Tight WCRT Analysis of synchronous C programs

Partha S Roop, Sidharta Andalam, Reinhard von Hanxleden, Simon Yuan, Claus Traulsen

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

CASES
October 2009
Overview of presentation

Outline

Problem and motivation.
PRET-C for modelling time-critical systems.
Intermediate format for timing analysis.
Model checking based analysis.
Results and conclusions.
Overview of presentation

Outline

- Problem and motivation.
Overview of presentation

Outline

- Problem and motivation.
- PRET-C for modelling time-critical systems.
Overview of presentation

Outline

- Problem and motivation.
- PRET-C for modelling time-critical systems.
- Intermediate format for timing analysis.
Outline

- Problem and motivation.
- PRET-C for modelling time-critical systems.
- Intermediate format for timing analysis.
- Model checking based analysis.
Overview of presentation

Outline

- Problem and motivation.
- PRET-C for modelling time-critical systems.
- Intermediate format for timing analysis.
- Model checking based analysis.
- Results and conclusions.
Problem and motivation

main

loop each tick{
    ReadInputs();
    ReactiveFunction();
    EmitOutputs();
}

How to determine the "tight/optimal" worst case tick length of the Reactive Function?
Related Work

Current approaches

MaxPlus [Boldt et al., SLA++P’07]: CKA G based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.

WCRT algebra [Mendler et al., DATE’09]: max-plus algebra with support for infeasible path pruning.

Timed KS [Logothetis et al., RTSS’03]: Synchronous program compiled into a timed Kripke structure.

ILP formulation [Ju et al., CODES+ISSS’08]: Esterel program mapped to C using CEC and then the C code is analysed.

Limitation

All existing approaches ignore state-dependencies while determining infeasible paths. A model checking based formulation may be used to compute the reachable state-space and this reachability analysis automatically prunes unreachable paths.
Related Work

Current approaches

- MaxPlus [Boldt et al., SLA++P’07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.

- W CRT algeb ra [Mendler et al., DATE’09]: max-plus algebra with support for infeasible path pruning.

- Timed KS [Logothetis et al., RTSS’03]: Synchronous program compiled into a timed Kripke structure.

- ILP formulation [Ju et al., CODES+ISSS’08]: Esterel program mapped to C using CEC and then the C code is analysed.

Limitation

All existing approaches ignore state-dependencies while determining infeasible paths. A model checking based formulation may be used to compute the reachable state-space and this reachability analysis automatically prunes unreachable paths.
Related Work

Current approaches

- MaxPlus [Boldt et al., SLA++P’07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.

- WCRT algebra [Mendler et al., DATE’09]: max-plus algebra with support for infeasible path pruning.

Limitation

All existing approaches ignore state-dependencies while determining infeasible paths. A model checking based formulation may be used to compute the reachable state-space and this reachability analysis automatically prunes unreachable paths.
Related Work

Current approaches

- **MaxPlus** [Boldt et al., SLA++P’07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.
- **WCRT algebra** [Mendler et al., DATE’09]: max-plus algebra with support for infeasible path pruning.
- **Timed KS** [Logothetis et al., RTSS’03]: Synchronous program compiled into a timed Kripke structure.

Limitation

All existing approaches ignore state-dependencies while determining infeasible paths. A model checking based formulation may be used to compute the reachable state-space and this reachability analysis automatically prunes unreachable paths.
Related Work

Current approaches

- **MaxPlus** [Boldt et al., SLA++P’07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.
- **WCRT algebra** [Mendler et al., DATE’09]: max-plus algebra with support for infeasible path pruning.
- **Timed KS** [Logothetis et al., RTSS’03]: Synchronous program compiled into a timed Kripke structure.
- **ILP formulation** [Ju et al., CODES+ISSS’08]: Esterel program mapped to C using CEC and then the C code is analysed.

Limitation

All existing approaches ignore state-dependencies while determining infeasible paths. A model checking based formulation may be used to compute the reachable state-space and this reachability analysis automatically prunes unreachable paths.
### Related Work

#### Current approaches

- **MaxPlus** [Boldt et al., SLA++P’07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.

- **WCRT algebra** [Mendler et al., DATE’09]: max-plus algebra with support for infeasible path pruning.

- **Timed KS** [Logothetis et al., RTSS’03]: Synchronous program compiled into a timed Kripke structure.

