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

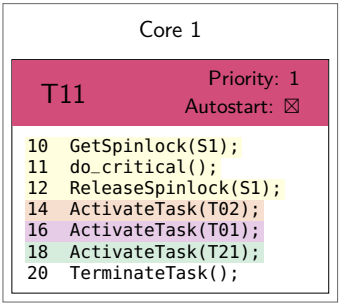
Gerion Entrup, Björn Fiedler, Daniel Lohmann
{entrup, fiedler, lohmann}@sra.uni-hannover.de

Leibniz Universität Hannover

May 12, 2023

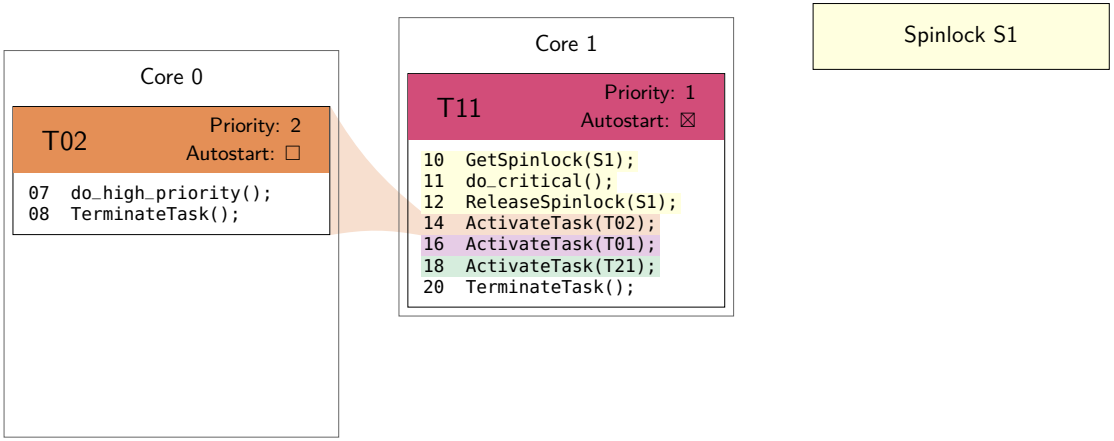
supported by DFG

Optimization: RTOS Tailoring

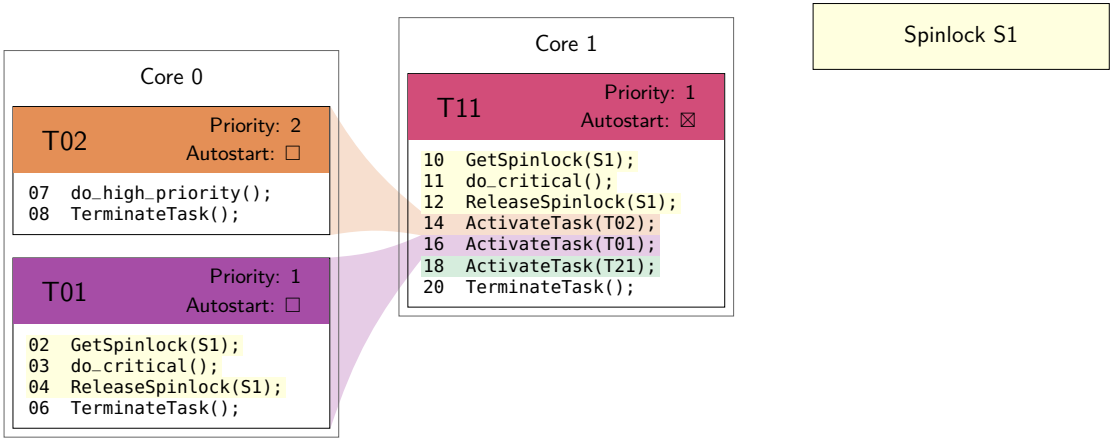


Spinlock S1

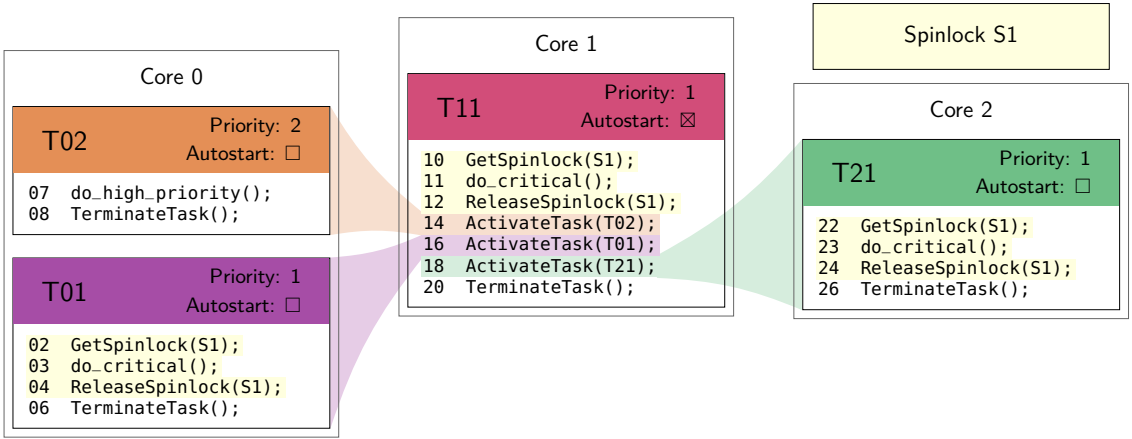
Optimization: RTOS Tailoring



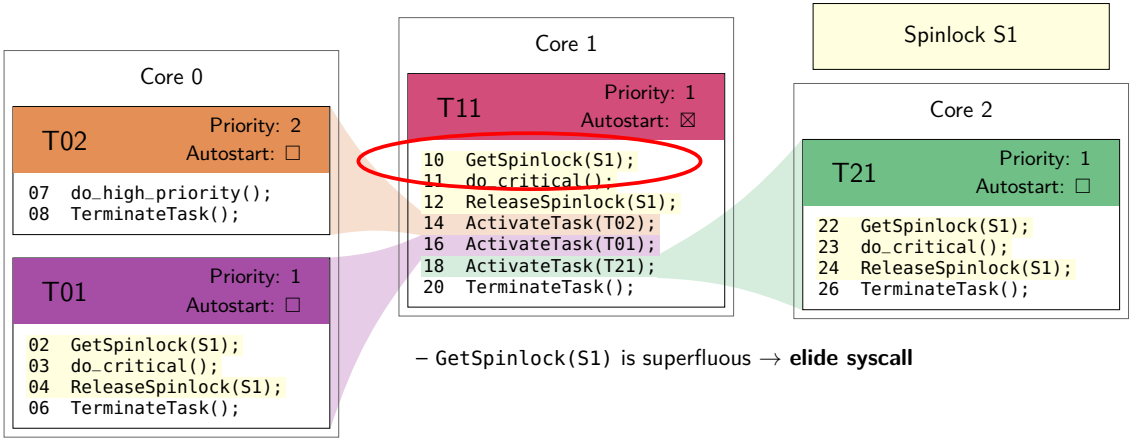
Optimization: RTOS Tailoring



