MultiSSE: Static Syscall Elision and Specialization for Event-Triggered Multi-Core RTOS

Gerion Entrup, Björn Fiedler, Daniel Lohmann
{entrup, fiedler, lohmann}@sra.uni-hannover.de

Leibniz Universität Hannover

May 12, 2023
Optimization: RTOS Tailoring

Core 1

T11

Priority: 1
Autostart: ⊠

10 GetSpinlock(S1);
11 do _critical();
12 ReleaseSpinlock(S1);
14 ActivateTask(T02);
16 ActivateTask(T01);
18 ActivateTask(T21);
20 TerminateTask();
Optimization: RTOS Tailoring

T11
Priority: 1
Autostart: ⊠
10 GetSpinlock(S1);
11 do _critical();
12 ReleaseSpinlock(S1);
14 ActivateTask(T02);
16 ActivateTask(T01);
18 ActivateTask(T21);
20 TerminateTask();

Core 0

Core 1

Spinlock S1

Express control flow with Atomic Basic Blocks [SCH10]

Build the Multi-State Transition Graph (MSTG):
1. Capture core-local effects in local states.
2. In case of a cross core system call: Build a multicore state. We call them synchronization points (SPs).
3. Interpret that state.
4. Split the multicore state and locally interpret the local states.
5. Merge duplicated states (here "idle")
6. Find all possible pairing partners (states that execute simultaneously).
7. Iterate (step 2) until stabilization.

Initialization: A synchronization point over all cores.

SP a (Act.Task(T02))
α (07)
β (15)
γ (idle)
Act.Task(T01)
α = [60, 80]
β = [10, 20]
γ = [0, ∞]
α + γ = β

Solution: α = β = [10, 20]
Optimization: RTOS Tailoring

Core 0

T02

Priority: 2
Autostart: □

07 do_high_priority();
08 TerminateTask();

T01

Priority: 1
Autostart: □

02 GetSpinlock(S1);
03 do_critical();
04 ReleaseSpinlock(S1);
06 TerminateTask();

Core 1

T11

Priority: 1
Autostart: ✗

10 GetSpinlock(S1);
11 do_critical();
12 ReleaseSpinlock(S1);
14 ActivateTask(T02);
16 ActivateTask(T01);
18 ActivateTask(T21);
20 TerminateTask();

Spinlock S1

Gerion Entrup MultiSSE – Motivation
### Optimization: RTOS Tailoring

#### Core 0

**T02**
- **Priority:** 2
- **Autostart:** False

07. `do_high_priority();`
08. `TerminateTask();`

---

**T01**
- **Priority:** 1
- **Autostart:** False

02. `GetSpinlock(S1);`
03. `do_critical();`
04. `ReleaseSpinlock(S1);`
06. `TerminateTask();`

#### Core 1

**T11**
- **Priority:** 1
- **Autostart:** False

10. `GetSpinlock(S1);`
11. `do_critical();`
12. `ReleaseSpinlock(S1);`
14. `ActivateTask(T02);`
16. `ActivateTask(T01);`
18. `ActivateTask(T21);`
20. `TerminateTask();`

#### Spinlock S1

#### Core 2

**T21**
- **Priority:** 1
- **Autostart:** False

22. `GetSpinlock(S1);`
23. `do_critical();`
24. `ReleaseSpinlock(S1);`
26. `TerminateTask();`
Optimization: RTOS Tailoring

### Core 0

- **T01**
  - Priority: 1
  - Autostart: □
  - Sequence:
    - 02 GetSpinlock(S1);
    - 03 do_critical();
    - 04 ReleaseSpinlock(S1);
    - 06 TerminateTask();

- **T02**
  - Priority: 2
  - Autostart: □
  - Sequence:
    - 07 do_high_priority();
    - 08 TerminateTask();

### Core 1

- **T11**
  - Priority: 1
  - Autostart: ☑
  - Sequence:
    - 10 GetSpinlock(S1);
    - 11 do_critical();
    - 12 ReleaseSpinlock(S1);
    - 14 ActivateTask(T02);
    - 16 ActivateTask(T01);
    - 18 ActivateTask(T21);
    - 20 TerminateTask();

