Response-Time Analysis for Non-Preemptive Global Scheduling with Spin Locks

Suhail Nogd

Supervised by: Dr. ir. Geoffrey Nelissen

Co-supervised by: Dr. Mitra Nasri

August 2020
Real-time systems

A system’s correctness depends on

Logical correctness of results

AND

Timing correctness of results
Real-time systems

A system’s correctness depends on

Logical correctness of results

AND

Timing correctness of results

Automotive

Medical

Aircraft
Future real-time systems require high computing power.
Future real-time systems require high computing power

Multiprocessor Platforms

- Increased Computing Power
- Ability to exploit parallelism
Future real-time systems require high computing power

Multiprocessor Platforms

- Increased Computing Power
- Ability to exploit parallelism

- Increased Complexity (Scheduling & Analysis)
Multiprocessor scheduling

Partitioned
Tasks do not migrate

Migration
Multiprocessor scheduling

**Task** – A specific functionality of a system

**Job** – An instance of a task

**Partitioned**
Tasks do not migrate

**Migration**
Multiprocessor scheduling

**Task** – A specific functionality of a system

**Job** – An instance of a task

- **Partitioned**
  - Tasks do not migrate

- **Semi-Partitioned**
  - Some tasks can migrate
Multiprocessor scheduling

**Task** – A specific functionality of a system

**Job** – An instance of a task

**Partitioned**
Tasks do not migrate

**Semi-Partitioned**
Some tasks can migrate

**Global**
All tasks can migrate

Migration
Multiprocessor scheduling

**Task** – A specific functionality of a system

**Job** – An instance of a task

<table>
<thead>
<tr>
<th>Partitioned</th>
<th>Semi-Partitioned</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks do not migrate</td>
<td>Some tasks can migrate</td>
<td>All tasks can migrate</td>
</tr>
</tbody>
</table>

More efficient utilization of platform resources!
Global job-level fixed-priority (JLFP)

What is it?
A range of policies that assign jobs to processor(s) at specific time intervals to execute their workload
Global job-level fixed-priority (JLFP)

What is it?
A range of policies that assign jobs to processor(s) at specific time intervals to execute their workload
- *Earliest Deadline First (EDF)*
Global job-level fixed-priority (JLFP)

What is it?
A range of policies that assign jobs to processor(s) at specific time intervals to execute their workload

- *Earliest Deadline First (EDF)*
- *Fixed-Priority (FP)*
Shared resource

**What is it?**
- A resource that is requested by two or more tasks
Shared resource

What is it?
- A resource that is requested by two or more tasks

```c
// global variable
int global = 5;
```
Suspension-based locks vs. spin-based locks

**Suspension-based locks**
(i.e., yield core when blocked)

**Spin-based locks**
(i.e., busy-wait on core when blocked)
Suspension-based locks vs. spin-based locks

**Suspension-based locks**
(i.e., yield core when blocked)
- Blocked core used for useful executions
- Increased overhead
- Increased analysis complexity
- Increased OS support

**Spin-based locks**
(i.e., busy-wait on core when blocked)
## Suspension-based locks vs. spin-based locks

### Suspension-based locks (i.e., yield core when blocked)
- Blocked core used for useful executions
- Increased overhead
- Increased analysis complexity
- Increased OS support

### Spin-based locks (i.e., busy-wait on core when blocked)
- Reduced analysis complexity
- Reduced OS support
- Reduced overhead
- Busy-waiting (core not used for useful computation)
Suspension-based locks vs. spin-based locks

**Suspension-based locks**
(i.e., yield core when blocked)
- Blocked core used for useful executions
- Increased overhead
- Increased analysis complexity
- Increased OS support

**Spin-based locks**
(i.e., busy-wait on core when blocked)
- Reduced analysis complexity
- Reduced OS support
- Reduced overhead
- Busy-waiting (core not used for useful computation)

**Critical Section** – Execution section that requires a shared resource
Suspension-based locks vs. spin-based locks

**Suspension-based locks** (i.e., yield core when blocked)

- Blocked core used for useful executions
- More efficient for long critical sections
- Increased overhead
- Increased analysis complexity
- Increased OS support

**Spin-based locks** (i.e., busy-wait on core when blocked)

- Reduced analysis complexity
- Reduced OS support
- Reduced overhead
- More efficient for short critical sections
- Busy-waiting (core not used for useful computation)

Critical Section – Execution section that requires a shared resource
Suspension-based locks vs. spin-based locks

Suspension-based locks (i.e., yield core when blocked)
- Blocked core used for useful executions
- More efficient for long critical sections
- Increased overhead
- Increased analysis complexity
- Increased OS support

Spin-based locks (i.e., busy-wait on core when blocked)
- Reduced analysis complexity
- Reduced OS support
- Reduced overhead
- More efficient for short critical sections
- Busy-waiting (core not used for useful computation)

Critical Section – Execution section that requires a shared resource
## Suspension-based locks vs. spin-based locks

<table>
<thead>
<tr>
<th>Suspension-based locks (i.e., yield core when blocked)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Blocked core used for useful executions</td>
</tr>
<tr>
<td>• More efficient for long critical sections</td>
</tr>
<tr>
<td>• Increased overhead</td>
</tr>
<tr>
<td>• Increased analysis complexity</td>
</tr>
<tr>
<td>• Increased OS support</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Spin-based locks (i.e., busy-wait on core when blocked)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Reduced analysis complexity</td>
</tr>
<tr>
<td>• Reduced OS support</td>
</tr>
<tr>
<td>• Reduced overhead</td>
</tr>
<tr>
<td>• More efficient for short critical sections</td>
</tr>
<tr>
<td>• Busy-waiting (core not used for useful computation)</td>
</tr>
</tbody>
</table>

### Critical Section – Execution section that requires a shared resource

- **FIFO-ordered spin locks**
- **priority-ordered spin locks**
Spin locks

An example schedule of 3 jobs & 1 shared resource:

Legend:
- Normal computation
- Shared resource access

Core 1:

0  2  5  7

Core 2:

Time

Core 3:

Time

Releases:

0  2  3

Next to schedule jobs: (J2)

Priorities:

J1

J2

J3

(J3)

(J2)

(J3)

J2

J3

Low

High

TU Delft

Response-Time Analysis for Non-Preemptive Global Scheduling with Spin Locks
Spin locks

An example schedule of 3 jobs & 1 shared resource:

Legend:
- Normal computation
- Shared resource access

Core 1:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>7</th>
</tr>
</thead>
</table>

Core 2:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

Core 3:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

Releases:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

Next to schedule jobs:

<table>
<thead>
<tr>
<th>Time</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
</table>

Priorities:

- \( J_1 \) Low
- \( J_2 \) High
- \( J_3 \)
An example schedule of 3 jobs & 1 shared resource:

Core 1:

Time

0 2 5 7

J

Releases:

0 2 3

Legend:

- Normal computation
- Shared resource access

Core 2:

Time

Core 3:

Time

Next to schedule jobs: (J2)

Priorities:

J

J

J

High

Low
An example schedule of 3 jobs & 1 shared resource:

Core 1:

<table>
<thead>
<tr>
<th>Time</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Core 2:

<table>
<thead>
<tr>
<th>Time</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 3:

<table>
<thead>
<tr>
<th>Time</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Releases:

<table>
<thead>
<tr>
<th>Time</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>J1</td>
</tr>
<tr>
<td>2</td>
<td>J2</td>
</tr>
<tr>
<td>3</td>
<td>J3</td>
</tr>
</tbody>
</table>

Next to schedule jobs: 
- (J2) 5
- (J3) 2

Priorities:
- J1 Low
- J2 High
- J3 High

Legend:
- Normal computation
- Shared resource access
An example schedule of 3 jobs & 1 shared resource:

Core 1:

<table>
<thead>
<tr>
<th>Time</th>
<th>Core 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Time

Core 2:

Time

Core 3:

Time

Releases:

<table>
<thead>
<tr>
<th>Time</th>
<th>Releases</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Next to schedule jobs: (J2)  

Priorities:

J1 Low

J2 High

J3

Legend:

Normal computation

Shared resource access

Spin locks
Fifo-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

Legend:
- Normal computation
- Shared resource access

Core 1:
- Job 1 (J1)
  - Time: 0-5
- Job 2 (J2)
  - Time: 2-7
- Job 3 (J3)
  - Time: 3-8

Core 2:
- Job 1 (J1)
  - Time: 0-5
- Job 2 (J2)
  - Time: 2-7

Core 3:
- Job 2 (J2)
  - Time: 5-7
- Job 3 (J3)
  - Time: 7-8

Releases:
- Job 1 (J1)
  - Time: 0, 5
- Job 2 (J2)
  - Time: 2, 7
- Job 3 (J3)
  - Time: 3, 8

Next to schedule jobs: (J2) 5 (J3) 2

Priorities:
- J1 Low
- J2 High
- J3 High
Fifo-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

Legend:
- Normal computation
- Shared resource access

Core 1:
- Job $J_1$ with busy waiting
- Time: 0-2, 5-7

Core 2:
- Job $J_2$
- Time: 2-3, 5-7

Core 3:
- Job $J_3$
- Time: 3-4, 7-8

Releases:
- $J_1$: 0, 2, 5
- $J_2$: 2, 5
- $J_3$: 3, 7

Next to schedule jobs: (J2)

Priorities:
- $J_1$: Low
- $J_2$: High
- $J_3$: Low

An example schedule of 3 jobs & 1 shared resource:
Fifo-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

Legend:
- Normal computation
- Shared resource access

Next to schedule jobs:  (J2)
Priorities:  J2  High
            J1  Low

Core 1:
J1

Core 2:
J2

Core 3:
J3

Releases:
J1  J2  J3

0  2  3  5  7  9

Time
Fifo-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

Legend:
- Normal computation
- Shared resource access

Core 1:

<table>
<thead>
<tr>
<th>Time</th>
<th>Job</th>
<th>Job</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

Core 2:

<table>
<thead>
<tr>
<th>Time</th>
<th>Job</th>
<th>Job</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

Core 3:

<table>
<thead>
<tr>
<th>Time</th>
<th>Job</th>
<th>Job</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

Releases:

<table>
<thead>
<tr>
<th>Time</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Next to schedule jobs: (J2)

Priorities:

J1 Low
J2 High
J3
Fifo-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

Legend:
- Normal computation
- Shared resource access

Core 1:
- Jobs: $J_1$, $J_2$, $J_3$
- Releases: 0, 2, 3
- Time: 0, 2, 3, 5, 7, 9

Core 2:
- Jobs: $J_1$, $J_2$, $J_3$
- Releases: 0, 2, 3
- Time: 2, 3, 5, 7, 9

Core 3:
- Jobs: $J_1$, $J_2$, $J_3$
- Releases: 0, 2, 3
- Time: 3, 4, 7, 8

Next to schedule jobs: $(J_2)$

Priorities:
- $J_1$: Low
- $J_2$: High
- $J_3$: High

Response-Time Analysis for Non-Preemptive Global Scheduling with Spin Locks
Priority-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

Legend:
- Normal computation
- Shared resource access

Core 1:
- Time: 0 2 5 7
- Jobs:
  - $J_1$ (0-7)

Core 2:
- Time:
- Jobs:
  - $J_2$ (0-2)
  - $J_3$ (3-5)

Core 3:
- Time:
- Jobs:
  - $J_2$ (0-5)
  - $J_3$ (2-7)

Releases:
- Time: 0 2 3
- Releases:
  - $J_1$
  - $J_2$
  - $J_3$

Next to schedule jobs: ($J_2$)
  - $J_2$ (5)

Priorities:
- $J_1$: Low
- $J_3$: High

TU Delft Analysis for Non-Preemptive Global Scheduling with Spin Locks
Priority-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