Optimization: RTOS Tailoring

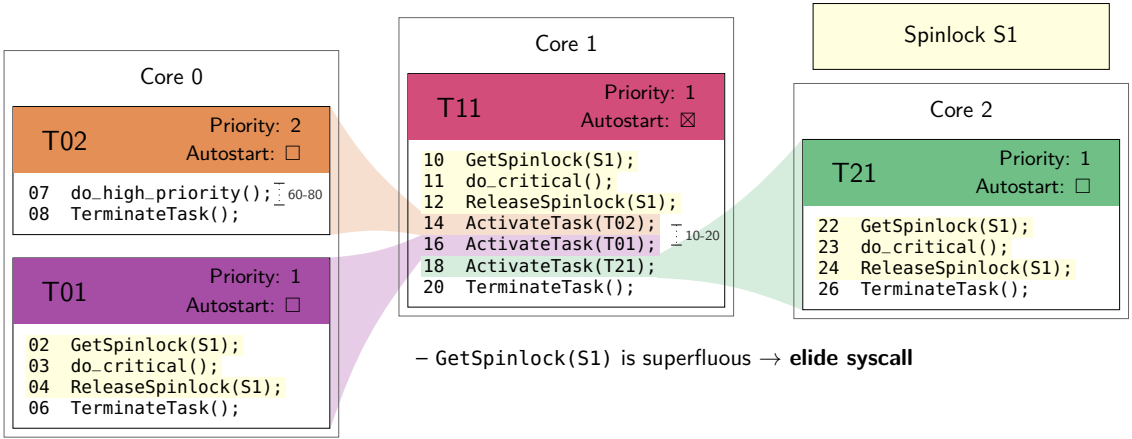


Optimization: RTOS Tailoring



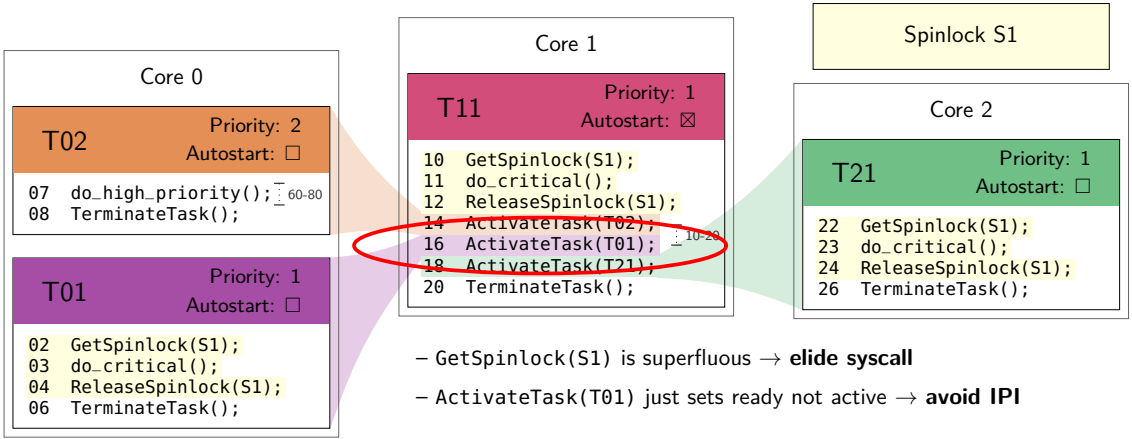
– `GetSpinlock(S1)` is superfluous → **elide syscall**

Optimization: RTOS Tailoring



– `GetSpinlock(S1)` is superfluous → **elide syscall**

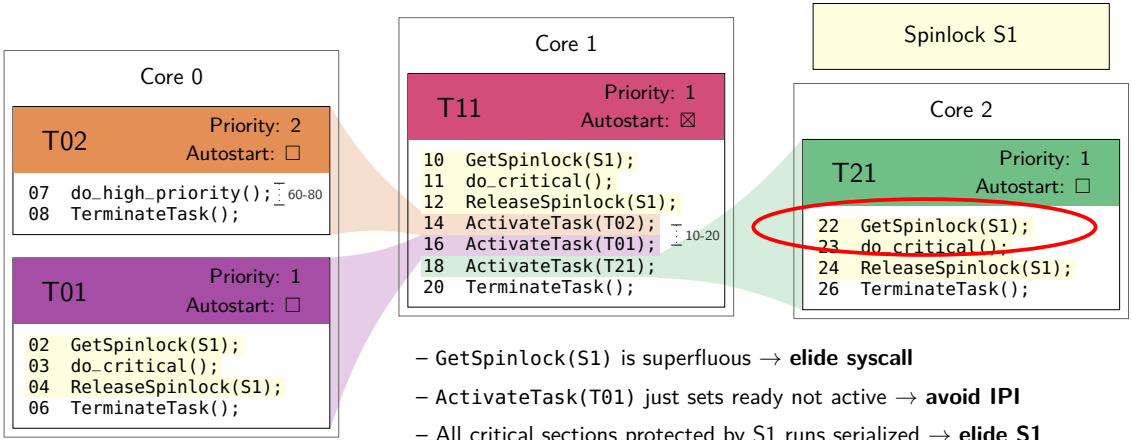
Optimization: RTOS Tailoring



– `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 $\rightarrow$ **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 $\rightarrow$ **combinatoric explosion**

