Boosting Job-Level Migration by Static Analysis

Workshop on Operating Systems Platforms for Embedded Real-Time Applications
July 09, 2019

Tobias Klaus, Peter Ulbrich, Phillip Raffeck, Benjamin Frank,
Lisa Wernet, Maxim Ritter von Onciul, Wolfgang Schröder-Preikschat

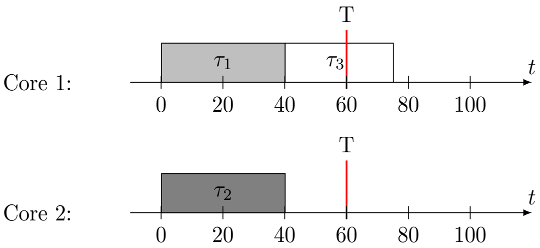
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

SCHR 603/9-2
SCHR 603/13-1
SCHR 603/14-2
CRC/TRR 89 Project C1

EU EFRE funds
0704/883 25

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

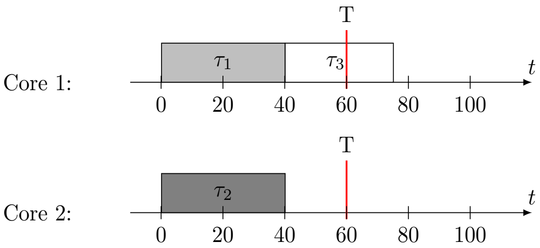
Multi-Core Scheduling



Multi-Core Systems

- Static allocation of tasks to cores

Multi-Core Scheduling

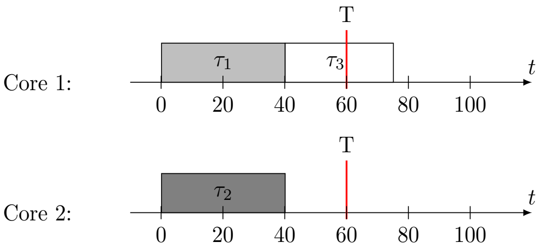


Multi-Core Systems

- Static allocation of tasks to cores

→ **Poor utilization and schedulability**

Multi-Core Scheduling



Multi-Core Systems

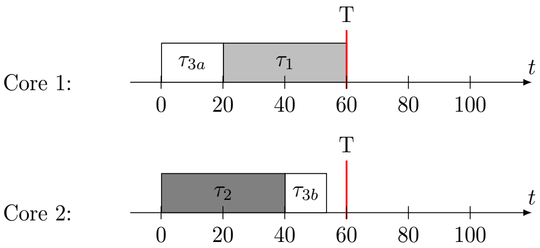
- Static allocation of tasks to cores

→ **Poor utilization and schedulability**

Solution: Full Migration

- Dynamic (re)allocation of tasks
- Good utilization and schedulability

Multi-Core Scheduling



Multi-Core Systems

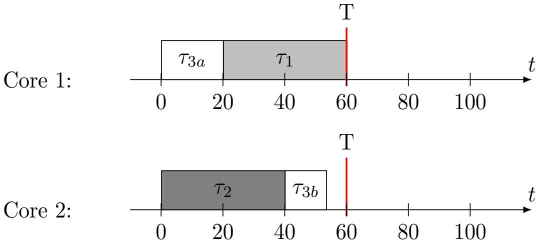
- Static allocation of tasks to cores

→ **Poor utilization and schedulability**

Solution: Full Migration?

- Dynamic (re)allocation of tasks
- Good utilization and schedulability

Multi-Core Scheduling



Multi-Core Systems

- Static allocation of tasks to cores

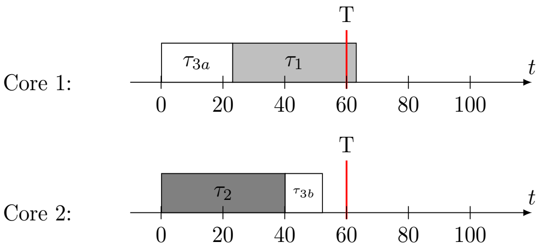
→ **Poor utilization and schedulability**

Solution: Full Migration?

