>
<td>F blue</td>
<td></td>
</tr>
</tbody>
</table>
```

color A red
color B red
color C red
color E blue
color F blue

thread 1

ld A \* trap to user-level handler, start red critical section
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

Palette
A red
B red
C red
E blue
F blue

Owned Colors
Array
red

thread 1

ld A ✓ trap to user-level handler, start red critical section
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

Palette
A red
B red
C red
E blue
F blue

Owned Colors Array
red

thread 1

ld A \(\triangleright\) trap to user-level handler, start red critical section
st B
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

Palette
A red
B red
C red
E blue
F blue

Owned Colors
Array
red

thread 1

ld A \checkmark\ trap to user-level handler, start red critical section
st B
st C
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

Palette
A red
B red
C red
E blue
F blue

Owned Colors
Array
red

thread 1

ld A ✓ trap to user-level handler, start red critical section
st B
st C

 ✓ trap to user-level handler, exit red critical section
Colorama Operation Example

- Palette:
  - A red
  - B red
  - C red
  - E blue
  - F blue

- Owned Colors Array:

```
thread 1
```

- Color assignments:
  - color A red
  - color B red
  - color C red
  - color E blue
  - color F blue

- Execution:
  - ld A \(\triangleright\) trap to user-level handler, start red critical section
  - st B
  - st C
  - \(\triangleright\) trap to user-level handler, exit red critical section
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

Palette
A red
B red
C red
E blue
F blue

Owned Colors Array
thread 1

1d A ✓ trap to user-level handler, start red critical section
st B
st C

✓ trap to user-level handler, exit red critical section
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

Palette
A red
B red
C red
E blue
F blue

Owned Colors Array
blue

thread 1

ld A ✓trap to user-level handler, start red critical section
st B
st C ✓trap to user-level handler, exit red critical section
st E ✓trap to user-level handler, start blue critical section
st F
Colorama Operation Example

```
<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>red</td>
</tr>
<tr>
<td>B</td>
<td>red</td>
</tr>
<tr>
<td>C</td>
<td>red</td>
</tr>
<tr>
<td>E</td>
<td>blue</td>
</tr>
<tr>
<td>F</td>
<td>blue</td>
</tr>
</tbody>
</table>
```

Palette:
```
<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>red</td>
</tr>
<tr>
<td>B</td>
<td>red</td>
</tr>
<tr>
<td>C</td>
<td>red</td>
</tr>
<tr>
<td>E</td>
<td>blue</td>
</tr>
<tr>
<td>F</td>
<td>blue</td>
</tr>
</tbody>
</table>
```

Owned Colors Array:
```
<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>blue</td>
</tr>
</tbody>
</table>
```

Thread 1:
```
ld  A  ✓ trap to user-level handler, start red critical section
st  B
st  C

  ✓ trap to user-level handler, exit red critical section

st  E  ✓ trap to user-level handler, start blue critical section
st  F

  ✓ trap to user-level handler, exit blue critical section
```
Colorama Operation Example

color A red
color B red
color C red
color E blue
color F blue

Palette
A red
B red
C red
E blue
F blue

Owned Colors Array

thread 1

ld A  ✓trap to user-level handler, start red critical section
st B
st C  ✓trap to user-level handler, exit red critical section

st E  ✓trap to user-level handler, start blue critical section
st F  ✓trap to user-level handler, exit blue critical section
Colorama Operation Example

```
Thread 1

<table>
<thead>
<tr>
<th>Color</th>
<th>Palette</th>
<th>Owned Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A red</td>
<td>A red</td>
<td></td>
</tr>
<tr>
<td>B red</td>
<td>B red</td>
<td></td>
</tr>
<tr>
<td>C red</td>
<td>C red</td>
<td>E blue</td>
</tr>
<tr>
<td>E blue</td>
<td>E blue</td>
<td>F blue</td>
</tr>
<tr>
<td>F blue</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
ld A   ✓ trap to user-level handler, start red critical section
st B
st C   ✓ trap to user-level handler, exit red critical section

st E   ✓ trap to user-level handler, start blue critical section
st F   ✓ trap to user-level handler, exit blue critical section
```

Inferred critical sections

Luis Ceze

Colorama
February 2007

i-ACOMA group
Colorama Operation Example

Underlying synchronization substrate: TM or locks. Focusing on TM.

Palette