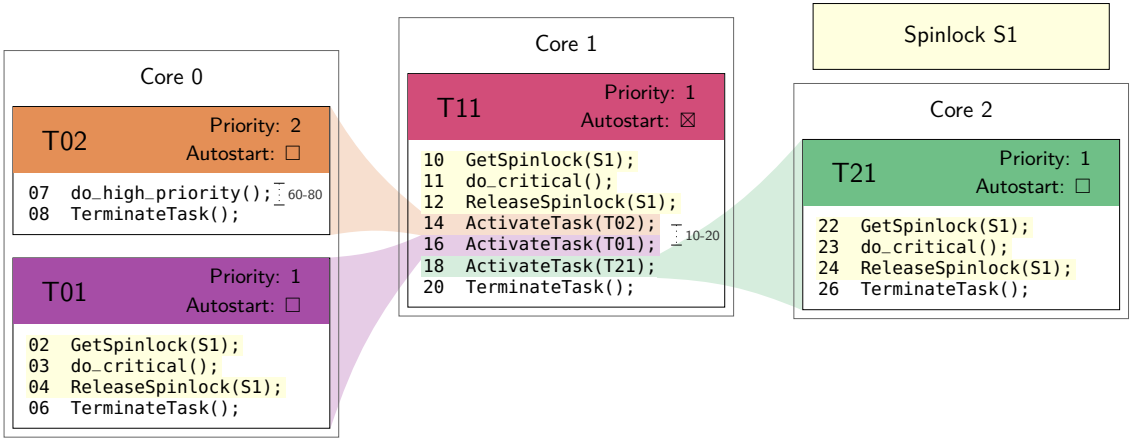
Observation

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

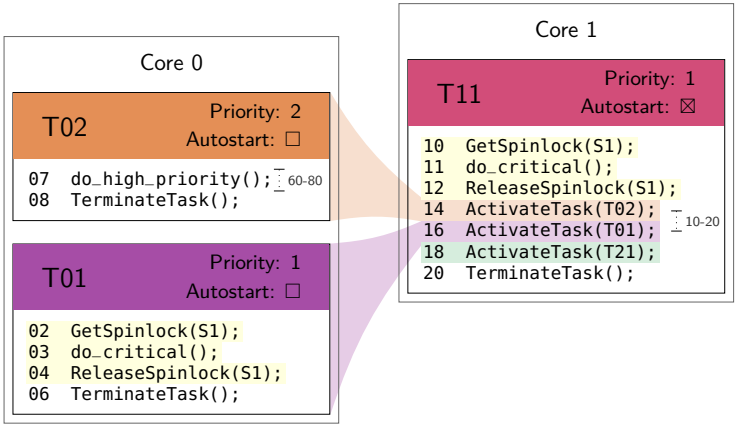
Idea

Calculate cross core interleavings **just** for that specific parts.