Legend:
- Normal computation
- Shared resource access

Core 1:
- J₁
- Time: 0, 2, 5, 7

Core 2:
- J₂
- Time: 2, 3, 6, 8, 10

Core 3:
- J₃
- Time: 3, 4, 5, 6

Releases:
- J₁, J₂, J₃
- Time: 0, 2, 3

Next to schedule jobs: (J₂)
- Time: 5

Priorities:
- J₁ (Low)
- J₂, J₃ (High)
Priority-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

Core 1:

Core 2:

Core 3:

Releases:

Next to schedule jobs:  (J2)  (J3)

Priorities:

Legend:
- Normal computation
- Shared resource access

J1

J2

J3
Priority-ordered spin locks

An example schedule of 3 jobs & 1 shared resource:

Core 1:

0 2 5 7
J \_1

Core 2:

2 3 6 8 10
J \_2

Core 3:

3 4 5
J \_3

Releases:

0 2 3

Next to schedule jobs:  
(J\_2)

Priorities:

J \_1 Low

J \_2 High

J \_3
Non-preemptive scheduling
Non-preemptive scheduling

No added context switch cost

Non-preemptive

Preemptive

Cost of saving and restoring context

WCET
Non-preemptive scheduling

- No added context switch cost
- Less cache-related preemption delays

No added context switch cost

Cost of saving and restoring context

Fewer cache evictions
Non-preemptive scheduling

- No added context switch cost
- Less cache-related preemption delays
- Improved timing predictability
- More accurate WCET estimation

WCET

Cost of saving and restoring context

Non-preemptive

Preemptive
Response-time analysis

What is it?
- Analysis that checks whether a given workload can be processed within the specified timing requirements
Response-time analysis

What is it?
- Analysis that checks whether a given workload can be processed within the specified timing requirements

Why is it important?
- Helps predict best- and worst-case timing behaviour of a system and allows to validate temporal requirements
Response-time analysis

**What is it?**
- Analysis that checks whether a given workload can be processed *within the specified timing constraints*

**Why is it important?**
- Helps **predict** best- and worst-case timing behaviour of a system and allows to **validate** temporal requirements

---

**Diagram:**
- Collision
- Too early
- Good
- Too late
- **Response Time**
- **Airbag fully open**
- **Time**
What analyses currently exist?
Blocking-aware schedulability analyses for global scheduling

Most interesting work
Blocking-aware schedulability analyses for global scheduling

Most interesting work
- Holistic blocking spin-based analysis FMLP short
- Holistic blocking suspension-based analysis OMLP

Blocking-aware schedulability analyses for global scheduling

Most interesting work
- Holistic blocking spin-based analysis FMLP short
- Holistic blocking suspension-based analysis OMLP
- Linear-programming suspension-based analysis FMLP long
- Linear-programming suspension-based analysis PIP

Blocking-aware schedulability analyses for global scheduling

Most interesting work
- Holistic blocking spin-based analysis FMLP short
- Holistic blocking suspension-based analysis OMLP
- Linear-programming suspension-based analysis FMLP long
- Linear-programming suspension-based analysis PIP

Blocking-aware schedulability analyses for global scheduling

Most interesting work
- Holistic blocking spin-based analysis FMLP short
- Holistic blocking suspension-based analysis OMLP
- Linear-programming suspension-based analysis FMLP long
- Linear-programming suspension-based analysis PIP

Analysis for non-preemptive execution and spin locks?

Blocking-aware schedulability analyses for global scheduling

Most interesting work

- Holistic blocking spin-based analysis FMLP short
- Holistic blocking suspension-based analysis OMLP
- Linear-programming suspension-based analysis FMLP long
- Linear-programming suspension-based analysis PIP

Analysis for non-preemptive execution and spin locks?

Currently no such analysis exists!

The problem in a nutshell
The problem in a nutshell

We obtain the **best-case** and **worst-case response time**
The problem in a nutshell

We obtain the **best-case** and **worst-case** response time

**Workload model**

*Tasks/Jobs that share resources (Spin locks)*
The problem in a nutshell

We obtain the **best-case** and **worst-case response time**

<table>
<thead>
<tr>
<th>Workload model</th>
<th>Execution model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks/Jobs that share resources <em>(Spin locks)</em></td>
<td>Non-preemptive</td>
</tr>
</tbody>
</table>
The problem in a nutshell

We obtain the **best-case** and **worst-case response time**

<table>
<thead>
<tr>
<th>Workload model</th>
<th>Execution model</th>
<th>Platform model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tasks/Jobs that share resources</strong> <em>(Spin locks)</em></td>
<td><strong>Non-preemptive</strong></td>
<td><strong>Multicore</strong> <em>(identical cores)</em></td>
</tr>
</tbody>
</table>
The problem in a nutshell

We obtain the **best-case and worst-case response time**

<table>
<thead>
<tr>
<th>Workload model</th>
<th>Execution model</th>
<th>Platform model</th>
<th>Scheduler model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks/Jobs that share resources <em>(Spin locks)</em></td>
<td>Non-preemptive</td>
<td>Multicore <em>(identical cores)</em></td>
<td>Global job-level fixed priority <em>(JLFP)</em></td>
</tr>
</tbody>
</table>
The problem in a nutshell

We obtain the best-case and worst-case response time

Workload model
Tasks/Jobs that share resources (Spin locks)

Execution model
Non-preemptive

Platform model
Multicore (identical cores)

Scheduler model
Global job-level fixed priority (JLFP)

Task model

Arrival model
Bounded uncertainty

Execution model
Resource independent
Resource dependent
Bounded uncertainty

Best case (BCET)
Worst case (WCET)

Deadline (hard or soft)
The problem in a nutshell

We obtain the **best-case** and **worst-case** response time

**Workload model**
- Tasks/Jobs that share resources (Spin locks)

**Execution model**
- Non-preemptive

**Platform model**
- Multicore (identical cores)

**Scheduler model**
- Global job-level fixed priority (JLFP)

**Task model**

**Arrival model**
- Time

**Execution model**
- Resource independent
- Resource dependent
- Bounded uncertainty

**Bounded uncertainty**
- Best case (BCET)
- Worst case (WCET)

**Deadline** (hard or soft)
The problem in a nutshell

We obtain the **best-case and worst-case response time**

<table>
<thead>
<tr>
<th>Workload model</th>
<th>Execution model</th>
<th>Platform model</th>
<th>Scheduler model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks/Jobs that share resources (Spin locks)</td>
<td>Non-preemptive</td>
<td>Multicore (identical cores)</td>
<td>Global job-level fixed priority (JLFP)</td>
</tr>
</tbody>
</table>