### Core 2

- **T21**
  - Priority: 1
  - Autostart: □
  - Sequence:
    - 22 GetSpinlock(S1);
    - 23 do_critical();
    - 24 ReleaseSpinlock(S1);
    - 26 TerminateTask();

- GetSpinlock(S1) is superfluous → **elide syscall**
Optimization: RTOS Tailoring

- GetSpinlock(S1) is superfluous → elide syscall
Optimization: RTOS Tailoring

Core 0

T02

Priority: 2
Autostart: □
07 do_high_priority();
08 TerminateTask();

T01

Priority: 1
Autostart: □
02 GetSpinlock(S1);
03 do_critical();
04 ReleaseSpinlock(S1);
06 TerminateTask();

Core 1

T11

Priority: 1
Autostart: ⊠
10 GetSpinlock(S1);
11 do_critical();
12 ReleaseSpinlock(S1);
14 ActivateTask(T02);
16 ActivateTask(T01);
18 ActivateTask(T21);
20 TerminateTask();

Core 2

T21

Priority: 1
Autostart: □
22 GetSpinlock(S1);
23 do_critical();
24 ReleaseSpinlock(S1);
26 TerminateTask();

Spinlock S1

- GetSpinlock(S1) is superfluous → elide syscall
- ActivateTask(T01) just sets ready not active → avoid IPI
Optimization: RTOS Tailoring

- GetSpinlock(S1) is superfluous → **elide syscall**
- ActivateTask(T01) just sets ready not active → **avoid IPI**
- All critical sections protected by S1 runs serialized → **elide S1**
How to extract information

- Optimization only possible with appropriate information.
- We need a proper analysis.
How to extract information

- Optimization only possible with appropriate information.
- We need a proper analysis.

MultiSSE

- Static analysis
- Designed for detecting multicore interleavings
- Implemented within ARA [Fie+21] (a whole system analyzer)
Basic idea

- Single-core analysis already present: System-State Enumeration (SSE)
  - Developed by Dietrich [DHL15]
  - Calculation of all possible RTOS states for a given system

- Naive approach: Cross product of all that states → **combinatoric explosion**

**Observation**

- Cross-core interactions are neither random nor frequent.
- Bound to statically determinable syscalls and interrupts.
Basic idea

- Single-core analysis already present: System-State Enumeration (SSE)
  - Developed by Dietrich [DHL15]
  - Calculation of all possible RTOS states for a given system
- Naive approach: Cross product of all that states → combinatoric explosion

Observation

- Cross-core interactions are neither random nor frequent.
- Bound to statically determinable syscalls and interrupts.

Idea

Calculate cross core interleavings just for that specific parts.
MultiSSE in a nutshell

Core 0

T02
- Priority: 2
- Autostart: □
  07 do_high_priority();
  08 TerminateTask();

T01
- Priority: 1
- Autostart: □
  02 GetSpinlock(S1);
  03 do_critical();
  04 ReleaseSpinlock(S1);
  06 TerminateTask();

Core 1

T11
- Priority: 1
- Autostart: ⊠
  10 GetSpinlock(S1);
  11 do_critical();
  12 ReleaseSpinlock(S1);
  14 ActivateTask(T02);
  16 ActivateTask(T01);
  18 ActivateTask(T21);
  20 TerminateTask();

Spinlock S1

Core 2

T21
- Priority: 1
- Autostart: □
  22 GetSpinlock(S1);
  23 do_critical();
  24 ReleaseSpinlock(S1);
  26 TerminateTask();
MultiSSE in a nutshell

Core 0

T02
Priority: 2
Autostart: □
07 do_high_priority(); 60-80
08 TerminateTask();

T01
Priority: 1
Autostart: □
02 GetSpinlock(S1);
03 do_critical();
04 ReleaseSpinlock(S1);
06 TerminateTask();

Core 1

T11
Priority: 1
Autostart: ⊠
10 GetSpinlock(S1);
11 do_critical();
12 ReleaseSpinlock(S1);
14 ActivateTask(T02); 10-20
16 ActivateTask(T01);
18 ActivateTask(T21);
20 TerminateTask();

Gerion Entrup MultiSSE – Approach
MultiSSE in a nutshell

T11
Priority: 1
Autostart: ⊠
10 GetSpinlock(S1);
11 do _critical();
12 ReleaseSpinlock(S1);
14 ActivateTask(T02);
16 ActivateTask(T01);
18 ActivateTask(T21);
20 TerminateTask();