```
<table>
<thead>
<tr>
<th>owned colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A red</td>
</tr>
<tr>
<td>B red</td>
</tr>
<tr>
<td>C red</td>
</tr>
<tr>
<td>E blue</td>
</tr>
<tr>
<td>F blue</td>
</tr>
</tbody>
</table>
```

Owned Colors Array

```
[ ]
```

Owned Colors Array

```
[ ]
```

Thread 1

```
thread 1
```

Trap to user-level handler, exit red critical section

```
\[ \text{trap to user-level handler, exit red critical section} \]
```

Trap to user-level handler, start blue critical section

```
\[ \text{trap to user-level handler, start blue critical section} \]
```

Trap to user-level handler, exit blue critical section

```
\[ \text{trap to user-level handler, exit blue critical section} \]
```
Exiting a Critical Section
Exiting a Critical Section

• Knowing when to *end* a critical section is difficult
Exiting a Critical Section

• Knowing when to *end* a critical section is difficult

• Optimal place undecidable
Exiting a Critical Section

• Knowing when to end a critical section is difficult

• Optimal place undecidable

• Solution is to rely on an agreed Exit Policy
  • Exit Policy becomes part of the programming model
Exiting a Critical Section

- Knowing when to end a critical section is difficult
- Optimal place undecidable
- Solution is to rely on an agreed Exit Policy
  - Exit Policy becomes part of the programming model
- We use:
  - Return of subroutine where the critical section started [Vaziri’06, Wang’06]
Exit Policy
Exit Policy

void foo1()
{
  ...
  ...
  A = ...
  ...
  ...
  ...
}

ColorID
A

critical
section
void foo1() {
    ...
    ...
    A = ...
    ...
    ...
    ...
    B = ...
    ...
    ...
    ...
    }

void foo1() {
    ...
    ...
    A = ...
    ...
    ...
    ...
    B = ...
    ...
    ...
    ...
    }
Exit Policy

```c
void foo1()
{
    ...
    ...
    A = ...
    ...
    ...
    ...
}
```

```c
void foo1()
{
    ...
    ...
    A = ...
    foo2();
    ...
    ...
    B = ...
    ...
    ...
}
```

```c
void foo2()
{
    ...
    B = ...
    ...
    ...
}
```
Architecture Components

Start Address  End Address  ColorID

Palette

Shared

Thread Color Status

Owned Colors Array

ColorID_i

Color Acquire Bitmap Register (CAB)

Color Release Bitmap Register (CRB)

Per Thread
Detailed Operation

\[ f() \]

Color Acquire Bitmap (CAB register)

Owned Colors Array
Detailed Operation

f()

1d Z

Color Acquire Bitmap (CAB register)

Owned Colors Array
Detailed Operation

\[ f() \]

\[ \text{ld } Z \]

Color Acquire Bitmap (CAB register)

\[ 1 \]

Owned Colors Array

Blue
Detailed Operation

```
f()  
1d Z  
call g()  
```

```
Color Acquire Bitmap  
(CAB register)  

1

Owned Colors  
Array  

Blue
```
Detailed Operation

\[
\begin{align*}
&\text{f()}
\quad \downarrow \\
&\text{ld } z
\quad \downarrow \\
&\text{call g()}
\quad \downarrow \\
&\text{stack } \leftarrow \text{CAB}
\end{align*}
\]

Color Acquire Bitmap (CAB register)

Owned Colors Array

Blue
**Detailed Operation**

```
\begin{align*}
\text{f()} & \rightarrow \text{ld z} \\
\text{call g()} & \rightarrow \text{stack} \leftarrow \text{CAB} \\
& \rightarrow \text{CAB} \leftarrow 0
\end{align*}
```

**Color Acquire Bitmap (CAB register):**

