SyncProf: Detecting, Localizing, and Optimizing Synchronization Bottlenecks

Tingting Yu
University of Kentucky

Michael Pradel
TU Darmstadt
Motivation

Challenge: Synchronization bottlenecks

Photo: Jürgen Schoner
Motivation

Challenge:
Synchronization bottlenecks

Profiling tools:
Very limited
Motivation

Challenge: Synchronization bottlenecks

Profiling tools: Very limited

Finding, understanding, and fixing synchronization bottlenecks: Mostly manual
Example

Synchronization bottleneck in KVM/QEMU driver:

Thread 1

Thread 2

Thread 3

Thread 4

Thread 5

.. critical section with time to obtain lock, colors = locks

CS1

CS3
Example

Synchronization bottleneck in KVM/QEMU driver:

- Thread 1
  - Critical Section (CS1)

- Thread 2
  - Critical Section (CS2)
  - Critical Section (CS3)

- Thread 3
  - Critical Section (CS4)

- Thread 4
  - Critical Section (CS5)

- Thread 5
  - Critical Section (CS6)

.. critical section with time to obtain lock, colors = locks
Example

Synchronization bottleneck in KVM/QEMU driver:

Thread 1

Thread 2

Thread 3

Thread 4

Thread 5

Longest wait time

.. critical section with time to obtain lock, colors = locks
Example

Synchronization bottleneck in KVM/QEMU driver:

Thread 1: CS1
Thread 2: CS2, CS3
Thread 3: CS4
Thread 4: CS5
Thread 5: CS6

Longest critical section

.. critical section with time to obtain lock, colors = locks
Example

Synchronization bottleneck in KVM/QEMU driver:

Thread 1: CS1
Thread 2: CS2 → CS3
Thread 3: CS4
Thread 4: CS5
Thread 5: CS6

---

.. critical section with time to obtain lock, colors = locks
Goals & Challenges

Find synchronization bottlenecks

Locate the root cause of a bottleneck

Help optimize the bottleneck
Goals & Challenges

Find synchronization bottlenecks

Locate the root cause of a bottleneck

Help optimize the bottleneck

This talk: SyncProf
Actionable performance profiling for concurrent programs
Overview of SyncProf

Program + Inputs

Bottleneck detection

Root cause analysis

Find optimization strategies

Synchronization bottlenecks and suggestions for optimizations
Overview of SyncProf

Program + Inputs

Bottleneck detection

Root cause analysis

Find optimization strategies

Synchronization bottlenecks and suggestions for optimizations

Complexity & overhead

Considered program parts
Overview of SyncProf

Program + Inputs

Bottleneck detection

Root cause analysis

Find optimization strategies

Synchronization bottlenecks and suggestions for optimizations

Complexity & overhead

Considered program parts
Bottleneck Detection

Find **inputs** that **trigger** synchronization bottlenecks

Configurable workload size $s$  

\[
\begin{align*}
\text{Test 1} \\
\text{Test 2} \\
\vdots
\end{align*}
\]

Program
Bottleneck Detection

Find **inputs** that **trigger** synchronization **bottlenecks**

Configurable workload size $s$

\[
\begin{aligned}
\text{Test 1} \\
\text{Test 2} \\
\vdots
\end{aligned}
\]

Program

For each test $t$:

- Execute $t$ with increasing $s$
- If increase of $s$ implies increase of execution time and CPU usage $\lessgtr$ threshold: Keep $t$ and $s$
Overview of SyncProf

Program + Inputs

Bottleneck detection

Root cause analysis

Find optimization strategies

Synchronization bottlenecks and suggestions for optimizations

Complexity & overhead

Considered program parts
Graph-based Root Cause Analysis

1) Summarize execution into graph
2) Analyze graph to find root cause

Synchronization dependence graph
- Nodes: Dynamic instances of critical sections
- Edges: Waits-for relations
Example

Critical sections with time to obtain lock, colors = locks
Example

Direct waits-for relations

- Thread 1: CS1, CS2
- Thread 2: CS3, CS4
- Thread 3: CS5, CS6
- Thread 4: CS7

.. critical section with time to obtain lock, colors = locks
Example

Indirect waits-for relations

- Thread 1
  - CS1
  - CS2

- Thread 2
  - CS3
  - CS4

- Thread 3
  - CS5
  - CS6

- Thread 4
  - CS7

.. critical section with time to obtain lock, colors = locks
Example

Associate cost to each edge

Thread 1

Thread 2

Thread 3

Thread 4

Time

CS1

CS3

CS5

CS2

CS4

CS6

CS7

.. critical section with time to obtain lock, colors = locks
Example

Associate cost to each edge

Thread 1

Thread 2

Thread 3

Thread 4

Time

.. critical section with time to obtain lock, colors = locks
Example

Associate cost to each edge

- Thread 1
  - CS1
  - CS2
  - CS3
  - CS4
  - CS5
  - CS6
  - CS7

- Thread 2
- Thread 3
- Thread 4

```
.. critical section with time to obtain lock, colors = locks
```
Example

Graph with cost-labeled edges
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause

Metric 1: All-path wait time
- How long did other critical sections wait for a particular critical section?
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause

Metric 1: All-path wait time

- How long did other critical sections wait for a particular critical section?