- Dynamic (re)allocation of tasks
- Good utilization and schedulability

→ **Impractical in real-time systems**

Multi-Core Scheduling



Multi-Core Systems

- Static allocation of tasks to cores

→ **Poor utilization and schedulability**

Solution: Full Migration?

- Dynamic (re)allocation of tasks
- Good utilization and schedulability

→ **Impractical in real-time systems**

Static Allocation Again?

- Split tasks to appropriate size

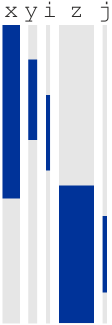
Splitting the Execution


```
int32_t x = 0;
uint16_t y = foo();
for (uint8_t i = 0; i < 5; i++) {
  x += y * bar[i];
}
int64_t z = x * 4711;
for (uint8_t j = 0; j < 5; j++) {
  z += baz[j];
}
return z;
```

Find Appropriate Split Points

```
int32_t x = 0;
uint16_t y = foo();
for (uint8_t i = 0; i < 5; i++) {
  x += y * bar[i];
}
int64_t z = x * 4711;
for (uint8_t j = 0; j < 5; j++) {
  z += baz[j];
}
return z;
```

Lifespan:

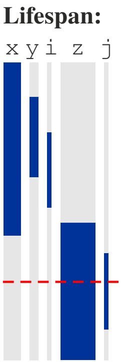


Find Appropriate Split Points

- Static analysis

Splitting the Execution


```
int32_t x = 0;
uint16_t y = foo();
for (uint8_t i = 0; i < 5; i++) {
  x += y * bar[i];
}
int64_t z = x * 4711;
for (uint8_t j = 0; j < 5; j++) {
  z += baz[j];
}
return z;
```

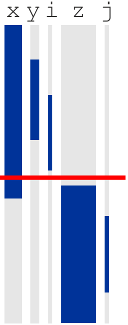


Find Appropriate Split Points

- Static analysis
- Consider WCET

```
int32_t x = 0;
uint16_t y = foo();
for (uint8_t i = 0; i < 5; i++) {
  x += y * bar[i];
}
int64_t z = x * 4711;
for (uint8_t j = 0; j < 5; j++) {
  z += baz[j];
}
return z;
```

Lifespan:



Find Appropriate Split Points

- Static analysis
- Consider WCET
- Minimize migration cost

Migration

Challenges

- Split tasks to target WCET

Migration

Challenges

- Split tasks to target WCET
- Reduce migration cost

Migration

Challenges

- Split tasks to target WCET
- Reduce migration cost

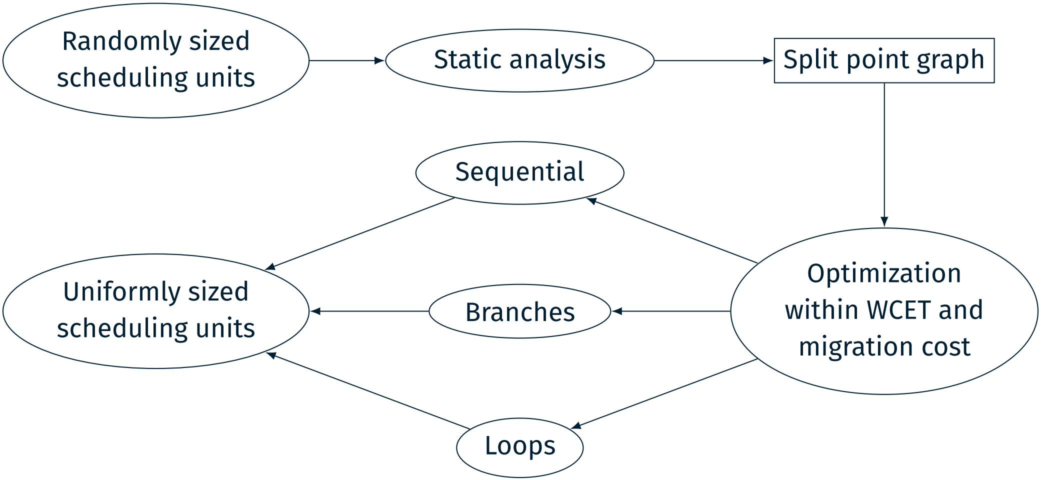
Approach

→ **Job-Level Migration**

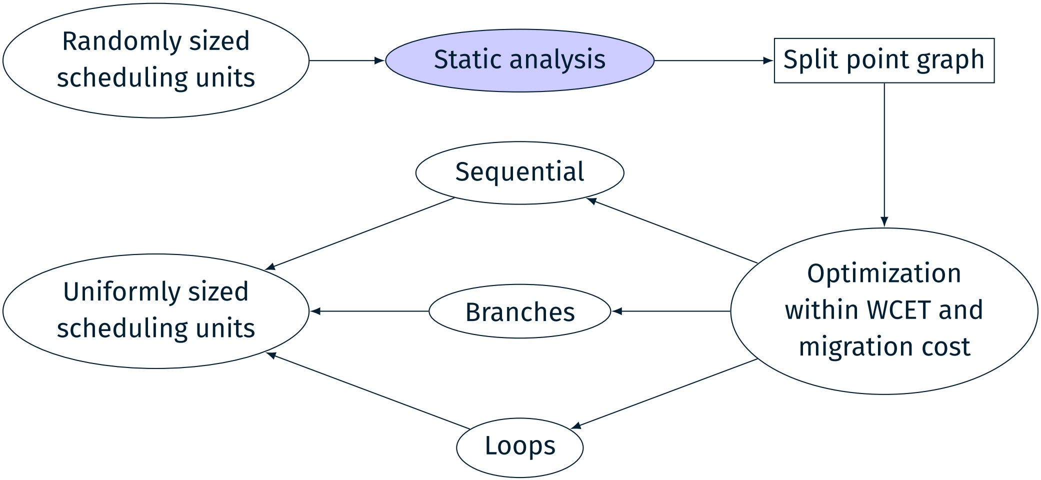
→ **Static Analysis**

→ **Optimization within two dimensions**

Overview

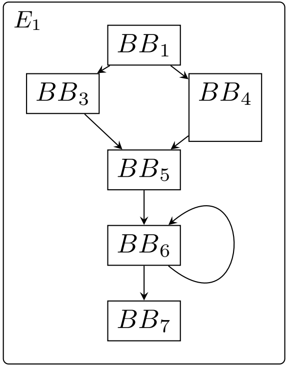

Randomly sized scheduling units
Static analysis
Split point graph
Sequential
Uniformly sized scheduling units
Branches
Optimization within WCET and migration cost
Loops

Overview



Randomly sized scheduling units
Static analysis
Split point graph
Sequential
Uniformly sized scheduling units
Branches
Optimization within WCET and migration cost
Loops

