CASPAR: BREAKING SERIALIZATION IN LOCK-FREE MULTICORE SYNCHRONIZATION

Tanmay Gangwani †, Adam Morrison*, Josep Torrellas ‡
University of Illinois, Urbana-Champaign ‡
Technion – Israel Institute of Technology*
Concurrent Data Structures

push/pop

stack-top
Concurrent Data Structures

- **Lock-based Synchronization**

  ![Diagram of lock-based synchronization](image)
Concurrent Data Structures

• **Lock-free Synchronization**

1. read stack-top

2. read-modify-write stack-top

\[
\text{CAS} (A, X)
\]
CAS Failures

Adding node to shared stack

```c
void push () {
    Node *new_top = malloc();
    while(true) {
        old_top = stack-top;
        new_top->next = old_top;
        if(CAS(&stack-top, old_top, new_top))
            return;
    }
}
```
CAS Failures

Adding node to shared stack

void push () {
    Node *new_top = malloc();
    while(true) {
        old_top = stack-top;
        new_top->next = old_top;
        if(CAS(&stack-top, old_top, new_top))
            return;
    }
}
Adding node to shared stack

```c
void push() {
    Node *new_top = malloc();
    while (true) {
        old_top = stack-top;
        new_top->next = old_top;
        if (CAS(&stack-top, old_top, new_top))
            return;
    }
}
```
CAS Failures

Adding node to shared stack

```c
void push () {
    Node *new_top = malloc();
    while(true) {
        old_top = stack-top;
        new_top->next = old_top;
        if(CAS(&stack-top, old_top, new_top))
            return;
    }
}
```

CAS FAILURE
Load-to-CAS Atomicity

Adding node to shared stack

```c
void push () {  
    Node *new_top = malloc();  
    while(true) {  
        old_top = stack-top;  
        new_top->next = old_top;  
        if(CAS(&stack-top, old_top, new_top))  
            return;  
    }  
}
```
Scourge of Serialization

Atomic

Compute new
... Load old
... CAS(old, new)

Core A
Compute new
Ld old
CAS

Core B
Compute new
Ld old
Ld old
Stall
CAS

Core C
Compute new
Ld old
Ld old
Stall
CAS

Time
CASPAR reduces stalls in lock-free programs using two new techniques:

- Eager Forwarding
- Group Validation
CASPAR Fights Serialization

Baseline (Prior work)

- Hardware Queueing
  - Directory chains cores in hardware

Contributions

- Contribution I: Eager Forwarding
  - Directory facilitates passing of speculative values between cores

- Contribution II: Parallel Validation
  - Directory facilitates group-validation of multiple cores

- Uses cache line bit for locking
- Dynamic detection of contended CAS — No source code modifications
- Deadlock-free
Serialization in Hardware Queueing

Compute new
... Load old
... CAS(old, new)
...

Serial passing of cache line between cores

Atomic

Core A  Core B  Core C  Core D  Core E  Core F

Directory

F
E
D
C
B
A
Serialization in Hardware Queueing

Serial passing of cache line between cores
Serialization in Hardware Queueing

- Compute new
  - ...
  - Load old
  - ... CAS(old, new)
  - ...

Serial passing of cache line between cores

- Atomic

- Directory
Adding node to shared stack

```c
void push () {
    Node *new_top = malloc();
    while(true) {
        old_top = stack;
        new_top->next = old_top;
        if(CAS(&stack, old_top, new_top))
            return;
    }
}
```

new_top generated early on, independent of old_top

Core A
- Compute new
- Ld old
- CAS

Core B
- Compute new
- Ld old
- Stall

Core C
- Compute new
- Ld old
- Stall

Time

Contribution I: Eager Forwarding - Insight
Adding node to shared stack

```c
void push () {
    Node *new_top = malloc();
    while(true) {
        old_top = stack;
        new_top->next = old_top;
        if(CAS(&stack, old_top, new_top))
            return;
    }
    return;
}
```

Atomic

new_top generated early on, independent of old_top

---

**CASPAR – Eager Forwarding**

Out-of-window Speculative Execution after register check-pointing
CASPAR – Eager Forwarding

Atomic

Compute new
... Load old
... CAS(\text{old, new})
...