```
1
```

**Owned Colors Array:**

- Blue

---

---
Detailed Operation

\[
\begin{align*}
&f() \\
&\text{ld } z \\
&\text{call } g() \\
&\text{stack } \leftarrow \text{CAB} \\
&\text{CAB } \leftarrow 0
\end{align*}
\]

Color Acquire Bitmap (CAB register)

Owned Colors Array

Blue
Detailed Operation

1. `f()`
2. `ld z`
3. `call g()`
4. `stack ← CAB`
5. `CAB ← 0`
6. `...`

**Color Acquire Bitmap (CAB register)**

**Owned Colors Array**
- Blue
Detailed Operation

\[ f() \]
\[ \text{call } g() \]
\[ \text{ld } Z \]
\[ \text{ld } A \]

Color Acquire Bitmap (CAB register)

Owned Colors Array

Blue
Detailed Operation

\[ \text{f() \rightarrow ld \ Z \ \text{call g()} \rightarrow } \]
\[ \text{stack } \leftarrow \text{CAB} \]
\[ \text{CAB } \leftarrow 0 \]
\[ \ldots \]
\[ \text{ld A} \]

Color Acquire Bitmap (CAB register)

Owned Colors Array

Blue
Red
Detailed Operation

\[
\begin{align*}
\text{f}() & \quad \text{ld } Z \\
\text{call } g() & \quad \text{stack} \leftarrow \text{CAB} \\
& \quad \text{CAB} \leftarrow 0 \\
& \quad \ldots \\
& \quad \text{ld } A \\
& \quad \text{ld } T
\end{align*}
\]

Color Acquire Bitmap (CAB register)

Owned Colors Array

Blue
Red
Detailed Operation

\[ f() \]
\[ \text{call } g() \]
\[ \text{ld } Z \]
\[ \text{stack } \leftarrow \text{CAB} \]
\[ \text{CAB } \leftarrow 0 \]
\[ \ldots \]
\[ \text{ld } A \]
\[ \text{ld } T \]
\[ \text{ld } B \]

Color Acquire Bitmap (CAB register)

Owned Colors Array

Blue
Red
Detailed Operation

\[ f() \]
\[ \text{ld } Z \]
\[ \text{call } g() \]

\[ \text{stack } \leftarrow \text{CAB} \]
\[ \text{CAB } \leftarrow 0 \]
\[ \ldots \]
\[ \text{ld } A \]
\[ \text{ld } T \]
\[ \text{ld } B \]

Color Acquire Bitmap (CAB register)

Owned Colors Array

- Blue
- Red
- Green
Detailed Operation

\[ f() \]

\[ \text{ld } Z \]
\[ \text{call } g() \]

\[ \text{stack } \leftarrow \text{CAB} \]
\[ \text{CAB } \leftarrow 0 \]
\[ ... \]
\[ \text{ld } A \]
\[ \text{ld } T \]
\[ \text{ld } B \]
\[ ... \]

Color Acquire Bitmap (CAB register)

\[ \begin{array}{c}
1 \\
1 
\end{array} \]

Owned Colors Array

- Blue
- Red
- Green
Detailed Operation

\[ f() \]

\[ \text{ld } Z \]

\[ \text{call } g() \]

\[ \text{stack } \leftarrow \text{CAB} \]

\[ \text{CAB } \leftarrow 0 \]

\[ \ldots \]

\[ \text{ld } A \]

\[ \text{ld } T \]

\[ \text{ld } B \]

\[ \ldots \]

\[ \text{CRB } \leftarrow \text{CAB} \]

\[ \text{Color Acquire Bitmap (CAB register)} \]

\[ 1 \quad 1 \]

\[ \text{Owned Colors Array} \]

\[ \text{Blue} \]

\[ \text{Red} \]

\[ \text{Green} \]
Detailed Operation

\[
\begin{align*}
f() & \quad \text{Color Acquire Bitmap (CAB register)} \\
\downarrow \quad \text{call } g() & \quad \text{ Owned Colors Array} \\
\text{ld } Z & \quad \text{Blue} \\
\text{ld } A & \quad \text{Red} \\
\text{ld } T & \quad \text{Green} \\
\text{ld } B & \quad \text{ins critical sections Red and Green} \\
\text{CRB} & \quad \text{System stack}\end{align*}
\]

\[
\begin{align*}
\text{stack} & \leftarrow \text{CAB} \\
\text{CAB} & \leftarrow 0 \\
\ldots & \\
\text{ld } A & \\
\text{ld } T & \\
\text{ld } B & \\
\ldots & \\
\text{CRB} & \leftarrow \text{CAB}
\end{align*}
\]
Detailed Operation

\[ f() \]
\[ \text{ld } z \]
\[ \text{call g()} \]
\[ \text{stack } \leftarrow \text{CAB} \]
\[ \text{CAB } \leftarrow 0 \]
\[ \ldots \]
\[ \text{ld A} \]
\[ \text{ld T} \]
\[ \text{ld B} \]
\[ \ldots \]
\[ \text{CRB } \leftarrow \text{CAB} \quad \checkmark \text{exits critical sections Red and Green} \]
Detailed Operation