**Task model**

**Arrival model**

**Execution model**

- Bounded uncertainty
- Resource independent
- Resource dependent
- Bounded uncertainty

**Best case (BCET)**

**Worst case (WCET)**

**Deadline** (hard or soft)

Time

**Workload model**

- Tasks/Jobs that share resources (Spin locks)

**Execution model**

- Non-preemptive

**Platform model**

- Multicore (identical cores)

**Scheduler model**

- Global job-level fixed priority (JLFP)
The problem in a nutshell

We obtain the best-case and worst-case response time

Workload model
- Tasks/Jobs that share resources (Spin locks)

Execution model
- Non-preemptive

Platform model
- Multicore (identical cores)

Scheduler model
- Global job-level fixed priority (JLFP)

Task model
- Arrival model
  - Bounded uncertainty

Execution model
- Resource independent
  - Best case (BCET)
- Resource dependent
  - Worst case (WCET)

Deadline (hard or soft)

Time

Bounded uncertainty
The problem in a nutshell

We obtain the **best-case** and **worst-case** response time

<table>
<thead>
<tr>
<th>Workload model</th>
<th>Execution model</th>
<th>Platform model</th>
<th>Scheduler model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks/Jobs that share resources (Spin locks)</td>
<td>Non-preemptive</td>
<td>Multicore (identical cores)</td>
<td>Global job-level fixed priority (JLFP)</td>
</tr>
</tbody>
</table>

**Task model**

- **Arrival model**
- **Execution model**
  - Resource independent
  - Resource dependent

**Execution model**

- **Deadline** (hard or soft)

**Time**

- **Best case** (BCET)
- **Worst case** (WCET)

**Bounded uncertainty**

Response-Time Analysis for Non-Preemptive Global Scheduling with Spin Locks
Contributions & achievements

Response-time analysis considering **FIFO-ordered spin locks**
Contributions & achievements

Response-time analysis considering **FIFO-ordered spin locks**

Response-time analysis considering **priority-ordered spin locks**
Contributions & achievements

Response-time analysis considering FIFO-ordered spin locks

Response-time analysis considering priority-ordered spin locks

➢ Support for multi-unit resources & k-exclusion protocols
➢ Additional techniques to improve scalability
Contributions & achievements

Response-time analysis considering **FIFO-ordered spin locks**

Response-time analysis considering **priority-ordered spin locks**

- Support for multi-unit resources & k-exclusion protocols
- Additional techniques to improve scalability

Accepted and presented at CAPITAL

Full paper submitted to RTSS
Contributions & achievements

Response-time analysis considering FIFO-ordered spin locks

Response-time analysis considering priority-ordered spin locks

➢ Support for multi-unit resources & k-exclusion protocols

➢ Additional techniques to improve scalability

Accepted and presented at CAPITAL

Full paper submitted to RTSS

Plan for future:
Schedulability analyses
Schedulability analyses

Closed-form analyses
(e.g., problem-window analysis)

Runtime  Accuracy
Schedulability analyses

- **Closed-form analyses** (e.g., problem-window analysis)
  
  - **Runtime**
  - **Accuracy**

- **Exact tests in generic formal verification tools** (e.g., UPPAAL)
  
  - **Runtime**
  - **Accuracy**
Schedulability analyses

Closed-form analyses
(e.g., problem-window analysis)

Exact tests in generic formal verification tools (e.g., UPPAAL)

Response-time analysis using schedule abstraction

State of the art: schedule-abstraction-based analyses


State of the art: schedule-abstraction-based analyses

[ECRTS’18]
- Multiprocessor
- Independent non-preemptive jobs/tasks
- Global work-conserving job-level fixed-priority scheduling (JLFP)

[ECRTS’19]
- Support for DAG tasks with precedence constraints

[This work]
- Support for shared resources with FIFO and priority-based spin locks


Schedule-abstraction analysis
Schedule-abstraction graph input and output

Schedule-abstraction analysis

Schedulable / Non-schedulable
Schedule-abstraction graph input and output

Arrival time intervals

Schedule-abstraction analysis

Scheduled / Non-schedulable
Schedule-abstraction graph input and output

Arrival time intervals → Schedule-abstraction analysis
Requested resources → Schedule-abstraction analysis

Schedule-abstraction analysis → Schedulable / Non-schedulable
Schedule-abstraction graph input and output

- Arrival time intervals
- Requested resources
- Best-/worst-case execution times

Schedule-abstraction analysis

Schedulable / Non-schedulable
Schedule-abstraction graph input and output

- Arrival time intervals
- Requested resources
- Best-/worst-case execution times
- Priorities

Schedule-abstraction analysis

Schedulable / Non-schedulable
Schedule-abstraction graph input and output

- Arrival time intervals
- Requested resources
- Best-/worst-case execution times
- Priorities
- Deadlines

Schedule-abstraction analysis

Schedulable / Non-schedulable
Schedule-abstraction graph input and output

- Arrival time intervals
- Requested resources
- Best-/worst-case execution times
- Priorities
- Deadlines
- Number of cores

Schedule-abstraction analysis

Schedulable / Non-schedulable
Schedule-abstraction graph input and output

- Arrival time intervals
- Requested resources
- Best-/worst-case execution times
- Priorities
- Deadlines
- Number of cores

**Schedule-abstraction analysis**

- Best-/worst-case response times
- Worst-case spinning times
- Schedulable / Non-schedulable
Schedule-abstraction graph

A path reflects a sequence of job-dispatch decisions

A path represents the order in which Jobs are dispatched on the Processors

Different paths have different job orders

Schedule-abstraction graph

A **path** reflects a sequence of job-dispatch decisions

A **vertex** abstracts a system state and an **edge** represents a dispatched job

A vertex abstracts a system state and an edge represents a dispatched job.