![Diagram of critical sections with wait times indicated.]
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause

Metric 1: All-path wait time
- How long did other critical sections wait for a particular critical section?

- $4+4=8$
- Highest rank
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause

Metric 2: Critical path wait time
- Consider only critical path through synchronization dependence graph
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause

Metric 2: Critical path wait time
- Consider only critical path through synchronization dependence graph
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause

Metric 2: Critical path wait time
- Consider only critical path through synchronization dependence graph
Measuring Performance Impact

Rank critical sections based on their likelihood to be the **root cause**

**Metric 2: Critical path wait time**

- Consider only critical path through synchronization dependence graph

  - $2+2+2=6$
  - Highest rank
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause

Metric 3: All-path lock time
- How long did critical sections wait for a particular lock?
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause

Metric 3: All-path lock time
- How long did critical sections wait for a particular lock?

- 12 vs. 12
- Same rank
Measuring Performance Impact

Rank critical sections based on their likelihood to be the root cause

- One graph, several metrics
- Rank critical sections by one or more metrics
Overview of SyncProf

Program + Inputs

Bottleneck detection

Root cause analysis

Find optimization strategies

Synchronization bottlenecks and suggestions for optimizations
How to Optimize the Bottlenecks?

Challenge: Bottleneck ⇨ Optimizable

Dynamic analysis of likely root causes:

- Track reads and writes of critical sections
- Merge information across executions
- Suggest common optimization patterns
Pattern-based Suggestions

Suggest to ..

- eliminate synchronization
- split lock
- use read-writer lock

When ..

- no shared memory access
- critical sections access disjoint memory
- mostly read-only critical sections
Evaluation: Setup

Questions

- Effectiveness
- Efficiency
- Comparison with Valgrind’s lock contention profiler

Setup

- Firefox, MySQL, 6 benchmarks
- 15 known bottlenecks
Detected Bottlenecks

18 bottlenecks (15 known + 3 new)

Rank root cause by critical section

- 8 of 18 ranked first
- All in top 5% (of 27–119 critical sections)

Rank root cause by lock

- 15 of 18 ranked first
Optimizations

Out of 18 bottlenecks:

- 9 optimizations suggested
  - 7 match fix by developers
  - 2 false suggestions

- 5 reported as low-degree conflicts
  - Application-specific optimizations needed

- 4 without any match
Optimizations

Out of 18 bottlenecks:

- 9 optimizations suggested
  - 7 match fix by developers
  - 2 false suggestions
- 5 reported as low-degree conflicts
  - Application-specific optimizations needed
- 4 without any match

Example: MySQL

- Remove unnecessary lock for read-read accesses
Optimizations

Out of 18 bottlenecks:

- 9 optimizations suggested
  - 7 match fix by developers
  - 2 false suggestions

- 5 reported as low-degree conflicts
  - Application-specific optimizations needed

- 4 without any match

Example: Splash-2 Radiosity

- Turn shared queue into non-blocking queue
Optimizations

Out of 18 bottlenecks:

- 9 optimizations suggested
  - 7 match fix by developers
  - 2 false suggestions

- 5 reported as low-degree conflicts
  - Application-specific optimizations needed

- 4 without any match

Example: MySQL

- Instead of shared output buffer, use two buffers
## Comparison with Valgrind

<table>
<thead>
<tr>
<th></th>
<th>Valgrind</th>
<th>SyncProf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs &amp; executions</td>
<td>Developer must choose</td>
<td>Automatically selected and summarized</td>
</tr>
<tr>
<td>Critical sections to</td>
<td>Rank 1 to 14</td>
<td>Rank 1 to 5</td>
</tr>
<tr>
<td>inspect</td>
<td><em>Reduced by 55% (avg.)</em></td>
<td></td>
</tr>
<tr>
<td>Optimizations</td>
<td>No support</td>
<td>Common patterns</td>
</tr>
</tbody>
</table>
Efficiency

Runtime overhead

- Root cause analysis: 4x–10x
- Optimization suggestion: 60x–100x

Total time: 13–340 minutes per program
Efficiency

Runtime overhead

■ Root cause analysis: 4x–10x
■ Optimization suggestion: 60x–100x

Total time: 13–340 minutes per program

Acceptable for in-house profiling
Conclusion

SyncProf: Actionable performance profiling for concurrent programs

- Detect bottlenecks
- Identify root causes
- Suggest optimizations

Take-aways for analysis writers

- Multi-stage analysis with increasing complexity
- Generic graph as basis for multiple analyses
Conclusion

SyncProf: Actionable performance profiling for concurrent programs

- Detect bottlenecks
- Identify root causes
- Suggest optimizations

Take-aways for analysis writers

- Multi-stage analysis with increasing complexity
- Generic graph as basis for multiple analyses

Thanks!