T01
Priority: 1
Autostart: □
02 GetSpinlock(S1);
03 do_critical();
04 ReleaseSpinlock(S1);
06 TerminateTask();

T02
Priority: 2
Autostart: □
07 do_high_priority();
08 TerminateTask();

Core 0

Core 1

Core 0

Core 1

T11 starts

Act.Task(T02)

idle

high_priority()

TerminateTask()

idle

Gerion Entrup  MultiSSE – Approach
MultiSSE in a nutshell

Core 0

T02
- Priority: 2
- Autostart: □
07 do_high_priority();
08 TerminateTask();

T01
- Priority: 1
- Autostart: □
02 GetSpinlock(S1);
03 do_critical();
04 ReleaseSpinlock(S1);
06 TerminateTask();

Core 1

T11
- Priority: 1
- Autostart: ⊠
10 GetSpinlock(S1);
11 do_critical();
12 ReleaseSpinlock(S1);
14 ActivateTask(T02);
16 ActivateTask(T01);
18 ActivateTask(T21);
20 TerminateTask();

Core 0

idle

Act.Task(T02)

Core 1

T11 starts

idle

high_priority()

Act.Task(T01)

T01

Gerion Entrup

MultiSSE – Approach
MultiSSE in a nutshell

Core 0

<table>
<thead>
<tr>
<th>T02</th>
<th>Priority: 2</th>
<th>Autostart: □</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>do_high_priority();</td>
<td>60-80</td>
</tr>
<tr>
<td>08</td>
<td>TerminateTask();</td>
<td></td>
</tr>
</tbody>
</table>

Core 1

<table>
<thead>
<tr>
<th>T11</th>
<th>Priority: 1</th>
<th>Autostart: ☑</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>GetSpinlock(S1);</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>do_critical();</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>ReleaseSpinlock(S1);</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>ActivateTask(T02);</td>
<td>10-20</td>
</tr>
<tr>
<td>16</td>
<td>ActivateTask(T01);</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>ActivateTask(T21);</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>TerminateTask();</td>
<td></td>
</tr>
</tbody>
</table>

Core 0

- idle
- high_priority()
- TerminateTask()

Core 1

- T11 starts
- Act.Task(T02)
- Act.Task(T01)

T02
- Priority: 2
- Autostart: □

T01
- Priority: 1
- Autostart: □

Gerion Entrup
MultiSSE – Approach 5 – 15
MultiSSE in a nutshell

Core 0

T02
Priority: 2
Autostart: □
07 do_high_priority(); 60-80
08 TerminateTask();

T01
Priority: 1
Autostart: □
02 GetSpinlock(S1);
03 do_critical();
04 ReleaseSpinlock(S1);
06 TerminateTask();

Core 1

T11
Priority: 1
Autostart: ⊠
10 GetSpinlock(S1);
11 do_critical();
12 ReleaseSpinlock(S1);
14 ActivateTask(T02);
16 ActivateTask(T01);
18 ActivateTask(T21);
20 TerminateTask();

Core 0
Core 1

idle

13
14
T11 starts
Act.Task(T02)

15
16
Act.Task(T01)

07 high_priority()

08
TerminateTask()

T01

Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]
Build the Multi-State Transition Graph (MSTG):
1. Capture core-local effects in **local states**.

Express control flow with Atomic Basic Blocks (ABBs) [SCH10]
MultiSSE in a nutshell

Build the Multi-State Transition Graph (MSTG):
1. Capture core-local effects in **local states**.
2. In case of a cross core system call:
   Build a **multicore state**.
   We call them **synchronization points (SPs)**.

Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]
MultiSSE in a nutshell

Build the Multi-State Transition Graph (MSTG):
1. Capture core-local effects in **local states**.
2. In case of a cross core system call:
   - Build a **multicore state**.
   - We call them **synchronization points (SPs)**.
3. Interpret that state.