**Earliest and latest finish time of J**₁ **when it is dispatched after state v**

Start

\[ J₁: [2,6] \]

End


A path reflects a sequence of job-dispatch decisions. A vertex abstracts a system state and an edge represents a dispatched job. A state is labeled with the finish-time interval of any path reaching the state.

Schedule-abstraction graph

A system state

Start

End

Interpretation of an availability interval:

- Certainly not available
- Possibly available
- Certainly available
Building the schedule-abstraction graph

**Building the graph**
(a breadth-first method)

Repeat until every path includes all jobs
1. Find the shortest path
2. For each not-yet-dispatched job that can be dispatched after the path:
   2.1. **Expand** (add a new vertex)

Initial State
(System is idle)

Building the schedule-abstraction graph

Building the graph (a breadth-first method)

Repeat until every path includes all jobs
1. Find the shortest path
2. For each not-yet-dispatched job that can be dispatched after the path:
   2.1. Expand (add a new vertex)

Initial State
(System is idle)

J1

J2

J3

Building the schedule-abstraction graph

Initial State
(System is idle)

Repeat until every path includes all jobs
1. Find the shortest path
2. For each not-yet-dispatched job that can be dispatched after the path:
   2.1. Expand (add a new vertex)
   2.2. Merge (if possible, merge the new vertex with an existing vertex)

Designing a schedule-abstraction-based analysis

- State representation
- Expansion rules
- Merge rules
Agenda

To do’s:

- Design a system state representation
- Design expansion and merge rules
Schedule-abstraction graph considering shared resource accesses

Challenges
Handling shared resource

An example schedule of 2 jobs & 1 shared resource:

Core 1:

Core 2:

Releases:

Next to schedule Job (J₂):
Handling shared resource

An example schedule of 2 jobs & 1 shared resource:

Core 1:

Core 2:

Releases:

Spinning delay due to shared resource!
Handling shared resource

An example schedule of 2 jobs & 1 shared resource:

Core 1:

0 2 5 7

Core 2:

3 4 5 10

Start time of a Job does not necessarily dictate when it completes!
Handling shared resource

An example schedule of 2 jobs & 1 shared resource:

Core 1:

Start time of a Job does not necessarily dictate when it completes!

Core 2:

Start time of a Job does not necessarily dictate when it completes!
Handling shared resource

An example schedule of 2 jobs & 1 shared resource:

Core 1:

Core 2:

Releases:

Start time of a Job does not necessarily dictate when it completes!
Handling shared resource

An example schedule of 2 jobs & 1 shared resource:

Core 1:
- Job $J_1$ starts at time 0, finishes at time 7
- Job $J_2$ starts at time 5

Core 2:
- Job $J_2$ starts at time 3, finishes at time 10

Releases:
- Job $J_1$ released at time 0
- Job $J_2$ released at time 3

Keep track of shared resource availability!

Start time of a Job does not necessarily dictate when it completes!
Split a job into segments

Precedence constraint

Job Segment

Critical section length

Complete execution section length
Split a job into segments

- **Precedence constraint**

- **Job Segment**
  - $J_{i,1}$
  - $J_{i,2}$

- **Critical section length**

- **Complete execution section length**

**Schedule Abstraction Analysis operates on segment level**
System state abstraction

[ECRTS’19]
Available cores

\[
\begin{align*}
\{ & 1 \text{ core} \\
& 2 \text{ cores} \\
& \ldots \\
& m \text{ cores} \\
\}
\end{align*}
\]

[This work]
Available cores

\[
\begin{align*}
\{ & 1 \text{ core} \\
& 2 \text{ cores} \\
& \ldots \\
& m \text{ cores} \\
\}
\end{align*}
\]

Shared resources

\[
\begin{align*}
\{ & \text{Resource 1} \\
& \text{Resource 2} \\
& \ldots \\
& \text{Resource } r \\
\}
\end{align*}
\]
How to preserve non-preemptive execution?

An example schedule:

Core 1:

Releases:

$J_x$, $J_y$

$v_0 \rightarrow v_1$
How to preserve non-preemptive execution?

An example schedule:

Core 1:

Releases:

\[ J_x, 1 \]

\[ J_x \]

\[ J_y \]

\[ J_y, 1 \]

\[ J_x, 2 \]

\[ v_0 \rightarrow v_1 \]
How to preserve non-preemptive execution?

An example schedule:

Core 1:

\[ J_x \quad J_y \]

Releases:

\[ J_x \quad J_y \]

\[ J_x \quad J_x,1 \quad J_x,2 \]

\[ J_y \quad J_y,1 \]

\[ v_0 \quad v_1 \quad v_2 \]

\[ J_{x,1} \quad J_{y,1} \]
How to preserve non-preemptive execution?

An example schedule:

Core 1:

Releases:

\[ J_x \quad J_y \]

\[ J_x \quad J_y \]

\[ J_x, 1 \quad J_y, 1 \quad J_x, 2 \]

\[ v_0 \quad v_1 \quad v_2 \quad v_3 \]

\[ J_{x,1} \quad J_{y,1} \quad J_{x,2} \]
How to preserve non-preemptive execution?

An example schedule:

Core 1:

- $J_x$ running $J_{x,1}$
- $J_y$ running $J_{y,1}$
- $J_x$ running $J_{x,2}$

Releases:

- $J_x$ releases $J_y$
- $J_y$ releases $J_x$

Violates non-preemptive execution model!
How to preserve non-preemptive execution?

An example schedule:

Core 1: $J_x, J_y, J_x, J_y$

Releases: $J_x, J_y$

Violates non-preemptive execution model!
How to preserve non-preemptive execution?

Let jobs claim cores for their execution and release them after they finish.

An example schedule:

Core 1: $J_x,1 \rightarrow J_y,1 \rightarrow J_x,2$

Releases: $J_x$, $J_y$

Violates non-preemptive execution model!

