Bus-centric Optimization and Analysis for Multicore Hard Real-Time Systems

Dominic Oehlert    Heiko Falk

Institute of Embedded Systems
Hamburg University of Technology

firstname.surname@tuhh.de

11.06.2018
Outline

**Bus-aware Static Instruction SPM Allocation**
- Evaluation
- Conclusion

**Compiler-based Event Arrival Function Extraction**
- Extraction
- Outlook
Static Instruction SPM Allocation

- The Worst-Case Execution Time (WCET) of a program is crucial in hard real-time systems
- Scratchpad Memories (SPMs) offer great opportunities to reduce the WCET
- ILP-based optimizations can be used

⇒ SPM allocation involves many challenges when actually applied
  - Instructions added, referenced blocks, asymmetric jump costs, ...
  - Increased complexity for multicore systems
Static Instruction SPM Allocation

- The Worst-Case Execution Time (WCET) of a program is crucial in hard real-time systems

- Scratchpad Memories (SPMs) offer great opportunities to reduce the WCET

- ILP-based optimizations can be used

⇒ SPM allocation involves many challenges when actually applied
  - Instructions added, referenced blocks, asymmetric jump costs, ...
  - Increased complexity for multicore systems
Static Instruction SPM Allocation

- The Worst-Case Execution Time (WCET) of a program is crucial in hard real-time systems.

- Scratchpad Memories (SPMs) offer great opportunities to reduce the WCET.

- ILP-based optimizations can be used.

⇒ SPM allocation involves many challenges when actually applied:
  - Instructions added, referenced blocks, asymmetric jump costs,
  - Increased complexity for multicore systems.
Static Instruction SPM Allocation

- The Worst-Case Execution Time (WCET) of a program is crucial in hard real-time systems.

- Scratchpad Memories (SPMs) offer great opportunities to reduce the WCET.

- ILP-based optimizations can be used.

⇒ SPM allocation involves many challenges when actually applied:
   - Instructions added, referenced blocks, asymmetric jump costs, ...
   - Increased complexity for multicore systems.
Static Instruction SPM Allocation

- The Worst-Case Execution Time (WCET) of a program is crucial in hard real-time systems.
- Scratchpad Memories (SPMs) offer great opportunities to reduce the WCET.
- ILP-based optimizations can be used.

⇒ SPM allocation involves many challenges when actually applied:
  - Instructions added, referenced blocks, asymmetric jump costs, ...
  - Increased complexity for multicore systems.
Multicore Architecture

- \( N_c \) homogeneous parallel cores
- Private instruction scratchpad memories
- Shared Flash memory
- TDMA scheduled bus
Applying Singlecore SPM Allocation

```
orr  r3, r2, LSL #2
sub  r3, r3, #4
ldr  r1, [r3]
add  r1, r1, #1024
```

```
add  r0, r1, #5
eor  r0, r6, r0
beq B
```

```
t F
A
B
C
```
Applying Singlecore SPM Allocation

- 4 cores
- Private SPM size: 20 B
Applying Singlecore SPM Allocation

- 4 cores
- Private SPM size: 20 B

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>WCET (Cycles)</th>
<th>Size (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash</td>
<td>SPM</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>390</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>114</td>
<td>9</td>
</tr>
</tbody>
</table>
Applying Singlecore SPM Allocation

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>WCET (Cycles)</th>
<th>Size (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flash</td>
<td>SPM</td>
</tr>
<tr>
<td>A</td>
<td>411</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>114</td>
<td>9</td>
</tr>
</tbody>
</table>

- 4 cores
- Private SPM size: 20 B

⇒ Expected WCET Reduction: 84 Cycles (↓ 16%)
Applying Singlecore SPM Allocation

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>WCET (Cycles)</th>
<th>Size (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash</td>
<td>SPM</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>411</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>114</td>
<td>9</td>
</tr>
</tbody>
</table>

- 4 cores
- Private SPM size: 20 B

⇒ Expected WCET Reduction: 84 Cycles (↓ 16%)
⇒ Actual WCET Reduction: -7 Cycles (↑ 1%)
Cause of the False Estimation