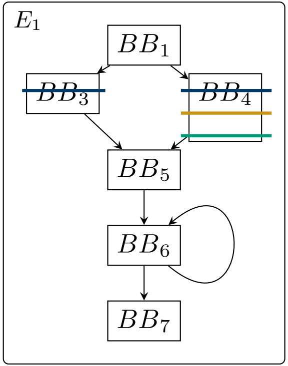
Static analysis



Basic Procedure

1. Create control-flow graph
2. WCET analysis
3. Lifespan analysis

Static analysis

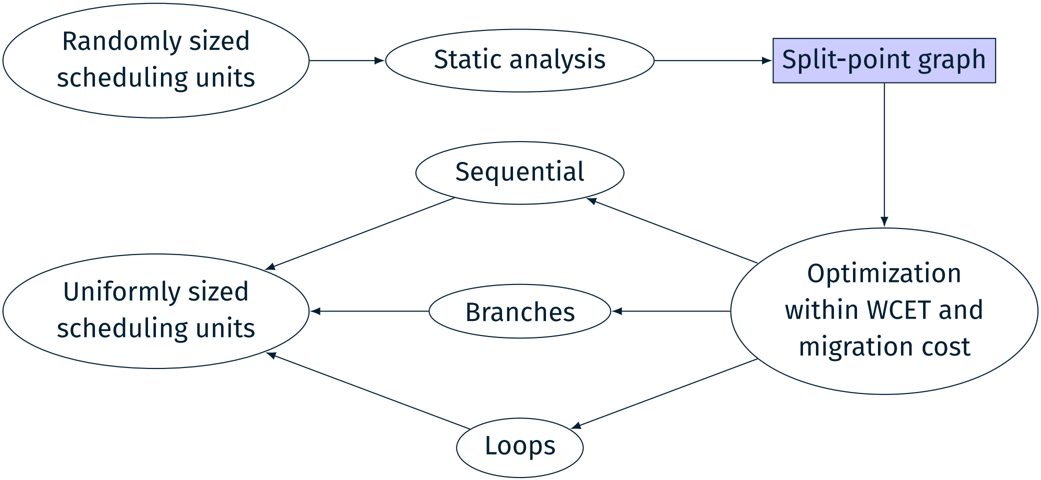


Basic Procedure

1. Create control-flow graph
2. WCET analysis
3. Lifespan analysis

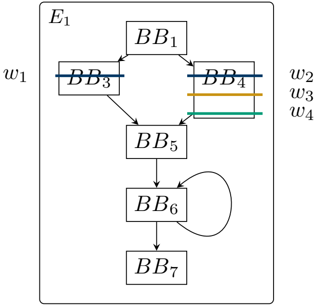
} Split-point candidates

Split-Point Graphs



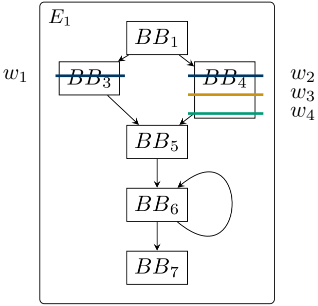
General Concept: Split-Point Graphs

Control-Flow Graph

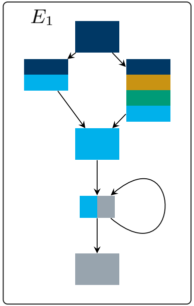


General Concept: Split-Point Graphs

Control-Flow Graph

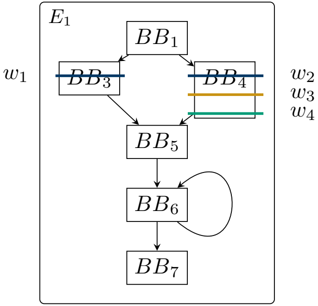


Intermediate Graph

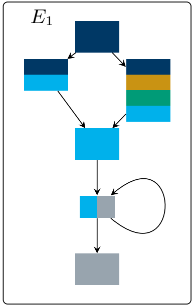


General Concept: Split-Point Graphs

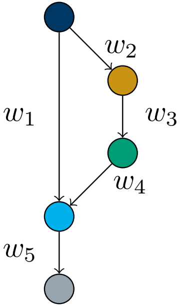
Control-Flow Graph



Intermediate Graph

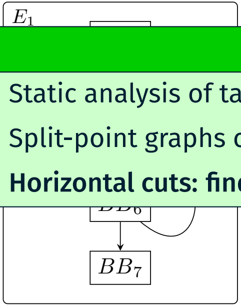


Split-Point Graph

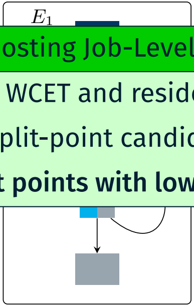


General Concept: Split-Point Graphs

Control-Flow Graph



Intermediate Graph

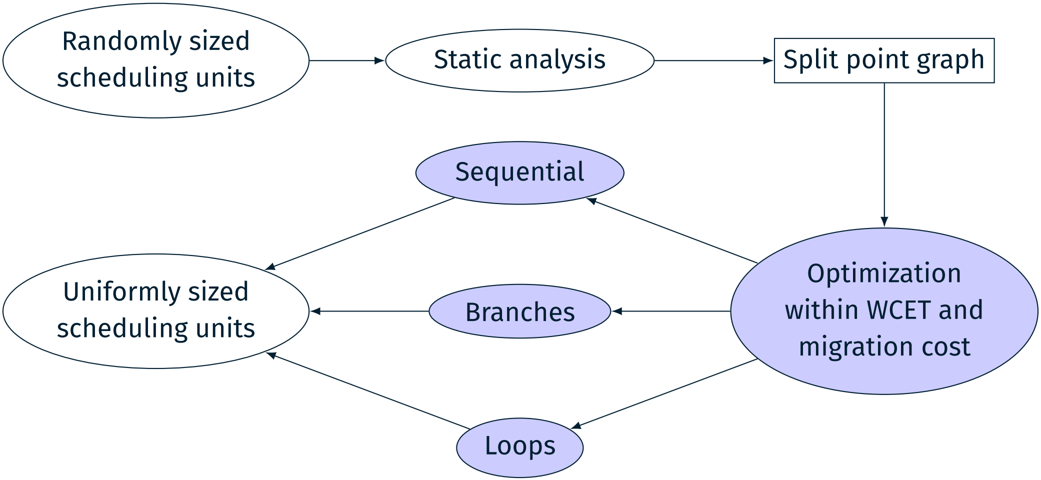


Split-Point Graph

Boosting Job-Level Migration

- Static analysis of tasks w.r.t. WCET and resident-set size