Compute new
... Load old
... CAS(\text{old, new})
...

Directory

Core A
Core B
Core C
Core D
Core E
Core F

ld old_A
ld old_B
ld old_C
ld old_D
ld old_E
ld old_F

F
E
D
C
B
A
CASPAR – Eager Forwarding

Atomic

Compute new
... 
Load old
... 
CAS(\text{old}, \text{new})
...

Core A  Core B  Core C  Core D  Core E  Core F

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| F | E | D | C | B | A

Line

Directory
CASPAR – Eager Forwarding

Parallel passing of speculative values between cores

Atomic
- Compute new
  ... Load old
  ... CAS(\text{old, new})
  ...

Directory

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>newA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>newB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>newC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>newD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>newE</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>newF</td>
</tr>
</tbody>
</table>

A \rightarrow \text{Directory} \\
B \rightarrow \text{Directory} \\
C \rightarrow \text{Directory} \\
D \rightarrow \text{Directory} \\
E \rightarrow \text{Directory} \\
F \rightarrow \text{Directory}
CASPAR – Eager Forwarding

Compute new
... Load old
... CAS(old, new)...

Atomic

Serial Validation

Compute new
... Load old
... CAS(old, new)...

Direction

Line

Core A → Core B → Core C → Core D → Core E → Core F

Spec.

F

new F

Directory
Limitations of Eager Forwarding
Contribution II: Parallel Validation

Time

Core A
- Compute new
  - Ld old
  - CAS

Core B
- Compute new
  - Ld old
  - CAS

Core C
- Compute new
  - Ld old
  - CAS

Directory

Validation

TWO-PHASE COMMIT (2PC) PROTOCOL TO PERFORM PARALLEL VALIDATION

Idea: Validate in the directory without ever sending the full line to the core
CASPAR – Parallel Validation

Atomic

Compute new
... Load old
... CAS(old, new)
...

Parallel passing of speculative values between cores
CASPAR – Parallel Validation

Compute new
... Load old
... CAS(old, new)
...

Parallel passing of speculative values between cores

Directory

<table>
<thead>
<tr>
<th>Core A</th>
<th>Core B</th>
<th>Core C</th>
<th>Core D</th>
<th>Core E</th>
<th>Core F</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_A</td>
<td>new_B</td>
<td>new_C</td>
<td>new_D</td>
<td>new_E</td>
<td>new_F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F</th>
<th>new_F</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>new_E</td>
</tr>
<tr>
<td>D</td>
<td>new_D</td>
</tr>
<tr>
<td>C</td>
<td>new_C</td>
</tr>
<tr>
<td>B</td>
<td>new_B</td>
</tr>
<tr>
<td>A</td>
<td>new_A</td>
</tr>
</tbody>
</table>
CASPAR – Parallel Validation

Atomic

- Compute new
- Load old
- CAS(old, new)

Parallel passing of speculative values between cores

Directory

Core A
Core B
Core C
Core D
Core E
Core F

Line

Line == new_A ??

F  new_F
E  new_E
D  new_D
C  new_C
B  new_B
A  new_A
CASPAR – Parallel Validation

Parallel passing of speculative values between cores

Atomic
Compute new
... Load old
... CAS(old, new)
...

Compute new
... Load old
... CAS(old, new)
...