All in Flash
Cause of the False Estimation

All in Flash

All in SPM
Cause of the False Estimation

All in Flash

All in SPM

C allocated to SPM
Cause of the False Estimation

```
sub r3, r3, #4
ldr r1, [r3]
add r1, r1, #1024
```

All in Flash

All in SPM

C allocated to SPM
Preliminaries (Slot Length)

- WCETs of basic blocks located in the shared Flash memory depend on the *ingoing* bus offset
  - Analyzed WCETs are not ensured to be safe anymore

⇒ Assume all TDMA slots equally-sized and fixed to the length of a Flash access
  - Analyzed WCETs are safe again (per BB in shared memory)
Preliminaries (Slot Length)

• WCETs of basic blocks located in the shared Flash memory depend on the *ingoing* bus offset

• Analyzed WCETs are not ensured to be safe anymore

⇒ Assume all TDMA slots equally-sized and fixed to the length of a Flash access
  • Analyzed WCETs are safe again (per BB in shared memory)
Preliminaries (Slot Length)

- WCETs of basic blocks located in the shared Flash memory depend on the \textit{ingoing} bus offset

- Analyzed WCETs are not ensured to be safe anymore

\[\Rightarrow\] Assume all TDMA slots equally-sized and fixed to the length of a Flash access

- Analyzed WCETs are safe again (per BB in shared memory)
Preliminaries (Slot Length)

- WCETs of basic blocks located in the shared Flash memory depend on the *ingoing* bus offset

- Analyzed WCETs are not ensured to be safe anymore

⇒ Assume all TDMA slots equally-sized and fixed to the length of a Flash access
  - Analyzed WCETs are safe again (per BB in shared memory)
Preliminaries (Sub Basic Blocks)

- Instructions located in the private SPM may access the shared memory

```
add r0, r1, #5
eor r0, r6, r0
sub r6, r6, #7
...  
ldr r3, [r9]
...  
or r1, r0, r2
cmp r1, r3
bne Y
```
Preliminaries (Sub Basic Blocks)

- Instructions located in the private SPM may access the shared memory.
- A basic block’s WCET may depend on the bus schedule, despite residing in the private SPM.
Preliminaries (Sub Basic Blocks)

- Instructions located in the private SPM may access the shared memory.

- A basic block’s WCET may depend on the bus schedule, despite residing in the private SPM.

⇒ Assume sub basic blocks, containing at maximum one such potential instruction.

```
add r0, r1, #5
eor r0, r6, r0
sub r6, r6, #7
    ...
ldr r3, [r9]
    ...
orr r1, r0, r2
cmp r1, r3
bne Y
```

```plaintext
add r0, r1, #5
eor r0, r6, r0
sub r6, r6, #7
    ...
ldr r3, [r9]
    ...
orr r1, r0, r2
cmp r1, r3
bne Y
```
Preliminaries (Sub Basic Blocks)

- Instructions located in the private SPM may access the shared memory.

- A basic block’s WCET may depend on the bus schedule, despite residing in the private SPM.

⇒ Assume sub basic blocks, containing at maximum one such potential instruction.

```
add r0, r1, #5
eor r0, r6, r0
add r6, r6, #7
...

Z
```
```
ldr r3, [r9]
Z'
```
```
mov r1, r0, r2
cmp r1, r3
bne Y
Z''
```
Base ILP Model

```
add r0, r1, #5
eor r0, r6, r0
...
beq B
```

```
orr r3, r2, LSL #2
```

```
sub r3, r3, #4
ldr r1, [r3]
add r1, r1, #1024
```
Base ILP Model

\[ C_{A,\text{Flash}} = 390 \text{ Cycles} \]

\[ C_{B,\text{Flash}} = 96 \text{ Cycles} \]

\[ C_{C,\text{Flash}} = 114 \text{ Cycles} \]
Base ILP Model

$C_{A, \text{Flash}} = 390 \text{ Cycles}$
$C_{A, SPM} = 20 \text{ Cycles}$

$C_{B, \text{Flash}} = 96 \text{ Cycles}$
$C_{B, SPM} = 1 \text{ Cycle}$

$C_{C, \text{Flash}} = 114 \text{ Cycles}$
$C_{C, SPM} = 9 \text{ Cycles}$
Base ILP Model