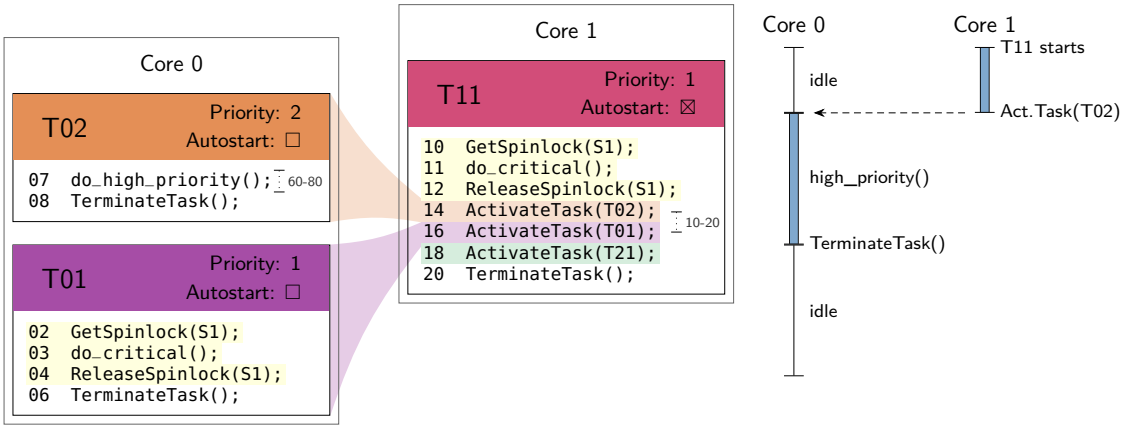
MultiSSE in a nutshell



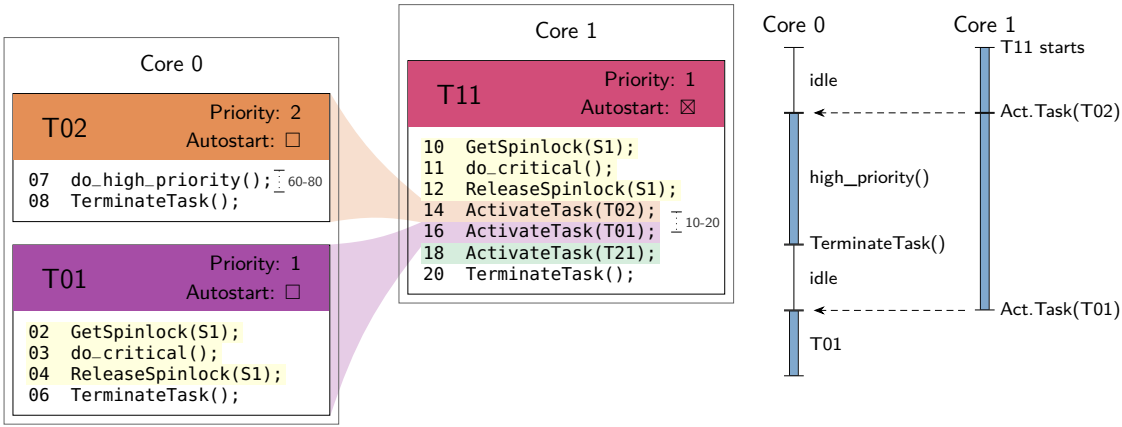
MultiSSE in a nutshell



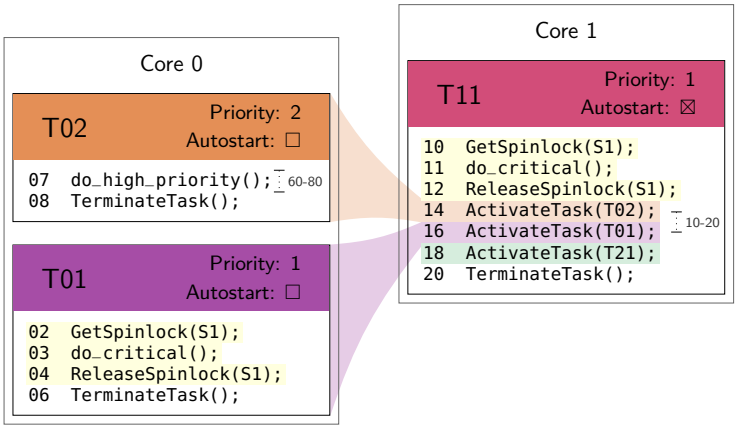
MultiSSE in a nutshell

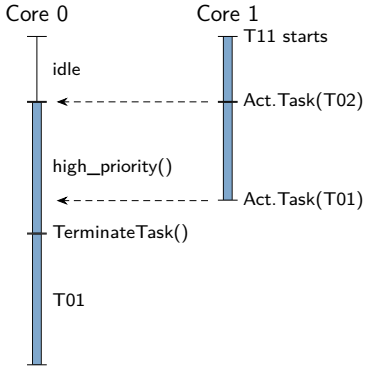


MultiSSE in a nutshell

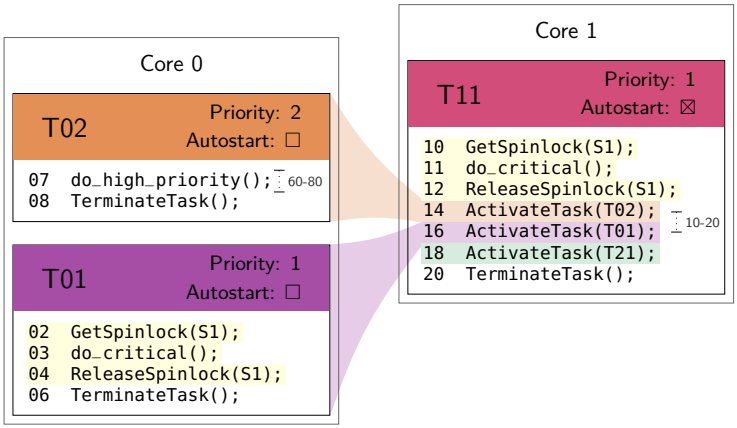


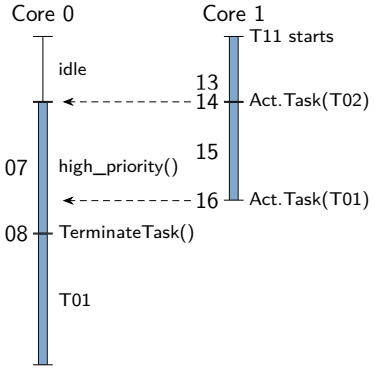
MultiSSE in a nutshell





MultiSSE in a nutshell





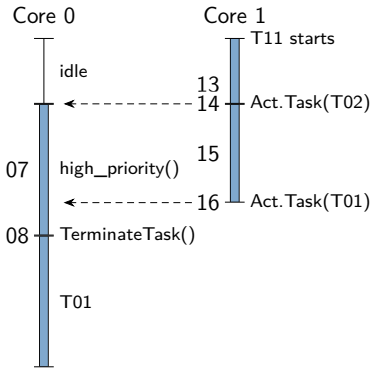
Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]

MultiSSE in a nutshell

idle 13

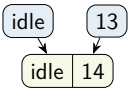
Build the Multi-State Transition Graph (MSTG):

1. Capture core-local effects in **local states**.



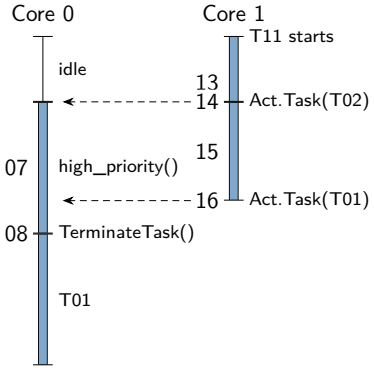
Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]

MultiSSE in a nutshell



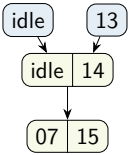
Build the Multi-State Transition Graph (MSTG):

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)**.



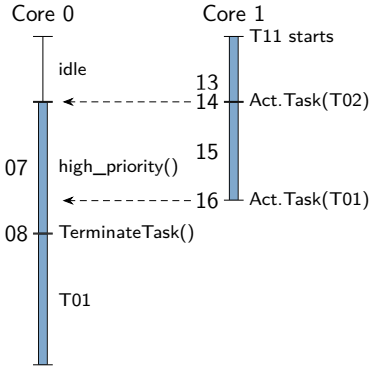
Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]

MultiSSE in a nutshell



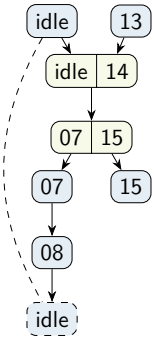
Build the Multi-State Transition Graph (MSTG):

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.



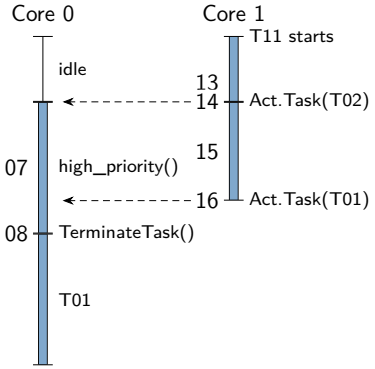
Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]

MultiSSE in a nutshell



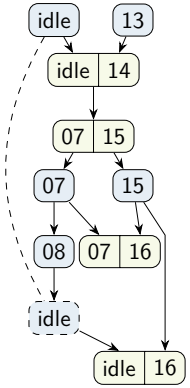
Build the Multi-State Transition Graph (MSTG):

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")



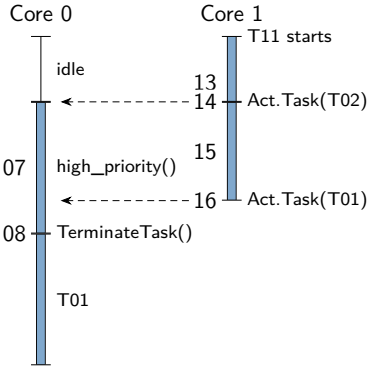
Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]

MultiSSE in a nutshell



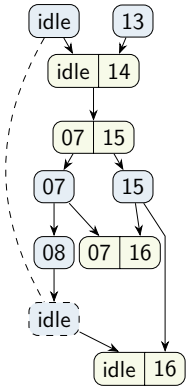
Build the Multi-State Transition Graph (MSTG):

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]