Parallel validation

Directory

Core A
Core B
Core C
Core D
Core E
Core F

F  new_F
E  new_E
D  new_D
C  new_C
B  new_B
CASPAR – Parallel Validation

Parallel passing of speculative values between cores

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>new&lt;sub&gt;F&lt;/sub&gt;</td>
<td>E</td>
<td>new&lt;sub&gt;E&lt;/sub&gt;</td>
<td>D</td>
<td>new&lt;sub&gt;D&lt;/sub&gt;</td>
</tr>
<tr>
<td>C</td>
<td>new&lt;sub&gt;C&lt;/sub&gt;</td>
<td>B</td>
<td>new&lt;sub&gt;B&lt;/sub&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CASPAR – Parallel Validation

Parallel passing of speculative values between cores

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>new_F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>new_E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>new_D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>new_C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>new_B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Directory

Ack

Nack

Compute new
Load old
CAS(old, new)
CASPAR – Parallel Validation

- Atomic
  - Compute \textit{new}
  - Load \textit{old}
  - CAS(\textit{old, new})

- Parallel passing of speculative values between cores

- Parallel Validation

Diagram:
- Core A
- Core B
- Core C
- Core D
- Core E
- Core F
- Commit
- Commit
- Commit
- Spec.
- Spec.
- Resume
- Ack’ing cores halt. ‘Resume’ re-starts execution
- Directory
- F \textit{new}_F
- E \textit{new}_E
Outcome

- Fully Parallel Data Transfer
- Fully Parallel De-queue
Evaluation

- **Sniper simulator**, 64 cores, OOO
- **Kernels**: FIFO, LIFO, LIFO-push-only, Mandelbrot, Larson
- **Applications**: Galois graph analytics suite (3x2), Barcelona OpenMP Tasks Suite (1x2)
Evaluation
Evaluation
Evaluation
Independent Set (IS) gains from Eager Forwarding since early data transfer exposes work that could be done speculatively and in parallel.
Evaluation

Connected Components (CC) and Delaunay Triangulation (DT) gain from Parallel Validation as it removes stalls in EF using group commit.

Normalized Time

- FFT-FIFO
- FFT-LIFO
- CC-FIFO
- CC-LIFO
- IS-FIFO
- IS-LIFO
- DT-FIFO
- DT-LIFO
- Avg.

2.5x
Also in the paper

✓ Information on proposed hardware changes
✓ Handling memory consistency issues
✓ Detailed working and correctness of Eager Forwarding and 2PC protocols
✓ Scalability plots, cycle-level breakdown for applications
✓ Supporting other primitives (e.g. LL/SC)
Conclusions

• CASPAR enables **true parallelism** in codes using lock-free synchronization
  - Eager Forwarding: **Parallel transfer of data**
  - Parallel Validation: **Parallel de-queue**

• No source code modifications

• **Future Work**
  - Can be used to break conflict serialization in TM designs
CASPAR: BREAKING SERIALIZATION IN LOCK-FREE MULTICORE SYNCHRONIZATION

Tanmay Gangwani ‡, Adam Morrison*, Josep Torrellas †

University of Illinois, Urbana-Champaign ‡
Technion – Israel Institute of Technology*
CASPAR BACKUP
Maintaining Consistency

Core A

ld old
{new value forwarded to core B}
Z = 1
CAS(old, new)

Core B

ld old
.. = Z (read 0)

Validation

Time
Maintaining Consistency

Maintaining Consistency

Forwarding causes no consistency violations
Evaluation (kernels)

CAS throughput improvements
- EF over Base: 53%
- C over Base: 83%
- EF over Queue: 10%
- C over Queue: 32%
Evaluation (scalability)

CASPAR greatly improves scalability with number of cores
### CASPAR in Entirety

#### Effective Serialization

<table>
<thead>
<tr>
<th>Module I : Hardware Queueing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory chains cores in hardware</td>
</tr>
</tbody>
</table>

#### Breaking Serialization

<table>
<thead>
<tr>
<th>Module II : Eager Forwarding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory facilitates passing of speculative values between cores</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module III : Two-Phase Group Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory facilitates collective-validation of multiple cores</td>
</tr>
</tbody>
</table>

---

**Diagram:**

- **Core A**
- **Core B**
- **Core C**

**Effective Serialization Diagram:**

- Core A ➔ Core B ➔ Core C

**Breaking Serialization Diagram:**

- Core A ➔ Core B ➔ Core C
- Passing Speculative Value

**Group Validation Diagram:**

- Core A ➔ Core B ➔ Core C
- Group Validation