\[ C_{A,Flash} = 390 \text{ Cycles} \]
\[ C_{A,SPM} = 20 \text{ Cycles} \]

\[ C_{B,Flash} = 96 \text{ Cycles} \]
\[ C_{B,SPM} = 1 \text{ Cycle} \]

\[ C_{C,Flash} = 114 \text{ Cycles} \]
\[ C_{C,SPM} = 9 \text{ Cycles} \]

\[ S_{SPM} = 20 \text{ B} \]
Base ILP Model

\[ w_B = C_{B,Flash} \]

\[ w_C = C_{C,Flash} \]
Base ILP Model

\[ w_A \geq C_{A,Flash} + w_B \]
\[ w_A \geq C_{A,Flash} + w_C \]

- **A**
  - add r0, r1, #5
  - eor r0, r6, r0
  - beq B

- **B**
  - orr r3, r2, LSL #2

- **C**
  - sub r3, r3, #4
  - ldr r1, [r3]
  - add r1, r1, #1024

\[ w_B = C_{B,Flash} \]
\[ w_C = C_{C,Flash} \]
Base ILP Model

\[ w_A \geq C_{A,Flash} + w_B \]
\[ w_A \geq C_{A,Flash} + w_C \]

\[ w_B = C_{B,Flash} \]
\[ w_C = C_{C,Flash} \]

\[ G_Y = C_{Y,Flash} - C_{Y,SPM} \]
Base ILP Model

\[ w_A \geq C_{A,\text{Flash}} - x_A \cdot G_A + w_B \]
\[ w_A \geq C_{A,\text{Flash}} - x_A \cdot G_A + w_C \]

\[ w_B = C_{B,\text{Flash}} - x_B \cdot G_B \]
\[ w_C = C_{C,\text{Flash}} - x_C \cdot G_C \]

\[ G_Y = C_{Y,\text{Flash}} - C_{Y,\text{SPM}} \]
Base ILP Model

\[ w_A \geq C_{A,\text{Flash}} - x_A \cdot G_A + w_B \]
\[ w_A \geq C_{A,\text{Flash}} - x_A \cdot G_A + w_C \]

add \ r0, r1, #5
eor \ r0, r6, r0
...
beq \ B

orr \ r3, r2, LSL #2

sub \ r3, r3, #4
ldr \ r1, [r3]
add \ r1, r1, #1024

\[ w_B = C_{B,\text{Flash}} - x_B \cdot G_B \]
\[ w_C = C_{C,\text{Flash}} - x_C \cdot G_C \]

\[ S_{SPM} \geq x_A \cdot S_A + x_B \cdot S_B + x_C \cdot S_C \]
Bus Offset Calculation

- Ingoing $o_{\nu}^{\text{In}}$ and outgoing bus offsets $o_{\nu}^{\text{Out}}$ are calculated per BB $\nu$.
Bus Offset Calculation

- Ingoing $o^\text{In}_\nu$ and outgoing bus offsets $o^\text{Out}_\nu$ are calculated per BB $\nu$

- $o_\nu = (o_{\text{low}}, o_{\text{high}})$
Bus Offset Calculation

- Ingoing $o_{\nu}^{\text{In}}$ and outgoing bus offsets $o_{\nu}^{\text{Out}}$ are calculated per BB $\nu$
- $o_{\nu} = (o_{\text{low}}, o_{\text{high}})$
- $o \in [0, \text{Bus Period} - 1]$
Bus-related Timings

- The WCET of a BB in the ILP model can be adjusted based on ...
  - bus offsets determined inside the ILP model
  - bus offsets from the WCET analysis

- Timing gain/penalty of a data access is expressed via $d_\nu$

- Bus-related timing of a jump correction is expressed via $l_{\nu,\mu}$
Bus-related Timings

- The WCET of a BB in the ILP model can be adjusted based on ...
  - bus offsets determined inside the ILP model
  - bus offsets from the WCET analysis

- Timing gain/penalty of a data access is expressed via \( d_\nu \)

- Bus-related timing of a jump correction is expressed via \( l_{\nu,\mu} \)
Bus-related Timings