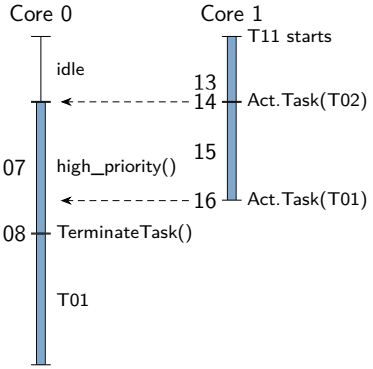
MultiSSE in a nutshell



Build the Multi-State Transition Graph (MSTG):

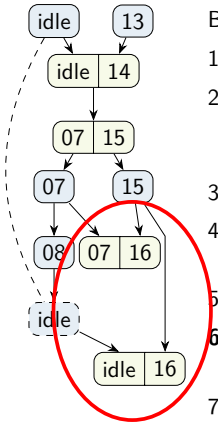
1. Capture core-local effects in **local states**.
2. In case of a cross core system call:
   Build a **multicore state**.
   We call them **synchronization points (SPs)**.
3. Interpret that state.
4. Split the **multicore state** and
   locally interpret the **local states**.
5. Merge duplicated states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

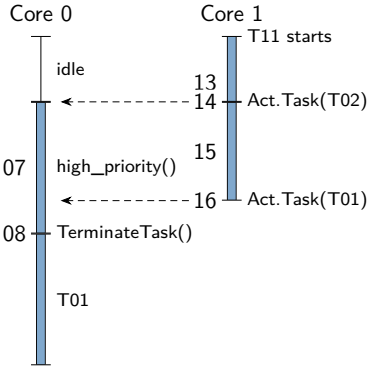
MultiSSE in a nutshell



Build the Multi-State Transition Graph (MSTG):

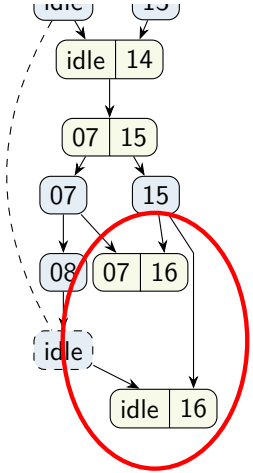
1. Capture core-local effects in **local states**.
2. In case of a cross core system call:
   Build a **multicore state**.
   We call them **synchronization points (SPs)**.
3. Interpret that state.
4. Split the **multicore state** and
   locally interpret the **local states**.
5. Merge duplicated states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



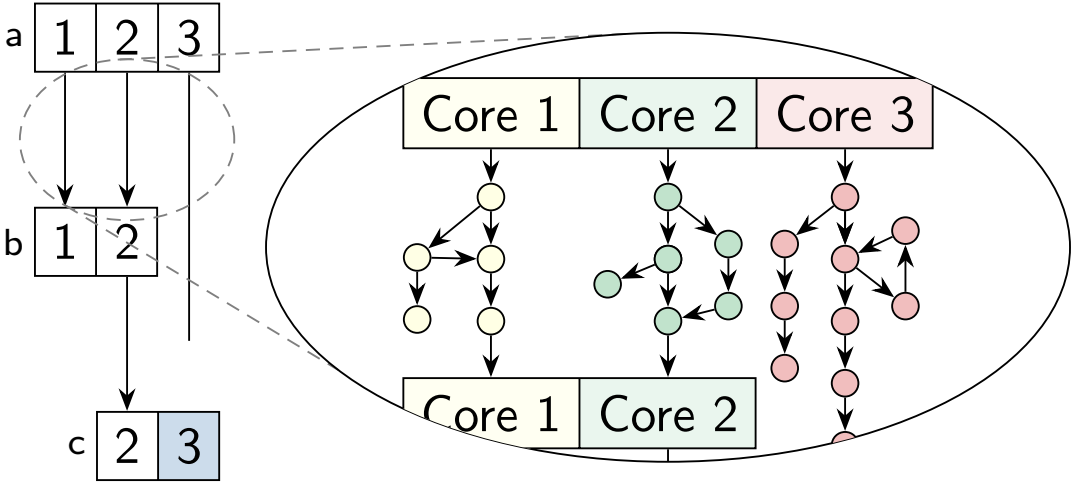
Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]

Pairing Partner Search

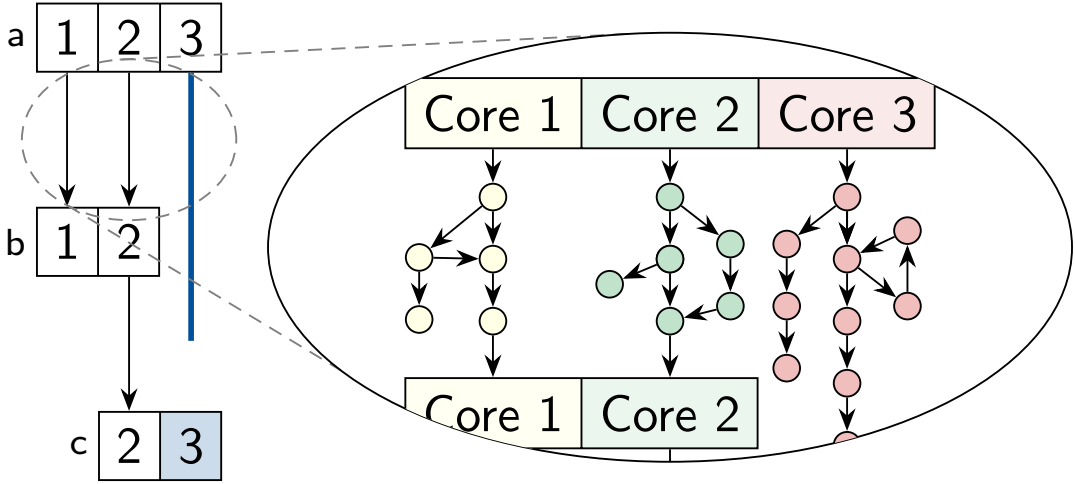


- Pairing Partners of a cross-core system call:
  States on other cores that can execute simultaneously.
- Restrict partner set based on the given control flow.
- Only possible states: **After** the last common SP.