Express control flow with
**Atomic Basic Blocks (ABBs)** [SCH10]
MultiSSE in a nutshell

Build the Multi-State Transition Graph (MSTG):
1. Capture core-local effects in **local states**.
2. In case of a cross core system call:
   - Build a **multicore state**.
   - We call them **synchronization points (SPs)**.
3. Interpret that state.
4. Split the **multicore state** and locally interpret the **local states**.
5. Merge duplicated states (here "idle")

Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]
MultiSSE in a nutshell

Build the Multi-State Transition Graph (MSTG):
1. Capture core-local effects in local states.
2. In case of a cross core system call:
   Build a multicore state.
   We call them synchronization points (SPs).
3. Interpret that state.
4. Split the multicore state and locally interpret the local states.
5. Merge duplicated states (here "idle")
6. Find all possible pairing partners (states that execute simultaneously).

Express control flow with Atomic Basic Blocks (ABBs) [SCH10]
MultiSSE in a nutshell

Build the Multi-State Transition Graph (MSTG):

1. Capture core-local effects in **local states**.
2. In case of a cross core system call:
   Build a **multicore state**.
   We call them **synchronization points (SPs)**.
3. Interpret that state.
4. Split the **multicore state** and
   locally interpret the **local states**.
5. Merge duplicated states (here "idle")
6. Find all possible pairing partners
   (states that execute simultaneously).
7. Iterate (step 2) until stabilization.
   Initialization: A synchronization point over all cores.

Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]
MultiSSE in a nutshell

Build the Multi-State Transition Graph (MSTG):
1. Capture core-local effects in **local states**.
2. In case of a cross core system call:
   - Build a **multicore state**.
   - We call them **synchronization points (SPs)**.
3. Interpret that state.
4. Split the **multicore state** and locally interpret the **local states**.
5. Merge duplicated states (here "idle")
6. **Find all possible pairing partners** (states that execute simultaneously).
7. Iterate (step 2) until stabilization.
   - Initialization: A synchronization point over all cores.

Express control flow with Atomic Basic Blocks (ABBs) [SCH10]
Pairing Partner Search

- Pairing Partners of a cross-core system call: States on other cores that can execute simultaneously.
- Restrict partner set based on the given control flow.
- Only possible states: After the last common SP.
Pairing Partner Search: Visualization
Pairing Partner Search: Visualization

Core 1  Core 2  Core 3

Core 1  Core 2

1 2 3
1 2
2 3
a
b
c

2 3 3 4
d  e
1 2 3 4
1 2
2 3 4
a
b
c

Gerion Entrup  MultiSSE – Analysis
Pairing Partner Search: Visualization

Gerion Entrup  MultiSSE – Analysis

Gerion Entrup  MultiSSE – Analysis
Pairing Partner Search: Visualization

Gerion Entrup MultiSSE – Analysis
Pairing Partner Search: Visualization
Pairing Partner Search: Visualization

- **Core 1**
- **Core 2**
- **Core 3**

Diagram showing the pairing process with nodes labeled as 1, 2, 3, and 4, and arrows indicating connections between them.
Pairing Partner Search: Visualization
Timing Information

- Pairing partner search restricts interleavings by their control flow.
- We have an additional dimension: The execution time.
- Restrict pairing partners even further with time information.
- Given: BCET and WCET of individual control flow blocks.

Main Question

Can the affected cores execute two control flow blocks simultaneously?

Solution

Use linear programming (solving of inequality systems).
Timing Information

a) 

Cross core system call on core "x".

Question: Can the requested state with interval $\beta$ happen simultaneously?

Solution: Yes, if $\alpha = \beta$ has a solution.

$\alpha = [\alpha_{\text{min}}, \alpha_{\text{max}}]$  
$\beta = [0, \beta_{\text{max}}]$  
$\alpha = \beta$
Timing Information

a) Cross core system call on core "x".

Question: Can the requested state with interval $\beta$ happen simultaneously?

Solution: Yes, if $\alpha = \beta$ has a solution.

b) Known intervals between SPs.

MultiSSE stores solutions.