- The WCET of a BB in the ILP model can be adjusted based on:
  - bus offsets determined inside the ILP model
  - bus offsets from the WCET analysis

- Timing gain/penalty of a data access is expressed via $d_{\nu}$

- Bus-related timing of a jump correction is expressed via $l_{\nu,\mu}$
Final ILP Model

\[ w_A \geq C_{A, \text{Flash}} - x_A \cdot G_A + w_B \]

\[ w_A \geq C_{A, \text{Flash}} - x_A \cdot G_A + w_C \]

\[ w_B = C_{B, \text{Flash}} - x_B \cdot G_B \]

\[ w_C \geq C_{C, \text{Flash}} - x_C \cdot G_C + w_{C'} \]

\[ w_{C'} \geq C_{C', \text{Flash}} - x_{C'} \cdot G_{C'} + w_{C''} \]

\[ w_{C''} = C_{C'', \text{Flash}} - x_{C''} \cdot G_{C''} \]

\[ S_{SPM} \geq x_A \cdot S_A + x_B \cdot S_B + x_C \cdot S_C \]
Final ILP Model

\[ w_A \geq C_{A, Flash} - x_A \cdot G_A + w_B + l_{A,B} \]
\[ w_A \geq C_{A, Flash} - x_A \cdot G_A + w_C + l_{A,C} \]
\[ w_B = C_{B, Flash} - x_B \cdot G_B \]

\[ w_C \geq C_{C, Flash} - x_C \cdot G_C + w_{C'} + l_{C,C'} \]
\[ w_{C'} \geq C_{C', Flash} - x_{C'} \cdot G_{C'} + w_{C''} + l_{C',C''} \]
\[ w_{C''} = C_{C'', Flash} - x_{C''} \cdot G_{C''} \]
\[ w_{C''} = C_{C'', Flash} - x_{C''} \cdot G_{C''} + d_{C'} \]

\[ S_{SPM} \geq x_A \cdot S_A + x_B \cdot S_B + x_C \cdot S_C \]
Overview

Bus-aware Static Instruction SPM Allocation
  Evaluation
  Conclusion

Compiler-based Event Arrival Function Extraction
  Extraction
  Outlook
Evaluation Setup

- MRTC benchmark suite
  - duff benchmark excluded (irregular loops are currently not supported)
  - petrinet & statemate excluded due to timeout
- Bus-unaware ILP-based instruction SPM allocation optimization as a baseline
- Evolutionary algorithm used as a reference for the ILP-based optimizations
- Evaluations performed on an Intel Xeon Server
- ILPs solved using Gurobi 7.0.1
- Compiled with the WCET-aware C compiler (WCC) using the -O2 flag
- SPM size was set to 50% of a benchmark’s size
ARM7TDMI Quadcore System

WCET

WCET Base

Bus-aware ILP
Evol. Algorithm
Overview

Bus-aware Static Instruction SPM Allocation
  Evaluation
  Conclusion

Compiler-based Event Arrival Function Extraction
  Extraction
  Outlook
Conclusion

• Instruction SPM allocation in a multicore environment needs special care

• Extended ILP model (under given assumptions) is able to:
  • Calculate TDMA bus offsets are within the ILP
  • Calculate bus timings based on the offsets

• 26% WCET reduction in average (in comparison to ILP-based bus-unaware optimization)

• Evolutionary algorithms are also applicable, but need more time in average
Conclusion

- Instruction SPM allocation in a multicore environment needs special care.

- Extended ILP model (under given assumptions) is able to:
  - Calculate TDMA bus offsets are within the ILP
  - Calculate bus timings based on the offsets

- 26% WCET reduction in average (in comparison to ILP-based bus-unaware optimization)

- Evolutionary algorithms are also applicable, but need more time in average.
Conclusion

• Instruction SPM allocation in a multicore environment needs special care

• Extended ILP model (under given assumptions) is able to:
  • Calculate TDMA bus offsets are within the ILP
  • Calculate bus timings based on the offsets

• 26% WCET reduction in average (in comparison to ILP-based bus-unaware optimization)

• Evolutionary algorithms are also applicable, but need more time in average
Conclusion