```
f()  
  ld Z  
  call g()  
  
  stack ← CAB  
  CAB ← 0  
  ...  
  ld A  
  ld T  
  ld B  
  ...  
  CRB ← CAB    ✓ exits critical sections Red and Green  
  CAB ← stack
```

Color Acquire Bitmap
(CAB register)
Detailed Operation

\[ f() \]
\[ \text{ld } z \]
\[ \text{call } g() \]
\[ \text{ld } A \]
\[ \text{ld } T \]
\[ \text{ld } B \]
\[ \text{ld } Z \]
\[ \text{stack } \leftarrow \text{CAB} \]
\[ \text{CAB } \leftarrow 0 \]
\[ \ldots \]
\[ \text{CAB } \leftarrow \text{stack} \]

Color Acquire Bitmap (CAB register)

Owned Colors Array

- Blue

\( \text{CRB } \leftarrow \text{CAB} \)

✓ exits critical sections Red and Green
Detailed Operation

f()

ld Z

call g()

stack ← CAB
CAB ← 0
...
ld A
ld T
ld B
...
CRB ← CAB ✓ exits critical sections Red and Green
CAB ← stack
ret

Color Acquire Bitmap (CAB register)

Owned Colors Array

Blue
Detailed Operation

\[ f() \]
\[ \text{ld } z \]
\[ \text{call } g() \]

\[
\begin{align*}
\text{stack} &\leftarrow \text{CAB} \\
\text{CAB} &\leftarrow 0 \\
&\ldots \\
\text{ld } A \\
\text{ld } T \\
\text{ld } B \\
&\ldots \\
\text{CRB} &\leftarrow \text{CAB} \\
\text{CAB} &\leftarrow \text{stack} \\
\text{ret}
\end{align*}
\]

```
Color Acquire Bitmap (CAB register)
```

```
\[
\begin{array}{c}
1 \\
\end{array}
\]
```

```
Owned Colors Array
```

```
\[
\text{Blue}
\]
```

- subroutine prologue/epilogue inserted by the compiler
- exits critical sections Red and Green
Detailed Operation

```plaintext
Detailed Operation

f()  
|  ld  z  |
|  call  g()  |
|  stack  →  CAB  |
|  CAB  ←  0  |

Color Acquire Bitmap  
(CAB register)  

Owned Colors  
Array  

Blue

Hardware is generic: exit policy configurable and specified by SW.