- **ILP formulation** [Ju et al., CODES+ISSS’08]: Esterel program mapped to C using CEC and then the C code is analysed.

#### Limitation

All existing approaches ignore state-dependencies while determining infeasible paths. A model checking based formulation may be used to compute the reachable state-space and this reachability analysis automatically prunes unreachable paths.
Example 1: Two simple threads
Example 1: Two simple threads
Example 1: Two simple threads
Example 1: Two simple threads
Example 1: Two simple threads
Example 1: Two simple threads
Example 1: Two simple threads
Example 1: Two simple threads
Example 1: Two simple threads

What is the WCRT of this program?
Example 1: Two simple threads

\[ WCRT = \max(T_1) + \max(T_2) \]
\[ = 13 + 15 \]
\[ = 28 \]
Motivation for the proposed approach

How?

Tight analysis depends on:

a) Data/Control dependency.
b) Tick/State alignment across threads.
Motivation for the proposed approach

A can be True or False

a) By considering Data/Control dependencies.
a) By considering Data/Control dependencies.
Motivation for the proposed approach

When True:
WCRT = 13 + 8 = 21

When False:
WCRT = 10 + 15 = 25

a) By considering Data/Control dependencies.
Motivation for the proposed approach

When True:
\[ WCRT = 13 + 8 = 21 \]

When False:
\[ WCRT = 10 + 15 = 25 \]

\[ WCRT = \max(21, 25) = 25 \]

a) By considering Data/Control dependencies.

Roop (University of Auckland)
Our Solution

Tighter

Tighter analysis by taking state and data dependencies into account. We can also keep track of our variables in every tick for further infeasible path pruning.
Our Solution

Tighter

Tighter analysis by taking state and data dependencies into account. We can also keep track of our variables in every tick for further infeasible path pruning.
Our Solution

Tighter

Tighter analysis by taking state and data dependencies into account. We can also keep track of our variables in every tick for further infeasible path pruning.

Model Checking

Synchronous C programs may be represented as a set of concurrent FSMs with transition guards that represent the execution cost. This can be exploited by a model checker to determine the reachable state-space and the maximum tick length.
b) By considering only Tick/State alignment.
b) By considering only Tick/State alignment.
Motivation for the proposed approach

b) By considering only **Tick/State alignment**.

Tick 0: $13 + 7 = 20$

Tick 1: $6 + 15 = 21$
Motivation for the proposed approach

b) By considering only Tick/State alignment.

Tick 0: $13 + 7 = 20$

Tick 1: $6 + 15 = 21$

$WCRT = \max(20, 21) = 21$
c) By considering data dependencies, tick alignment and by tracking the value of the variables.
c) By considering data dependencies, tick alignment and by tracking the value of the variables.
Motivation for the proposed approach

A is true during this scope.

c) By considering data dependencies, tick alignment and by tracking the value of the variables.
Motivation for the proposed approach

A is true during this scope.

Tick 0: $13 + 7 = 20$

c) By considering **data dependencies**, **tick alignment** and by **tracking** the value of the variables.
Motivation for the proposed approach

A is true during this scope.

Tick 0: $13 + 7 = 20$

Tick 1: $6 + 8 = 14$

c) By considering data dependencies, tick alignment and by tracking the value of the variables.
Motivation for the proposed approach

A is true during this scope.

Tick 0: $13 + 7 = 20$

Tick 1: $6 + 8 = 14$

$\text{WCRT} = \max(20, 14) = 20$

c) By considering data dependencies, tick alignment and by tracking the value of the variables.
Comparison

Methods
Comparison

Methods

- MaxPlus: 28
Comparison

**Methods**

- MaxPlus: 28
- MaxPlus + Data/Control: 25
Comparison

Methods

- MaxPlus: 28
- MaxPlus + Data/Control: 25
- MaxPlus + Tick alignment: 21
- MaxPlus + Data/Control + Track Variable: 20
Comparison

Methods
- MaxPlus: 28
- MaxPlus + Data/Control: 25
- MaxPlus + Tick alignment: 21
- MaxPlus + Data/Control + Tick alignment: 21
## Comparison