- Split-point graphs capture split-point candidates
- **Horizontal cuts: finding split points with low migration cost**

Overview



Randomly sized scheduling units
Static analysis
Split point graph
Sequential
Uniformly sized scheduling units
Branches
Optimization within WCET and migration cost
Loops

Original Loop

```
LOOP_Bound(x:10);
for(int i = 0; i < x; ++i)
{ .... }
```

- Splitting the loop body?
- # of iterations dominates WCET

Splitting Loops

Let the body untouched!

Original Loop

```
LOOP_Bound(x:10);
for(int i = 0; i < x; ++i)
{ .... }
```

- Splitting the loop body?
- # of iterations dominates WCET

→ **Split by number of iterations!**

Splitting Loops

Original Loop

```
LOOP_Bound(x:10);
for(int i = 0; i < x; ++i)
{ .... }
```

- Splitting the loop body?
- # of iterations dominates WCET

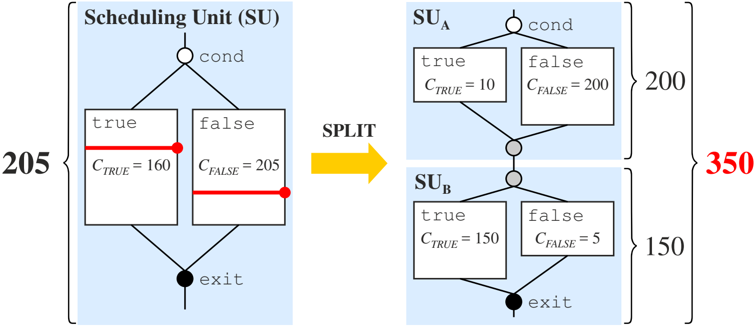
→ **Split by number of iterations!**

Loop after Splitting

```
int i = 0, C = 5;
for(; i < x && C; ++i)
{ --C; .... }
 ....
 C = 5;
for(; i < x && C; ++i)
{ --C; .... }
```

General Approach

- Compute number of iterations to fit target WCET
- Derive upper bound for the number of cuts
- Duplicate body and adjust loop condition