CRB  ←  CAB  
CAB  ←  stack  
ret

✓ exits critical sections Red and Green

Luis Ceze
Colorama  February 2007
```
Palette Implementation

Mondrian Memory Protection [Witchel ASPLOS’02]

MMP with the Palette extensions
Why Multiple Colors?
Why Multiple Colors?

• Intuitive way of nesting inferred transactions
Why Multiple Colors?

• Intuitive way of nesting inferred transactions
• Conveys more information on how shared data is used
Why Multiple Colors?

• Intuitive way of nesting inferred transactions

• Conveys more information on how shared data is used
  • Allows future exit policy refinement
Why Multiple Colors?

• Intuitive way of nesting inferred transactions

• Conveys more information on how shared data is used
  • Allows future exit policy refinement
  • More opportunity for optimizations
Why Multiple Colors?

• Intuitive way of nesting inferred transactions

• Conveys more information on how shared data is used
  • Allows future exit policy refinement
  • More opportunity for optimizations
    • e.g., assume transactions of different colors do not collide
Why Multiple Colors?

• Intuitive way of nesting inferred transactions

• Conveys more information on how shared data is used
  • Allows future exit policy refinement
  • More opportunity for optimizations
    • e.g., assume transactions of different colors do not collide

• Lock-based implementation: each color maps to a lock
Using Colorama to Debug Conventional Code
Using Colorama to Debug Conventional Code

• Programmer uses explicit transactions AND colors data
Using Colorama to Debug Conventional Code

- Programmer uses explicit transactions **AND** colors data
- The system can monitor:
Using Colorama to Debug Conventional Code

• Programmer uses explicit transactions AND colors data

• The system can monitor:
  ➔ colored data should only be accessed inside transactions
Using Colorama to Debug Conventional Code

• Programmer uses explicit transactions **AND** colors data

• The system can monitor:
  - ➞ colored data should only be accessed inside transactions
  - ➞ transactions of different colors should not collide
Using Colorama to Debug Conventional Code

• Programmer uses explicit transactions AND colors data

• The system can monitor:
  ➔ colored data should only be accessed inside transactions
  ➔ transactions of different colors should not collide
  ➔ transactions should typically access a single color
    - if multiple colors are accessed, nesting could be recommended
Evaluation
Evaluation

• No large Colorama programs (yet)
Evaluation

• No large Colorama programs (yet)

• Evaluation consists in detailed profiling of open-source parallel programs
  • Developed Pin tool to profile critical sections and memory allocation
  • Used MySQL, FireFox, aolserver, tuxracer, ...
Evaluation

• No large Colorama programs (yet)

• Evaluation consists in detailed profiling of open-source parallel programs
  • Developed Pin tool to profile critical sections and memory allocation
  • Used MySQL, FireFox, aolserver, tuxracer, ...

• Estimated Exit Policy suitability
  • Majority of critical sections entry/exit are inside same subroutine
Evaluation

• No large Colorama programs (yet)

• Evaluation consists in detailed profiling of open-source parallel programs
  • Developed Pin tool to profile critical sections and memory allocation
  • Used MySQL, FireFox, aolserver, tuxracer, ...

• Estimated Exit Policy suitability
  • Majority of critical sections entry/exit are inside same subroutine

• Estimated overheads
  • Palette increases memory footprint by about 2.5%
Also in the paper...

• More examples
• Detailed API
• Lock-based implementation issues
• Code compatibility issues
  • conditional wait, linking with non-Colorama code, ...
• Debugging Colorama code
• Additional characterization
Conclusion
Conclusion

• We presented a feasible implementation of DCS
Conclusion

• We presented a feasible implementation of DCS

• DCS can greatly simplify parallel programming
  • programmer only specifies the colors and follows a simple policy
  • the system, in return, guarantees consistency of shared data
Conclusion

• We presented a feasible implementation of DCS

• DCS can greatly simplify parallel programming
  • programmer only specifies the colors and follows a simple policy
  • the system, in return, guarantees consistency of shared data

• Current work:
  • Other exit policies under investigation
  • Deeper evaluation
Colorama: Architectural Support for Data-Centric Synchronization

Luis Ceze, Pablo Montesinos, Christoph von Praun† and Josep Torrellas

University of Illinois at Urbana-Champaign
†IBM Research

http://iacoma.cs.uiuc.edu

HPCA, February 2007