Let jobs claim cores for their execution and release them after they finish.
How to preserve non-preemptive execution?

Let jobs claim cores for their execution and release them after they finish.

An example schedule:

Core 1: $J_x,1$  

Releases: $J_x,1$, $J_y$

Claimed for $J_x$

$J_x$, $J_x,1$, $J_x,2$

$J_y$, $J_y,1$

$J_{x,1}$ $v_0$ $v_1$
How to preserve non-preemptive execution?

Let jobs claim cores for their execution and release them after they finish.

An example schedule:

Core 1:

- $J_x$ (Jx,1 - Jx,2)
- $J_y$

Releases:

- $J_x,1$
- $J_x,2$

Free for any Job

Let jobs claim cores for their execution and release them after they finish.

$J_x$ (Jx,1 - Jx,2)

$J_y$ (Jy,1)

$v_0 \rightarrow v_1 \rightarrow J_{x,2} \rightarrow v_2$

$v_0$ (v0)

$v_1$ (v1)

$v_2$ (v2)
How to preserve non-preemptive execution?

An example schedule:

Core 1:

- \( J_x \)
- \( J_y \)

Releases:

- \( v_1 \)
- \( v_2 \)

Let jobs claim cores for their execution and release them after they finish.
System state abstraction

[ECRTS’19]

Available cores

\[
\begin{align*}
&1 \text{ core} \\
&2 \text{ cores} \\
&\ldots \\
&m \text{ cores}
\end{align*}
\]

Free cores

\[
\begin{align*}
&1 \text{ core} \\
&\ldots \\
&k \text{ cores}
\end{align*}
\]

Claimed cores

\[
\begin{align*}
&\text{Core 1} \\
&\ldots \\
&\text{Core } c
\end{align*}
\]

Shared resources

\[
\begin{align*}
&\text{Resource 1} \\
&\text{Resource 2} \\
&\ldots \\
&\text{Resource } r
\end{align*}
\]

[This work]
Response-Time Analysis for Non-Preemptive Global Scheduling with Spin Locks

Agenda

To do’s:

- Design a system state representation
- Design expansion and merge rules
Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource
Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

→ Potentially ready segments
Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

- Potentially ready segments
- Earliest and latest start times
Expansion and merge rules

Expansion and *merge* rules to support the new system state abstraction and access to shared resource

- Potentially ready segments
- Earliest and latest start times
- Eligibility condition

Path

... → V → ...

Potentialities...
Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

- Potentially ready segments
- Earliest and latest start times
- Eligibility condition
- Earliest and latest finish times
Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

- Potentially ready segments
- Earliest and latest start times
- Eligibility condition
- Earliest and latest finish times
- Update availabilities

Path

\[ ... \rightarrow V \rightarrow \ldots \rightarrow ? \]
Expansion and merge rules

**Expansion** and **merge** rules to support the new system state abstraction and access to shared resource

- Potentially ready segments
- Earliest and latest start times
- Eligibility condition
- Earliest and latest finish times
- Update availabilities
- Sound merging

![Diagram of a path](image)
Expansion and merge rules

Expansion and merge rules to support the new system state abstraction and access to shared resource

- Potentially ready segments
- Earliest and latest start times
- Eligibility condition
- Earliest and latest finish times
- Update availabilities
- Sound merging

Path

...
Agenda

To do’s:

- Design a system state representation
- Design expansion and merge rules
Evaluation
### Evaluation baselines

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{EDF,FP}-SAG-SR-{FIFO,PRIOR}</td>
<td>Scenario-aware analysis</td>
<td>Our approach</td>
</tr>
</tbody>
</table>

---


### Evaluation baselines

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{EDF,FP}-SAG-SR-{FIFO,PRIO}</td>
<td>Scenario-aware analysis</td>
<td>Our approach</td>
</tr>
<tr>
<td>{EDF,FP}-SAG-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (NP+SAG)</td>
</tr>
<tr>
<td>{EDF,FP}-SAG-INF-{FIFO,PRIO}</td>
<td>Inflation-based analysis</td>
<td>SAG + inflation-based analysis</td>
</tr>
</tbody>
</table>

---


## Evaluation baselines

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{EDF,FP}-SAG-SR-{FIFO,PRIO}</td>
<td>Scenario-aware analysis</td>
<td>Our approach</td>
</tr>
<tr>
<td>{EDF,FP}-SAG-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (NP+SAG)</td>
</tr>
<tr>
<td>{EDF,FP}-SAG-INF-{FIFO,PRIO}</td>
<td>Inflation-based analysis</td>
<td>SAG + inflation-based analysis</td>
</tr>
<tr>
<td>EDF-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (P+EDF)</td>
</tr>
<tr>
<td>FP-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (P+FP)</td>
</tr>
</tbody>
</table>


## Evaluation baselines

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{EDF,FP}-SAG-SR-{FIFO,PRIO}</td>
<td>Scenario-aware analysis</td>
<td>Our approach</td>
</tr>
<tr>
<td>{EDF,FP}-SAG-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (NP+SAG)</td>
</tr>
<tr>
<td>{EDF,FP}-SAG-INF-{FIFO,PRIO}</td>
<td>Inflation-based analysis</td>
<td>SAG + inflation-based analysis</td>
</tr>
<tr>
<td>EDF-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (P+EDF)</td>
</tr>
<tr>
<td>FP-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (P+FP)</td>
</tr>
<tr>
<td>{EDF,FP}-FMLP-SHORT</td>
<td>Inflation-based analysis</td>
<td>Inflation-based analysis considering FMLP &amp; short critical sections</td>
</tr>
<tr>
<td>{EDF,FP}-OMLP</td>
<td>Inflation-based analysis</td>
<td>Inflation-based analysis considering OMLP</td>
</tr>
</tbody>
</table>