Avoid recalculations.

\[
\alpha = [\alpha_{\text{min}}, \alpha_{\text{max}}] \\
\beta = [0, \beta_{\text{max}}] \\
\gamma = [\gamma_{\text{min}}, \gamma_{\text{max}}]
\]

\[
\alpha = \beta + \gamma
\]
Timing Information – Complex Cases

\[ \alpha = [\alpha_{\text{min}}, \alpha_{\text{max}}] \]
\[ \beta = [0, \beta_{\text{max}}] \]
\[ \gamma = [\gamma_{\text{min}}, \gamma_{\text{max}}] \]
\[ \delta = [0, \alpha_{\text{max}}] \]
\[ \theta = [\theta_{\text{min}}, \theta_{\text{max}}] \]
\[ \epsilon = [0, \alpha_{\text{max}}] \]
\[ \eta = [\eta_{\text{min}}, \eta_{\text{max}}] \]
\[ \alpha = \epsilon + \delta \]
\[ \gamma = \eta + \epsilon \]
\[ \beta = \delta + \theta \]

Please read the paper for details.
Timing Information – Complex Cases

\[
\begin{align*}
\alpha &= [\alpha_{\min}, \alpha_{\max}] \\
\beta &= [0, \beta_{\max}] \\
\gamma &= [\gamma_{\min}, \gamma_{\max}] \\
\delta &= [0, \delta_{\max}] \\
\theta &= [\theta_{\min}, \theta_{\max}] \\
\epsilon &= [0, \epsilon_{\max}] \\
\eta &= [\eta_{\min}, \eta_{\max}] \\
\alpha &= \epsilon + \delta \\
\gamma &= \eta + \epsilon \\
\beta &= \delta + \theta
\end{align*}
\]

Please read the paper for details.
Analysis of the Example With Timing Information

Expression of control flow with Atomic Basic Blocks (ABBs) [SCH10]

Build the Multi-State Transition Graph (MSTG):
1. Capture core-local effects in local states.
2. In case of a cross core system call: Build a multicore state. We call them synchronization points (SPs).
3. Interpret that state.
4. Split the multicore state and locally interpret the local states.
5. Merge duplicated states (here "idle").
6. Find all possible pairing partners (states that execute simultaneously).
7. Iterate (step 2) until stabilization.

Initialization: A synchronization point over all cores.

SP a (Act.Task(T02))

α (07)
β (15)
γ (idle)

α = [0, 80]
β = [10, 20]  Solution: α = β = [10, 20] ✓
α = β

Core 0

idle
07 15
07 15
08 07 16
idle
idle 16

Core 1

T11 starts

07

08

TerminateTask()

high_priority()

T01

Act.Task(T01)

Act.Task(T02)

Gerion Entrup MultiSSE – Analysis
Analysis of the Example With Timing Information

Express control flow with Atomic Basic Blocks (ABBs) [SCH10]

Core 0

07 idle
07 high_priority()
08 TerminateTask()

β (15)

Act.Task(T01)

Core 1

13 T11 starts
14 Act.Task(T02)
15

α (07)

 idle

 idle 07 15

 idle 08

 idle 14

 idle 16

α = [0, 80]
β = [10, 20]
α = β ✓

Gerion Entrup
MultiSSE – Analysis
Analysis of the Example With Timing Information

T11

10 GetSpinlock(S1);
11 do _critical();
12 ReleaseSpinlock(S1);
14 ActivateTask(T02);
16 ActivateTask(T01);
18 ActivateTask(T21);
20 TerminateTask();

T01

02 GetSpinlock(S1);
03 do _critical();
04 ReleaseSpinlock(S1);
06 TerminateTask();

T02

07 do _high_priority();
08 TerminateTask();

T21

22 GetSpinlock(S1);
23 do _critical();
24 ReleaseSpinlock(S1);
26 TerminateTask();

Core 0

Core 1

Spinlock S1

GetSpinlock(S1) is superfluous → elide syscall

ActivateTask(T01) just sets ready not active → avoid IPI

All critical sections protected by S1 runs serialized → elide S1

Core 0

Core 1

idle

high_priority()

TerminateTask()

Act.Task(T01)

idle

Act.Task(T01)

idle

T01

T11 starts

Act.Task(T02)

Act.Task(T01)

T01

Express control flow with Atomic Basic Blocks (ABBs) [SCH10]

Build the Multi-State Transition Graph (MSTG):
1. Capture core-local effects in local states.
2. In case of a cross core system call: Build a multicore state. We call them synchronization points (SPs).
3. Interpret that state.
4. Split the multicore state and locally interpret the local states.
5. Merge duplicated states (here "idle").
6. Find all possible pairing partners (states that execute simultaneously).
7. Iterate (step 2) until stabilization.

Initialization: A synchronization point over all cores.