- Instruction SPM allocation in a multicore environment needs special care

- Extended ILP model (under given assumptions) is able to:
  - Calculate TDMA bus offsets are within the ILP
  - Calculate bus timings based on the offsets

- 26% WCET reduction in average (in comparison to ILP-based bus-unaware optimization)

- Evolutionary algorithms are also applicable, but need more time in average
Overview

Bus-aware Static Instruction SPM Allocation
  Evaluation
  Conclusion

Compiler-based Event Arrival Function Extraction
  Extraction
  Outlook
Event Arrival Functions

- Holistic WCET-estimation approaches for multicore architectures are very complex do not scale well

- System level analyses typically compute the worst-case timing using
  - WCET in isolation
  - Abstract notion of interfering events (e.g., task activations, shared memory accesses, ...)

⇒ Event arrival functions
Event Arrival Functions

- Holistic WCET-estimation approaches for multicore architectures are very complex do not scale well

- System level analyses typically compute the worst-case timing using
  - WCET in isolation
  - Abstract notion of interfering events (e.g., task activations, shared memory accesses, ...)

⇒ Event arrival functions
Event Arrival Functions

- Holistic WCET-estimation approaches for multicore architectures are very complex do not scale well

- System level analyses typically compute the worst-case timing using
  - WCET in isolation
  - Abstract notion of interfering events (e.g., task activations, shared memory accesses, ...)

⇒ Event arrival functions
Event Arrival Functions

• Holistic WCET-estimation approaches for multicore architectures are very complex do not scale well

• System level analyses typically compute the worst-case timing using
  • WCET in isolation
  • Abstract notion of interfering events (e.g., task activations, shared memory accesses, ...)

⇒ Event arrival functions
Event Arrival Functions

- Holistic WCET-estimation approaches for multicore architectures are very complex and do not scale well.

- System level analyses typically compute the worst-case timing using:
  - WCET in isolation
  - Abstract notion of interfering events (e.g., task activations, shared memory accesses, ...)

⇒ Event arrival functions
```c
int globalData[2] = {-1, 1};
volatile int comm;

int main() {
    for (int i = 0; i < 20; ++i) {
        if (comm == 0) {
            globalData[i % 2] = -1;
        }
    }
    return 0;
}
```
```c
int globalData[ 2 ] = {−1, 1};
volatile int comm;

int main() {
    for ( int i = 0; i < 20; ++i ) {
        if ( comm == 0 ) {
            globalData[ i % 2 ] = −1;
        }
    }
    return 0;
}
```
```c
int globalData[2] = {-1, 1};
volatile int comm;

int main() {
    for (int i = 0; i < 20; ++i) {
        if (comm == 0) {
            globalData[i % 2] = -1;
        }
    }
    return 0;
}
```
Extracting Curves?

• Capture traces
  ⇒ Potentially unsafe

• Rely on specifications
  ⇒ Potentially overly pessimistic

• Extraction based on the low-level representation
Extracting Curves?

- Capture traces
  - Potentially unsafe

- Rely on specifications
  - Potentially overly pessimistic

- Extraction based on the low-level representation
Extracting Curves?

- Capture traces  
  ⇒ Potentially unsafe

- Rely on specifications  
  ⇒ Potentially overly pessimistic

- Extraction based on the low-level representation
Extracting Curves?

- Capture traces
  \[\Rightarrow\] Potentially unsafe

- Rely on specifications
  \[\Rightarrow\] Potentially overly pessimistic

- Extraction based on the low-level representation
Extracting Curves?

- Capture traces
  - Potentially unsafe

- Rely on specifications
  - Potentially overly pessimistic

- Extraction based on the low-level representation
Bus-aware Static Instruction SPM Allocation

Compiler-based Event Arrival Function Extraction

Diagram:

- A
- B
- C
- D
- E
- Fun
- G
- H
- I
- F
- 2x...4x

Fun

Diagram details:

- Nodes A, B, C, D, E, F, G, H, I
- Connections between nodes
- Fun labeled node
- 2x...4x label
Bus-aware Static Instruction SPM Allocation

Compiler-based Event Arrival Function Extraction

A | B | E

0 20 40 60

A | C | F | G

0 20 40 60 80

A | C | F | H

0 20 40 60 80

A | C | F | G

0 20 40 60 80
Bus-aware Static Instruction SPM Allocation

Compiler-based Event Arrival Function Extraction
Bus-aware Static Instruction SPM Allocation