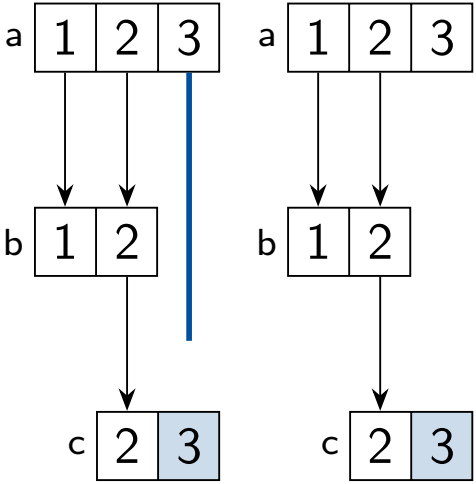
Pairing Partner Search: Visualization



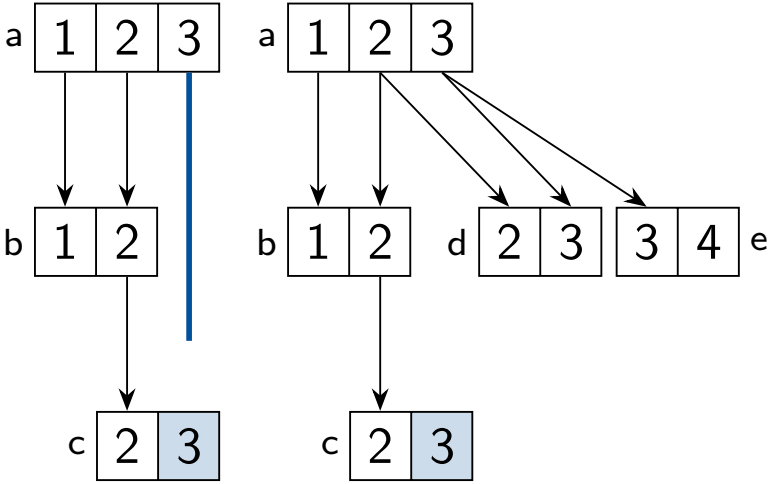
Pairing Partner Search: Visualization



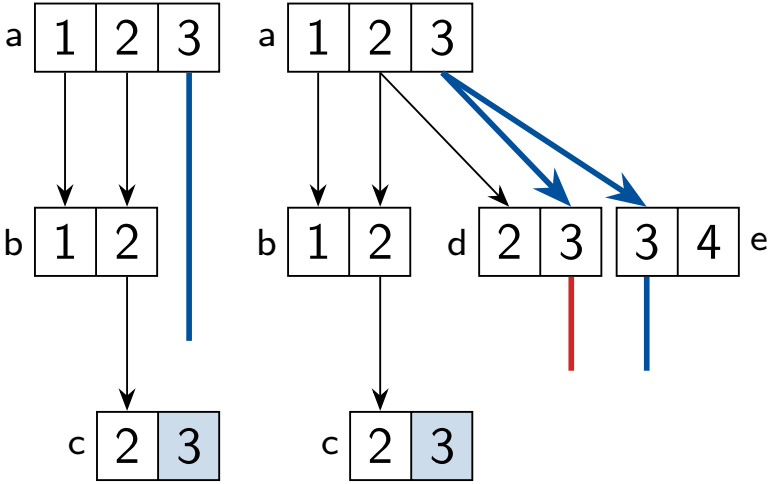
Pairing Partner Search: Visualization



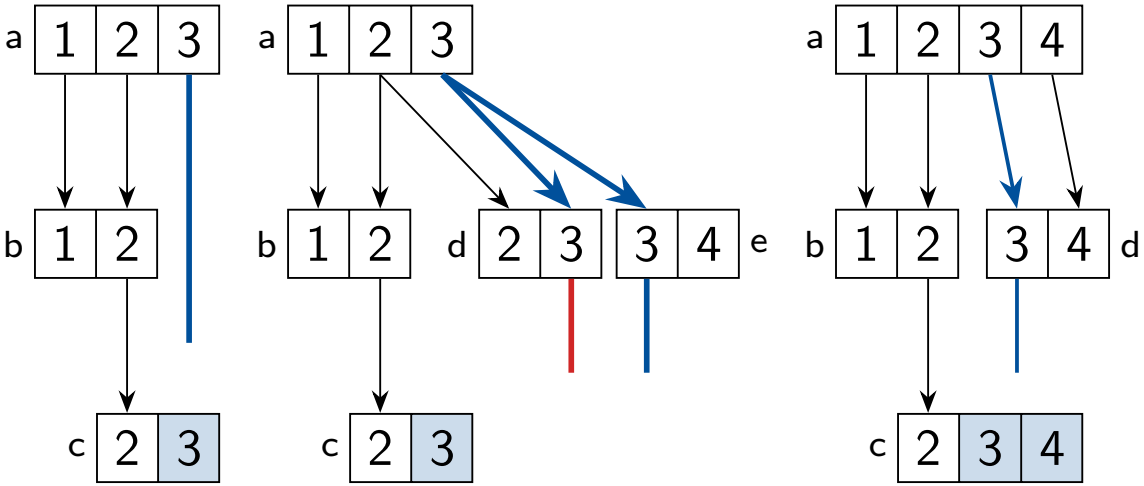
Pairing Partner Search: Visualization



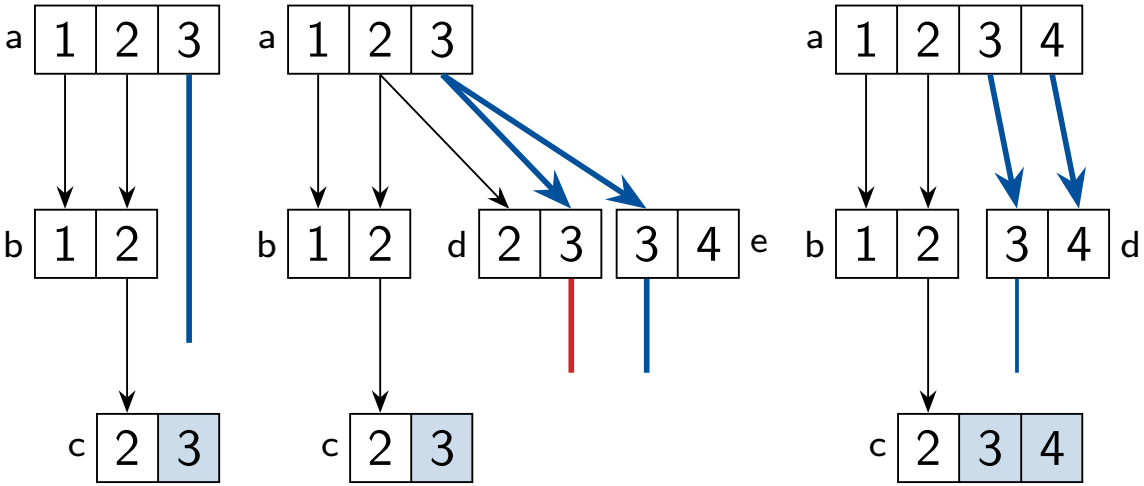
Pairing Partner Search: Visualization



Pairing Partner Search: Visualization



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.