## Evaluation baselines

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{EDF,FP}-SAG-SR-{FIFO,PRIO}</td>
<td>Scenario-aware analysis</td>
<td>Our approach</td>
</tr>
<tr>
<td>{EDF,FP}-SAG-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (NP+SAG)</td>
</tr>
<tr>
<td>{EDF,FP}-SAG-INF-{FIFO,PRIO}</td>
<td>Inflation-based analysis</td>
<td>SAG + inflation-based analysis</td>
</tr>
<tr>
<td>EDF-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (P+EDF)</td>
</tr>
<tr>
<td>FP-NO-BLOCKING</td>
<td>---</td>
<td>Upper bound on attainable schedulability (P+FP)</td>
</tr>
<tr>
<td>{EDF,FP}-FMLP-SHORT</td>
<td>Inflation-based analysis</td>
<td>Inflation-based analysis considering FMLP &amp; short critical sections</td>
</tr>
<tr>
<td>{EDF,FP}-OMLP</td>
<td>Inflation-based analysis</td>
<td>Inflation-based analysis considering OMLP</td>
</tr>
<tr>
<td>FP-FMLP-LONG</td>
<td>LP-based analysis</td>
<td>Linear-programming-based analysis for FMLP &amp; long critical sections</td>
</tr>
<tr>
<td>FP-PIP</td>
<td>LP-based analysis</td>
<td>Linear-programming-based analysis for PIP</td>
</tr>
</tbody>
</table>


Evaluation

Goal:

→ Evaluate the accuracy of the analysis
Evaluation

Goal:

→ Evaluate the accuracy of the analysis

Metric:

→ Schedulability Ratio: \[ \frac{\text{Number of task sets deemed schedulable}}{\text{Number of total evaluated task sets}} \]
Evaluation

Goal:

→ Evaluate the accuracy of the analysis

Metric:

→ Schedulability Ratio: \( \frac{Number\ of\ task\ sets\ deemed\ schedulable}{Number\ of\ total\ evaluated\ task\ sets} \)

Task sets:

→ Randomly generated
Results: SAG comparison

Legend:
- \( m \): Number of cores
- \( n \): Number of tasks
- \( n_{cs} \): Maximum per task critical sections
- \( n_r \): Number of shared resources
- \( L_{max} \): Maximum length of critical sections

Graph showing the relationship between schedulability ratio and total utilization for different values of \( m \) and \( n \). The graph indicates the performance of EDF-SAG-NO-BLOCKING scheduling algorithm with given parameters.
Results: SAG comparison

Legend:
- \( m \): Number of cores
- \( n \): Number of tasks
- \( n_{cs} \): Maximum per task critical sections
- \( n_r \): Number of shared resources
- \( L_{\text{max}} \): Maximum length of critical sections

\( m=4 \ n=8 \ n_{cs}=15 \ n_r=20 \ L_{\text{max}} \in [50,150] \)

- EDF-SAG-NO-BLOCKING
- EDF-SAG-INFLATION-FIFO
- EDF-SAG-INFLATION-PRI0
Results: SAG comparison

Legend:
- $m$: Number of cores
- $n$: Number of tasks
- $n_{cs}$: Maximum per task critical sections
- $n^r$: Number of shared resources
- $L^{max}$: Maximum length of critical sections
Results: EDF comparison

Legend:
- $m$: Number of cores
- $n$: Number of tasks
- $n_{cs}$: Maximum per task critical sections
- $n^r$: Number of shared resources
- $L_{max}$: Maximum length of critical sections
Results: EDF comparison

Legend:
- $m$: Number of cores
- $n$: Number of tasks
- $n_{cs}$: Maximum per task critical sections
- $n^r$: Number of shared resources
- $L_{max}$: Maximum length of critical sections
Results: EDF comparison

Legend:
- \( m \): Number of cores
- \( n \): Number of tasks
- \( n_{cs} \): Maximum per task critical sections
- \( n_r \): Number of shared resources
- \( L_{max} \): Maximum length of critical sections
Results: FP comparison 1

Legend:
- \( m \): Number of cores
- \( n \): Number of tasks
- \( n_{cs} \): Maximum per task critical sections
- \( n_r \): Number of shared resources
- \( L_{max} \): Maximum length of critical sections

Graph parameters:
- \( m = 4 \)
- \( n = 8 \)
- \( n_{cs} = 15 \)
- \( n_r = 8 \)
- \( L_{max} \in [50,150] \)
Results: FP comparison 1

Legend:
- $m$: Number of cores
- $n$: Number of tasks
- $n_{cs}$: Maximum per task critical sections
- $n^r$: Number of shared resources
- $L_{max}$: Maximum length of critical sections
Results: FP comparison 1

Legend:
- $m$: Number of cores
- $n$: Number of tasks
- $n_{cs}$: Maximum per task critical sections
- $n_r$: Number of shared resources
- $L_{\text{max}}$: Maximum length of critical sections

Graph: Comparison of FP algorithms with varying total utilization (%). The graph shows the schedulability ratio for $m=4$, $n=8$, $n_{cs}=15$, $n_r=8$, and $L_{\text{max}} \in [50,150]$. The algorithms compared include FP-NO-BLOCKING, FP-FMLP-SHORT, FP-OMLP, FP-FMLP-LONG, FP-PIP, FP-SAG-NO-BLOCKING, and FP-SAG-SR-PRIOR.
Results: FP comparison 2

Legend:
- $m$: Number of cores
- $n$: Number of tasks
- $n^{cs}$: Maximum per task critical sections
- $n^{r}$: Number of shared resources
- $L_{max}$: Maximum length of critical sections

$m=4$ $n=5$ $ncs=15$ $nr=20$ $L_{max} \in [50,150]$
Response-Time Analysis for Non-Preemptive Global Scheduling with Spin Locks
Extensions of the work
Partial order reduction

What is it?
→ Smart techniques to prune the graph

What is the goal?
→ Slow down the growth of the graph to improve scalability

What is the key idea?
→ Find states where branching is redundant!
Partial order reduction

An example schedule:

Core 1:  

Core 2:  

$J_1$ & $J_2$

Releases: 0

$(J_{1,1} \text{ )}$

$(J_{2,1} \text{ )}$
Partial order reduction

An example schedule:

Core 1: $J_{1,1}$

Core 2:

Releases:

$J_1 \& J_2$

$(J_{1,1})$

$(J_{2,1})$

$J_{1,1} \rightarrow v_1$

$v_0 \rightarrow v_1$

0 8

0 8

0 6
Partial order reduction

An example schedule:

Core 1:

\[ J_{1,1} \]

0 \[ \rightarrow \] 8

Core 2:

\[ J_{2,1} \]

0 \[ \rightarrow \] 6

\[ J_1 \] & \[ J_2 \]

Releases:

\[ (J_{1,1}) \]

0 \[ \rightarrow \] 8

\[ (J_{2,1}) \]

0 \[ \rightarrow \] 6

Graph representation:

\[ J_{1,1} \rightarrow v_1 \rightarrow J_{2,1} \rightarrow v_2 \]

\[ v_0 \]
Partial order reduction

An example schedule:

Core 1:

\( J_{2,1} \)

\[
\begin{align*}
\text{Core 1:} & \quad 0 \quad \text{6} \\
\text{Core 2:} & \quad \\
\text{Releases:} & \quad 0 \\
\end{align*}
\]

\( J_1 \) & \( J_2 \)

\[
\begin{align*}
(J_{1,1}) & \quad 8 \\
(J_{2,1}) & \quad 6 \\
\end{align*}
\]
Partial order reduction

An example schedule:

Core 1:

Core 2:

Releases:

(J₁₁) 8

(J₂₁) 6

\(J₁₁\) & \(J₂₁\)

\((J₁₁)\) 8

\((J₂₁)\) 6

\(v₀\)

\(v₁\)

\(v₂\)

\(v₁'\)

\(v₂'\)
Partial order reduction

An example schedule:

Core 1:

\[ J_{1,1} \]

Core 2:

\[ J_{2,1} \]

Releases:

\[ (J_{1,1}) \]

\[ (J_{2,1}) \]
Partial order reduction

An example schedule:

Core 1: $J_{1,1}$

0 \rightarrow 8

Core 2: $J_{2,1}$

0 \rightarrow 6

Releases: 0 \rightarrow 8

$J_1$ & $J_2$

As long as we do not affect other job executions!
Partial order reduction

(1) Non-starting segments that don’t access shared resources
→ Combined in workload representation
Partial order reduction

(1) Non-starting segments that don’t access shared resources
   → Combined in workload representation

(2) Non-starting segments that access shared resources
Partial order reduction

(1) Non-starting segments that don’t access shared resources
   → Combined in workload representation

(2) Non-starting segments that access shared resources

(3) Starting segments that don’t access shared resources
Multi-unit resources

What is it?

→ Shared resources that can be accessed by more than one task simultaneously
Multi-unit resources

What is it?

→ Shared resources that can be accessed by more than one task simultaneously

Why include it?

→ Common in real-time systems to find similar multiple units of the same resource
  • Communication channels
  • I/O buffers
Multi-unit resources

Idea:

→ Keep track when 1, 2, ..., k units of the same resource become(s) available

Shared
resource 1

1 unit

2 units

... 

k units
Summary & Conclusion
Summary

**First** blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks
Summary

First blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks

Extended the family of schedule-abstraction-based analysis to account for shared resources
Summary

First blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks

Extended the family of schedule-abstraction-based analysis to account for shared resources

Response-time analysis using schedule abstraction

+ New abstraction
Summary

First blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks

Extended the family of schedule-abstraction-based analysis to account for shared resources

Response-time analysis using schedule abstraction

+ New abstraction
+ Expansion and merge rules for shared resources supporting fifo- and priority-based spin locks
Summary

First blocking-aware response-time analysis for global non-preemptive systems that share resources via spin locks

Extended the family of schedule-abstraction-based analysis to account for shared resources

Response-time analysis using schedule abstraction

+ New abstraction
+ Expansion and merge rules for shared resources supporting fifo- and priority-based spin locks
+ Extended the work via partial order reduction techniques and to support multi-unit resources
Conclusion

+ Our scenario-aware SAG-based analysis significantly reduces pessimism that exists in current analyses.
Conclusion

+ **Our scenario-aware SAG-based analysis significantly reduces pessimism** that exists in current analyses

+ **Our scenario-aware SAG analysis** beats its inflation-based variant.
Conclusion

+ Our scenario-aware SAG-based analysis significantly reduces pessimism that exists in current analyses

+ Our scenario-aware SAG analysis beats its inflation-based variant

+ Our scenario-aware SAG analysis competes with and (sometimes even surpasses) analyses considering a preemptive system
Conclusion

+ Our scenario-aware SAG-based analysis **significantly reduces pessimism** that exists in current analyses

+ Our scenario-aware SAG analysis **beats its inflation-based variant**

+ Our scenario-aware SAG analysis **competes with and (sometimes even surpasses)** analyses considering a preemptive system

+ **Opens further research paths in this direction**
Future work

• Support for DAG tasks
• Support for nested locks
• Support for suspension-based locks
Response-time analysis considering FIFO-ordered spin locks

➢ Support for multi-unit resources & k-exclusion protocols
➢ Additional techniques to improve scalability

Response-time analysis considering priority-ordered spin locks

CAPITAL

RTSS

Thank YOU
Backup slides
Runtime results

\[ m = 4 \quad n = 5 \quad n^{cs} = 5 \quad n^{r} = 8 \quad L^{max} \in [1, 15] \]
Runtime results

\[ m = 4 \quad n = 5 \quad n^{cs} = 15 \quad n^r = 8 \quad L^{max} \in [1, 15] \]
Runtime results

\[ m = 4 \quad n = 8 \quad n^{cs} = 15 \quad n^r = 8 \quad L^{max} \in [1, 15] \]
Runtime results

EDF-SAG-SR-PRI0 95th Percentile

runtime (in seconds)

5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

total utilization (%)
Runtime results

EDF-SAG-SR-PRIO 98th Percentile

runtime (in seconds)

5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

total utilization (%)