Compiler-based Event Arrival Function Extraction

\[ A \quad B \quad E \]

\[ A \quad C \quad F \quad G \]

\[ A \quad C \quad F \quad H \quad G \]

\[ A \quad C \quad F \quad G \]

\[ \ldots \]

\[ \ldots \]

\[ \ldots \]
Path Analysis for Arrival Functions

• Explicit path analysis quickly becomes practically infeasible
  ⇒ Sliding window for all traces

• Adapt implicit path enumeration technique (IPET)
  • First introduced by Jacobs et al. [RTNS15]
Path Analysis for Arrival Functions

- Explicit path analysis quickly becomes practically infeasible
  ⇒ Sliding window for all traces

- Adapt implicit path enumeration technique (IPET)
  - First introduced by Jacobs et al. [RTNS15]
Path Analysis for Arrival Functions

- Explicit path analysis quickly becomes practically infeasible
  ⇒ Sliding window for all traces

- Adapt implicit path enumeration technique (IPET)
  - First introduced by Jacobs et al. [RTNS15]
Path Analysis for Arrival Functions

- Explicit path analysis quickly becomes practically infeasible
  ⇒ Sliding window for all traces

- Adapt implicit path enumeration technique (IPET)
  - First introduced by Jacobs et al. [RTNS15]
Implicit Path Enumeration Technique

\[ p_F = 1 \]

\[ p_F = p_{F,G} + p_{F,H} \]

\[ p_{F,G} = p_{G,I} \]

\[ p_{F,H} = p_{H,I} \]

\[ p_{G,I} + p_{H,I} = 1 \]
Implicit Path Enumeration Technique

\[ p_F = 1 \]
\[ p_F = p_{F,G} + p_{F,H} \]
\[ p_{F,G} = p_{G,I} \]
\[ p_{F,H} = p_{H,I} \]
\[ p_{G,I} + p_{H,I} = 1 \]
Implicit Path Enumeration Technique

\[ p_F = 1 \]
\[ p_F = p_{F,G} + p_{F,H} \]
\[ p_{F,G} = p_{G,I} \]
\[ p_{F,H} = p_{H,I} \]
\[ p_{G,I} + p_{H,I} = 1 \]
Implicit Path Enumeration Technique

\[ p_F = 1 \]
\[ p_F = p_{F,G} + p_{F,H} \]
\[ p_{F,G} = p_{G,I} \]
\[ p_{F,H} = p_{H,I} \]
\[ p_{G,I} + p_{H,I} = 1 \]
Implicit Path Enumeration Technique

\[ p_F = 1 \]
\[ p_F = p_{F,G} + p_{F,H} \]
\[ p_{F,G} = p_{G,I} \]
\[ p_{F,H} = p_{H,I} \]
\[ p_{G,I} + p_{H,I} = 1 \]
Implicit Path Enumeration Technique

\[ p_F = 1 \]
\[ p_F = p_{F,G} + p_{F,H} \]
\[ p_{F,G} = p_{G,I} \]
\[ p_{F,H} = p_{H,I} \]
\[ p_{G,I} + p_{H,I} = 1 \]

Enforces a complete path through the program.
Overview

Bus-aware Static Instruction SPM Allocation
    Evaluation
    Conclusion

Compiler-based Event Arrival Function Extraction
    Extraction
    Outlook
Adapted IPET

Introducing “movable” sources and sinks.

$$p_{F,G-e_G} = p_{G,I-s_G,I}$$

$$p_{G,I} + p_{H,I-e_I} = 0$$

Calculating number of events on path.

$$a_{Tot} = \sum_i A_i \cdot p_i$$

Maximize (resp. minimize) for a given time interval $\Delta t$.

$$\max : a_{Tot}, \text{ while } \Delta t \geq \sum_i C_i \cdot p_i - (...)$$
Adapted IPET

Introducing “movable” sources and sinks.

\[ p_{F,G} - e_G = p_{G,I} - s_G, I \]
\[ p_{G,I} + p_{H,I} - e_I = 0 \]