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxPlus</td>
<td>28</td>
</tr>
<tr>
<td>MaxPlus + Data/Control</td>
<td>25</td>
</tr>
<tr>
<td>MaxPlus + Tick alignment</td>
<td>21</td>
</tr>
<tr>
<td>MaxPlus + Data/Control + Tick alignment</td>
<td>21</td>
</tr>
<tr>
<td>Data/Control + Tick alignment + Track Variable</td>
<td>20</td>
</tr>
</tbody>
</table>
Overview of the Timing Analysis

Stages

1. PRET-C: simple synchronous extension to C (using macros).

2. TCCF G: intermediate format.

3. TFSM: FSM denoted with execution costs.

4. Model Checking: calculates the WCRT based on a set of TFSMs and a safety property.

Code

```c
void main() {
    while(1) {
        abort
            PAR(sampler,display);
        when pre (reset);
        EOT;
    }
}
```
Overview of the Timing Analysis

Stages

1. PRET-C: simple synchronous extension to C (using macros).
2. TCCFG: intermediate format.

Code

```c
void main() {
    while(1) {
        abort
        PAR(sampler,display);
        when pre (reset);
        EOT;
    }
}
```

TCCFG
Overview of the Timing Analysis

**Stages**

1. **PRET-C**: simple synchronous extension to C (using macros).
2. **TCCFG**: intermediate format.
3. **TFSM**: FSM denoted with execution costs.

**Code**

```c
void main() {
    while(1) {
        abort
        PAR(sampler,display);
        when pre (reset);
        EOT;
    }
}
```

**TCCFG**

**TFSM**
Overview of the Timing Analysis

Stages

1. PRET-C: simple synchronous extension to C (using macros).
2. TCCFG: intermediate format.
3. TFSM: FSM denoted with execution costs.
4. Model Checking: calculates the WCRT based on a set of TFSMs and a safety property.

Code

void main() {
    while(1) {
        abort
        PAR(samp
        when pre (EOT;
    }
}
Overview of the Timing Analysis

Stages

1. PRET-C: simple synchronous extension to C (using macros).
2. TCCFG: intermediate format.
3. TFSM: FSM denoted with execution costs.
4. Model Checking: calculates the WCRT based on a set of TFSMs and a safety property.

Code

```c
void main() {
    while(1) {
        abort
        PAR(samp
        when pre (
            EOT;
        }
    }
```
Precision Timed C (PRET-C)

Simple set of synchronous extensions to C for:

- light-weight multithreading in C.
- all extensions implemented as C macros.
- provides thread-safe shared memory access.
- supports predictable programming by mapping logical time to physical time through static analysis.
## Synchronous extensions to C

<table>
<thead>
<tr>
<th>Statement</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReactiveInput I</td>
<td>declares I as a reactive input coming from the environment</td>
</tr>
<tr>
<td>ReactiveOutput O</td>
<td>declares O as a reactive output emitted to the environment</td>
</tr>
<tr>
<td>PAR(T1,...,Tn)</td>
<td>synchronously executes in parallel the n threads Ti, with higher priority of Ti over Ti+1</td>
</tr>
<tr>
<td>EOT</td>
<td>marks the end of a tick (local or global depending on its position)</td>
</tr>
<tr>
<td>[weak] abort P when pre C</td>
<td>immediately kills P when C is true in the previous instant</td>
</tr>
</tbody>
</table>

**Table:** PRET-C extensions to C.
#include <pretc.h>
#define N 1000

ReactiveInput (int, reset, 0);
ReactiveInput (float, sensor, 0.0);
ReactiveOutput(float, out, 0.0);

int cnt=0;
float buffer[N];
Example: Producer Consumer

```c
void main() {
    while(1) {
        abort
        flush(buffer);
        PAR(sampler, display);
        when pre (reset);
        cnt=0;
        EOT;
    }
}
```
void main() {
    while(1) {
        abort
        flush(buffer);
        PAR(sampler,display);
        when pre (reset);
        cnt=0;
        EOT;
    }
}
Example: Producer Consumer

```c
void main() {
    while(1) {
        abort
        flush(buffer);
        PAR(sampler,display);
        when pre (reset);
        cnt=0;
        EOT;
    }
}
```

Clears the buffer and then spawns the sampler and the display threads.
```c
void main() {
    while(1) {
        abort
        flush(buffer);
        PAR(sampler,display);
        when pre (reset);
        cnt=0;
        EOT;
    }
}
```