SP a (Act.Task(T02))

α (07) = [60, 80]
β (15) = [10, 20]
γ (idle) = [0, ∞]

α + γ = β

Solution:

α = β ✓

Gerion Entrup MultiSSE – Analysis 11 – 15
Analysis of the Example With Timing Information

Expression control flow with Atomic Basic Blocks (ABBs) [SCH10]

SP a (Act.Task(T02))

Core 0

Core 1

T11 starts

T01

Act.Task(T02)

Act.Task(T01)

Act.Task(T01)

Act.Task(T02)

SP a (Act.Task(T02))

α = [60, 80]
β = [10, 20]
γ = [0, ∞]

α + γ = β

Gerion Entrup MultiSSE – Analysis

11–15
Interrupts and Preemption

- Event-Triggered RTS use interrupts.
- AUTOSAR example: Interrupt triggered alarm.

Diagram:

![Diagram showing artificial interrupt core with tasks and alarm]

- Artificial Interrupt Core
- Alarm 1
- idle
- ActivateTask(T01)
Interrupts and Preemption

- Event-Triggered RTS use interrupts.
- AUTOSAR example: Interrupt triggered alarm.
- MultiSSE has mechanism for Interrupts: Cross-core interleavings
- Model interrupts as syscalls on an extra core.

![Diagram](image-url)
Interrupts and Preemption

- Event-Triggered RTS use interrupts.
- AUTOSAR example: Interrupt triggered alarm.
- MultiSSE has mechanism for Interrupts: Cross-core interleavings
- Model interrupts as syscalls on an extra core.
- Example: WCET of idle models the interarrival time.
Evaluation

- Algorithm tested under four different settings
- Validation and Verification:
  Is the algorithm usable and does it work correctly?

Example application

- Show general working
- MSTG with 137 vertices and 816 edges
- With Timings: 67 vertices and 213 edges
- IPI and Lock S1 superfluous

Trampoline conformance tests

- Open source AUTOSAR implementation
- 12 multicore conformance tests
- Real-world multicore applications

Manually verified as correct
Evaluation

- Algorithm tested under four different settings
- Validation and Verification:
  Is the algorithm usable and does it work correctly?

Example application

- Show general working
- MSTG with 137 vertices and 816 edges
- With Timings: 67 vertices and 213 edges
- IPI and Lock S1 superfluous
Evaluation

- Algorithm tested under four different settings
- Validation and Verification:
  Is the algorithm usable and does it work correctly?

**Example application**

- Show general working
- MSTG with 137 vertices and 816 edges
- With Timings: 67 vertices and 213 edges
- IPI and Lock S1 superfluous

**Trampoline conformance tests**

- Open source AUTOSAR implementation
- 12 multicore conformance tests
- Real-world multicore applications
- Manually verified as correct
## Evaluation

### Synthetic benchmarks

- #cores (2, 4, 6), #threads (up to 15), #spinlocks (1, 3), #lock-users (2 6), #events (0, 5), #cross-core-act. (10%)
- 872 generated applications
- 464 vertices and 2837 edges on average
- Runtime: 23 s to 7825 s (1893 s mean)
**Evaluation**

**Synthetic benchmarks**
- #cores (2, 4, 6), #threads (up to 15), #spinlocks (1, 3), #lock-users (2-6), #events (0-5), #cross-core-act. (10%)
- 872 generated applications
- 464 vertices and 2837 edges on average
- Runtime: 23s to 7825s (1893s mean)

**I4Copter**
- Quadrocopter implementation
- Adapted to AUTOSAR and multicore
  - 2 cores, communication with a lock
- MultiSSE analysis time: 13 seconds
- MSTG with 294 vertices and 2143 edges
- With Timings: 48% less vertices, 70% less edges
MultiSSE: Static Analysis for RTOS Tailoring on Multicore

- Find cross core interleavings just when necessary
- Optionally leverage timing information
- Evaluated with real world applications and synthetic benchmarks

Published as FLOSS:
https://github.com/luhsra/ara
Gerion Entrup, entrup@sra.uni-hannover.de


Evaluation: Timing measurements

<table>
<thead>
<tr>
<th>Core count</th>
<th>Runtime in s × 10^4 without times</th>
<th>Runtime in s × 10^4 with times</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>