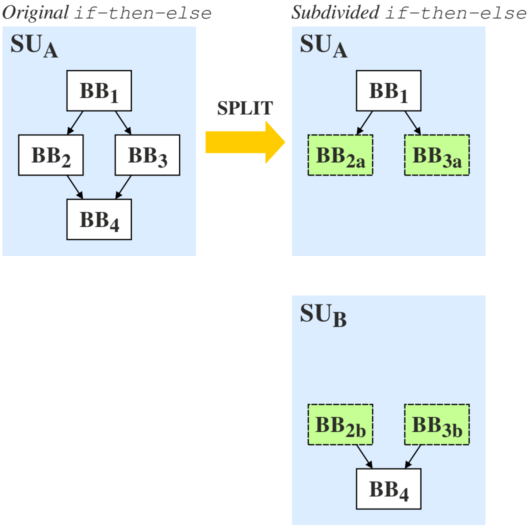


Additional Pessimism Caused by Naive Splitting

- Local optimization may lead to unbalanced cuts in branches
- Condition is unknown at compile time

→ **Overapproximation in timing analysis**

Splitting Branches

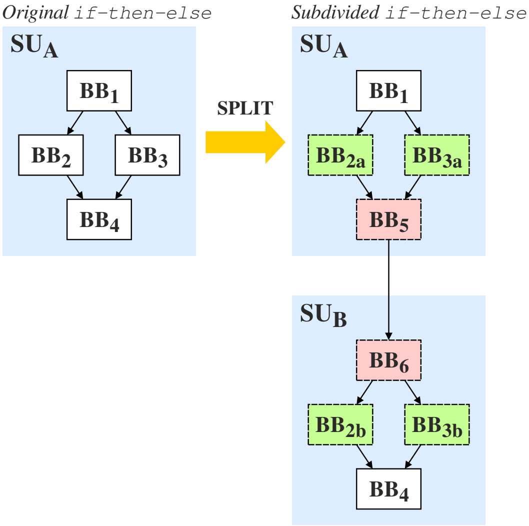


Global vs. Local Optimization

- Find suitable points locally
- Global alignment between branches

→ **Minimize size differences**

Splitting Branches



Global vs. Local Optimization

- Find suitable points locally
- Global alignment between branches

→ **Minimize size differences**

General Approach

- Add jump
- Additional logic

Overheads per Cut

Sequential Code

$$i^{+}_{seq} = 1$$

Overheads per Cut

Sequential Code

$$i^{+}_{seq} = 1$$

Branches

$$\begin{aligned} i^{+}_{if} = \; & n_{branch} * 2 && \text{Marking the active branch} \\ & + 1 && \text{Terminating the first scheduling unit} \\ & + 3 && \text{Proceeding with the correct branch} \end{aligned}$$

Overheads per Cut

Sequential Code

$$i^+_{seq} = 1$$

Branches

$i^+_{if} = n_{branch} * 2$	Marking the active branch
$+ 1$	Terminating the first scheduling unit
$+ 3$	Proceeding with the correct branch

Loops

$i^+_{loop} = (5 + 1)$	Counter for planned iterations
$+ 2$	Exiting the scheduling unit and resetting the iteration counter
$+ 3$	Executing the following part of the loop

i^+ # additional instructions

n_{branch} # branches, affected by a horizontal cut

Sequential Code

$$i^{+}_{seq} = 1$$

Branches

$$i^{+}_{if} = n_{branch} * 2$$ Marking the active branch

$$+ 1$$ Terminating the first scheduling unit

$$+ 3$$

Loops

$$i^{+}_{loop} = (5 + 1)$$

$$+ 2$$ …teration counter

$$+ 3$$ Executing the following part of the loop

Low overall overhead

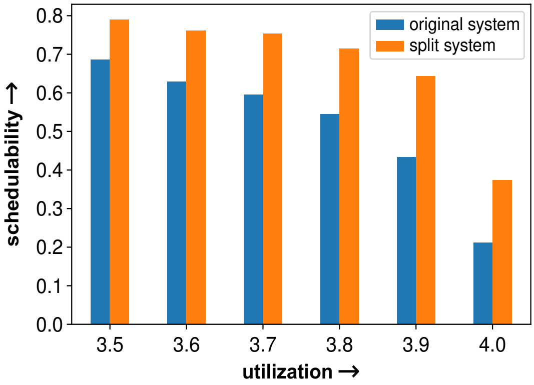
- Only few additional instructions for all different program constructs