Calculating number of events on path.

\[ a_{\text{Tot}} = \sum_i A_i \cdot p_i \]

Maximize (resp. minimize) for a given time interval \( \Delta t \).

\[ \max : a_{\text{Tot}}, \text{ while } \Delta t \geq \sum_i C_i \cdot p_i \quad - (\ldots) \]
Adapted IPET

Introducing “movable” sources and sinks.

\[ p_{F,G-e_G} = p_{G,I-S_G,I} \]
\[ p_{G,I} + p_{H,I-e_I} = 0 \]

Calculating number of events on path.

\[ a_{\text{Tot}} = \sum_i A_i \cdot p_i \]

Maximize (resp. minimize) for a given time interval \( \Delta t \).

\[ \max : a_{\text{Tot}}, \text{ while } \Delta t \geq \sum_i C_i \cdot p_i - (...) \]
Adapted IPET

Introducing “movable” sources and sinks.

\[ p_{F,G-eG} = p_{G,I-SG,I} \]
\[ p_{G,I} + p_{H,I-eI} = 0 \]

Calculating number of events on path.

\[ a_{\text{Tot}} = \sum_i A_i \cdot p_i \]

Maximize (resp. minimize) for a given time interval \( \Delta t \).

\[ \max : a_{\text{Tot}}, \ \text{while} \ \Delta t \geq \sum_i C_i \cdot p_i \ \text{---(...)} \]
Adapted IPET

Introducing “movable” sources and sinks.

\[ p_{F,G} - e_G = p_{G,I} - s_{G,I} \]
\[ p_{G,I} + p_{H,I} - e_I = 0 \]

Calculating number of events on path.

\[ a_{\text{Tot}} = \sum_i A_i \cdot p_i \]

Maximize (resp. minimize) for a given time interval \( \Delta t \).

\[ \max : a_{\text{Tot}}, \text{ while } \Delta t \geq \sum_i C_i \cdot p_i \quad (\ldots) \]
Adapted IPET

Introducing “movable” sources and sinks.

\[ p_{F,G-eG} = p_{G,I-SG,I} \]

\[ p_{G,I} + p_{H,I-eI} = 0 \]

Calculating number of events on path.

\[ a_{\text{Tot}} = \sum_i A_i \cdot p_i \]

Maximize (resp. minimize) for a given time interval \( \Delta t \).

\[ \max : a_{\text{Tot}}, \text{ while } \Delta t \geq \sum_i C_i \cdot p_i \quad -(...) \]
Automated Extraction

- Two dimensions of granularity:
  - Events per basic block
  - Sample rate

- Adjustable trade-off between runtime and tightness
  - Albeit, arrival functions will be safe
Automated Extraction

- Two dimensions of granularity:
  - Events per basic block
  - Sample rate

- Adjustable trade-off between runtime and tightness
  - Albeit, arrival functions will be safe
Automated Extraction

- Two dimensions of granularity:
  - Events per basic block
  - Sample rate

- Adjustable trade-off between runtime and tightness
  - Albeit, arrival functions will be safe
Automated Extraction

- Two dimensions of granularity:
  - Events per basic block
  - Sample rate

- Adjustable trade-off between runtime and tightness
  - Albeit, arrival functions will be safe
Automated Extraction

- Two dimensions of granularity:
  - Events per basic block
  - Sample rate

- Adjustable trade-off between runtime and tightness
  - Albeit, arrival functions will be safe
Bus-aware Static Instruction SPM Allocation

Compiler-based Event Arrival Function Extraction

- Number of Events
- \( \Delta t \) in Cycles

50 Sample Points
100 Sample Points
Overview

Bus-aware Static Instruction SPM Allocation
   Evaluation
   Conclusion

Compiler-based Event Arrival Function Extraction
   Extraction
   Outlook
Future Applications

- Bus-oriented low-level optimizations

- Precise analyses of complex multicore architectures

- Extension to multitask-multicore
Future Applications

- Bus-oriented low-level optimizations
- Precise analyses of complex multicore architectures
- Extension to multitask-multicore
Future Applications

- Bus-oriented low-level optimizations
- Precise analyses of complex multicore architectures
- Extension to multitask-multicore