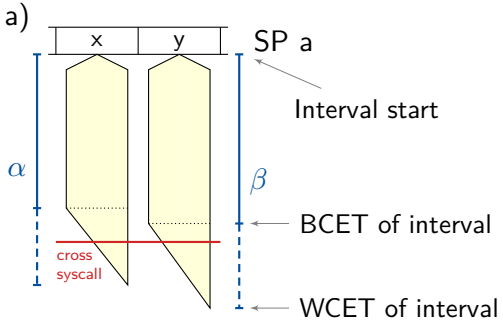
Main Question

Can the affected cores execute two control flow blocks simultaneously?

Solution

Use linear programming (solving of inequality systems).

Timing Information



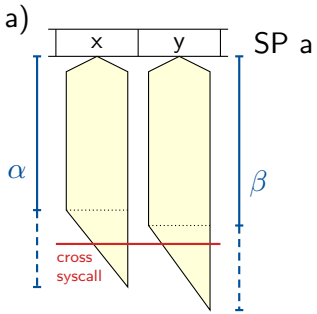
$\alpha = [\alpha_{\text{min}}, \alpha_{\text{max}}]$

$\beta = [0, \beta_{\text{max}}]$

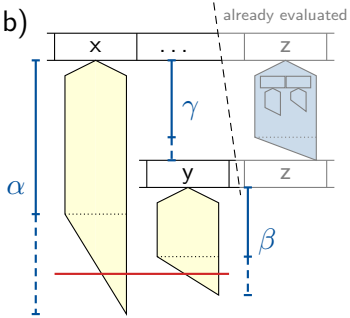
$\alpha = \beta$

- Cross core system call on core "x".
- Question: Can the requested state with interval β happen simultaneously?
- Solution: Yes, if $\alpha = \beta$ has a solution.

Timing Information



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

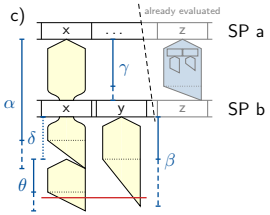


$$\alpha = [\alpha_{\min}, \alpha_{\max}]$$
$$\beta = [0, \beta_{\max}]$$
$$\gamma = [\gamma_{\min}, \gamma_{\max}]$$
$$\alpha = \beta + \gamma$$

- Known intervals between SPs.
- MultiSSE stores solutions.
- Avoid recalculations.

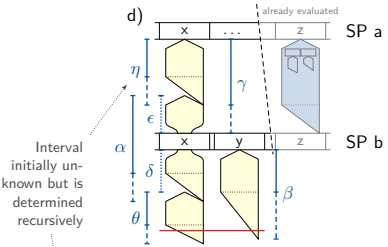
Timing Information – Complex Cases

c)



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

d)



Interval initially unknown but is determined recursively

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

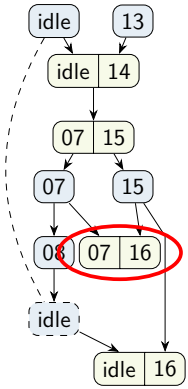
Timing Information – Complex Cases

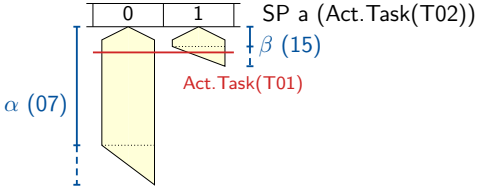
$\alpha = [\alpha_{\min}, \alpha_{\max}]$
$\beta = [0, \beta_{\max}]$
$\gamma = [\gamma_{\min}, \gamma_{\max}]$
$\delta = [0, \alpha_{\max}]$
$\theta = [\theta_{\min}, \theta_{\max}]$
$\alpha = \gamma + \delta$
$\beta = \delta + \theta$

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

Please read the paper for details.