⇒ Minor effects on overall execution time

i^{+} # additional instructions

n_{branch} # branches, affected by a horizontal cut

Schedulability



Effects on the schedulability of systems with high utilization

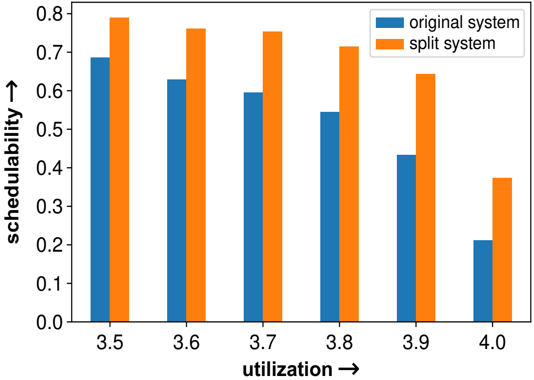
Experimental Setup

- System with four processor cores
- 12000 synthetic benchmark systems

Goal

- Feasible allocation and schedule for each task set

Schedulability



Effects on the schedulability of systems with high utilization

Experimental Setup

- System with four processor cores
- 12000 synthetic benchmark systems

Goal

- Feasible allocation and schedule for each task set

⇒ **70 percent more schedulable task sets for the highest utilization**

Migration Costs

Finding split points with low migration cost

Experimental Setup

- Real-world benchmarks taken from the TACLeBench suite
- Creation of OSEK systems: one benchmark task and two load tasks
  - Generate systems which are unschedulable on two cores without migration
  - Only cut benchmark tasks
- Recording of the resident-set size (in LLVM-IR types)
  - Worst-case migration cost observed in all possible split-point candidates
  - Migration cost of the split point chosen by our approach

Migration Costs

Benchmark	Worst-case Resident-set Size [bits]	Split-point Resident-set Size [bits]	Cost improvement [bits]
binarysearch	225	224	1
bitonic	65	64	1
complex_update	480	288	192
countnegative	2176	1568	608
filterbank	60 736	60 704	32
iir	432	400	32
insertsort	544	128	416
minver	17 568	16 800	768
petrinet	5057	5056	1

Migration Costs

Benchmark	Worst-case Resident-set Size [bits]	Split-point Resident-set Size [bits]	Cost improvement [bits]
binarysearch	225	224	1
bitonic	65	64	1
complex_update	480	288	192
countnegative	2176	1568	608
filterbank	60 736	60 704	32
iir	432	400	32
insertsort	544	128	416
minver	17 568	16 800	768
petrinet	5057	5056	1

⇒ **Lower worst-case migration overhead**

⇒ **Tighter results from timing analysis**

Conclusion and Outlook

Conclusion

- Compile time
  - Beneficial size of scheduling units

$\Rightarrow$ **Systems with high utilization become schedulable**

Conclusion and Outlook

Conclusion

- Compile time
  - Beneficial size of scheduling units

  ⇒ **Systems with high utilization become schedulable**
- Runtime
  - Migration at beneficial points
  - Only if necessary

  ⇒ **Reducing overapproximation in the WCET analysis**

Conclusion and Outlook

Conclusion

- Compile time
  - Beneficial size of scheduling units

  $\Rightarrow$ **Systems with high utilization become schedulable**
- Runtime
  - Migration at beneficial points
  - Only if necessary

  $\Rightarrow$ **Reducing overapproximation in the WCET analysis**

Current Work and Outlook

- More accurate WCET estimation
- Adapt an OS to support migration threshold
- Consider the OS and system calls within the analysis