Analysis of the Example With Timing Information



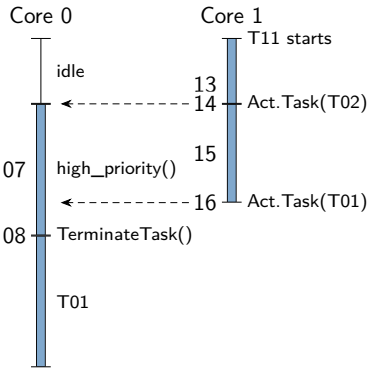


$\alpha = [0, 80]$

$\beta = [10, 20]$

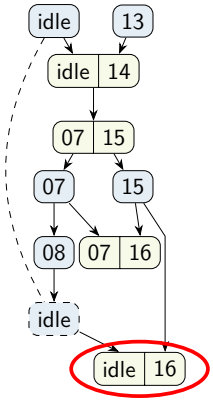
$\alpha = \beta$

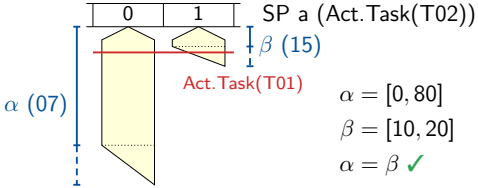
Solution: $\alpha = \beta = [10, 20]$ ✓



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

Analysis of the Example With Timing Information

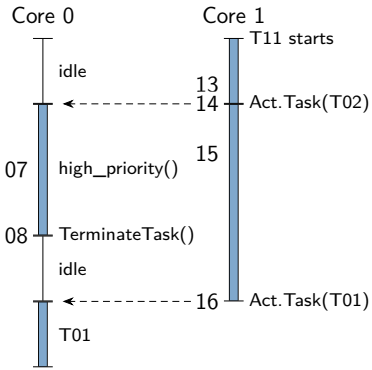




$\alpha = [0, 80]$

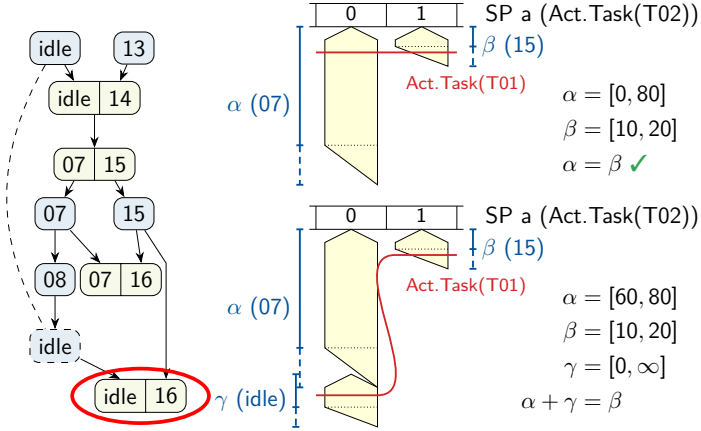
$\beta = [10, 20]$

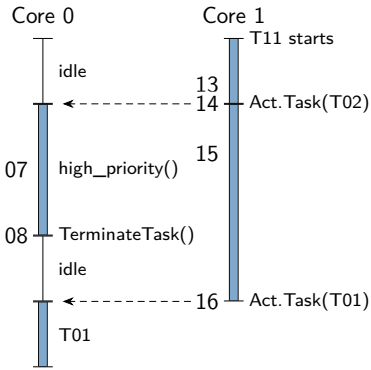
$\alpha = \beta$ ✓



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

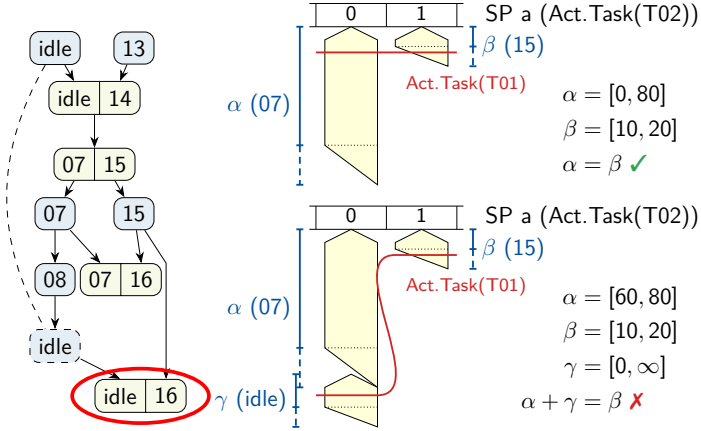
Analysis of the Example With Timing Information

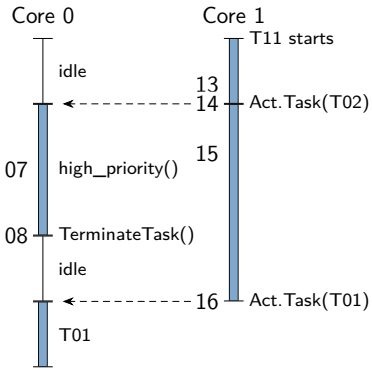




Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]

Analysis of the Example With Timing Information

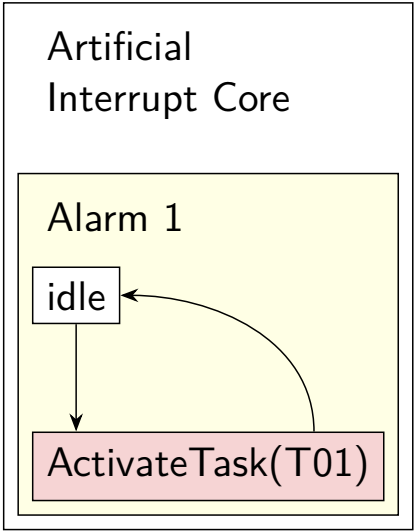




Express control flow with
Atomic Basic Blocks (ABBs) [SCH10]

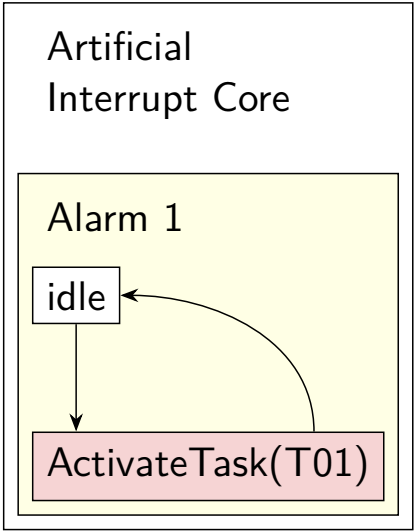
Interrupts and Preemption

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



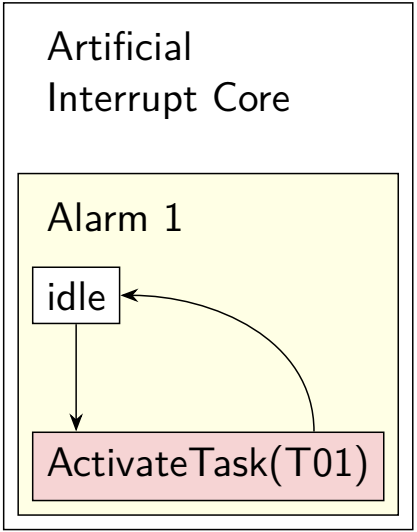
Interrupts and Preemption

- 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.



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?

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

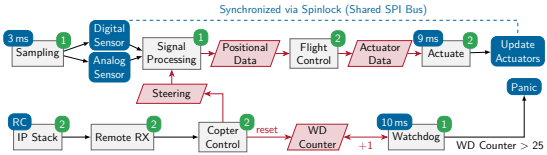
Synthetic benchmarks

- #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: 23 s to 7825 s (1893 s 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

References I

Christian Dietrich, Martin Hoffmann, and Daniel Lohmann. “Cross-Kernel Control-Flow-Graph Analysis for Event-Driven Real-Time Systems”. In: *Proceedings of the 2015 ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '15)* (Portland, Oregon, USA). New York, NY, USA: ACM Press, June 2015. ISBN: 978-1-4503-3257-6. DOI: 10.1145/2670529.2754963.

Björn Fiedler, Gerion Entrup, Christian Dietrich, and Daniel Lohmann. “ARA: Static Initialization of Dynamically-Created System Objects”. In: *Proceedings of the 27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'21)* (Virtual Event). May 2021, pp. 400–412. DOI: 10.1109/RTAS52030.2021.00039.

Fabian Scheler and Wolfgang Schröder-Preikschat. “The RTSC: Leveraging the Migration from Event-Triggered to Time-Triggered Systems”. In: *Proceedings of the 13th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '10)* (Carmona, Spain). Washington, DC, USA: IEEE Computer Society Press, May 2010, pp. 34–41. ISBN: 978-0-7695-4037-5. DOI: 10.1109/ISORC.2010.11.

Evaluation: Timing measurements