When preempted, cnt variable is set to zero and then executes the EOT macro.
Example: Producer Consumer

```c
void main() {
    while(1) {
        abort
        flush(buffer);
        PAR(sampler,display);
        when pre (reset);
        cnt=0;
        EOT;
    }
}
```

Body restarts in the next tick.
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}

void display() {
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}

void display(){
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}

Tick0:

i=0
sample=1.0
out=0.0
buffer={0.0, 0.0, 0.0}
**Example: Producer Consumer**

```c
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}
```

```c
void display(){
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}
```

**Tick0:**

- `i=0`
- `sample=1.0`
- `out=0.0`
- `cnt =0`
- `buffer={0.0, 0.0, 0.0}`
void sampler() {
  int i = 0; float sample;
  while (1) {
    sample = read(sensor);
    EOT;
    while (cnt==N) EOT;
    buffer[i] = sample;
    EOT;
    i = (i + 1) % N
    cnt = cnt + 1;
  }
}

void display() {
  int i = 0; float out;
  while (1) {
    EOT;
    while (cnt==0) EOT;
    out = buffer[i]
    EOT;
    i = (i + 1) % N;
    cnt = cnt - 1;
    EOT;
    WriteLCD(out);
  }
}
Example: Producer Consumer

```c
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}

void display(){
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}
```

Tick 1:

i=0
sample=1.0
out=0.0
cnt =0
buffer={1.0, 0.0, 0.0}
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}

doxygen void display() {
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}

Tick 2:

\[ \begin{array}{ccc}
    i=1 & i=0 & cnt =1 \\
    sample=2.0 & out=0.0 & buffer={1.0, 0.0, 0.0} \\
\end{array} \]
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        cnt = cnt + 1;
        i = (i + 1)% N
    }
}

do display() {
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}
void sampler() {
  int i = 0; float sample;
  while (1) {
    sample = read(sensor);
    EOT;
    while (cnt==N) EOT;
    buffer[i] = sample;
    EOT;
    i = (i + 1)% N
    cnt = cnt + 1;
  }
}

void display() {
  int i = 0; float out;
  while (1) {
    EOT;
    while (cnt==0) EOT;
    out = buffer[i]
    EOT;
    i = (i + 1)% N
    cnt = cnt - 1;
    EOT;
    WriteLCD(out);
  }
}

Tick 3:

i=1  i=0  cnt =1
sample=2.0  out=1.0  buffer={1.0, 2.0, 0.0}
Example: Producer Consumer

```c
void sampler()
{
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}

void display()
{
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}
```

Tick 3:

```
i=1
sample=2.0
```
```
i=1
out=1.0
```
```
cnt =0
buffer={1.0, 2.0, 0.0}
```
Example: Producer Consumer

void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}

void display() {
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}

Tick 4: i=2 i=1 cnt =1
sample=3.0 out=1.0 buffer={1.0, 2.0, 0.0}
Example: Producer Consumer

```c
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}

void display(){
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}
```

Tick 4:

```
i=2
sample=3.0
```
```
i=1
out=1.0
```
```
cnt =1
buffer={1.0, 2.0, 0.0}
```
Example: Producer Consumer

```c
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1) % N
        cnt = cnt + 1;
    }
}
```

```c
void display() {
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1) % N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}
```

Tick 5:

- i = 2
- sample = 3.0
- i = 1
- out = 1.0
- cnt = 1
- buffer = {1.0, 2.0, 3.0}
Example: Producer Consumer

```c
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}
```

```c
void display(){
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}
```

Tick 5:

i=2 sample=3.0 i=1 out=2.0 cnt =1 buffer={1.0, 2.0, 3.0}
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}

void display() {
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        WriteLCD(out);
    }
}

Tick 6:

i=3
sample=4.0
i=1
out=2.0
cnt =2
buffer={1.0, 2.0, 3.0}
void sampler() {
    int i = 0; float sample;
    while (1) {
        sample = read(sensor);
        EOT;
        while (cnt==N) EOT;
        buffer[i] = sample;
        EOT;
        i = (i + 1)% N
        cnt = cnt + 1;
    }
}

void display() {
    int i = 0; float out;
    while (1) {
        EOT;
        while (cnt==0) EOT;
        out = buffer[i]
        EOT;
        i = (i + 1)% N;
        cnt = cnt - 1;
        EOT;
        writeLCD(out);
    }
}

Tick 6:

i=3  i=2  cnt =1
sample=4.0  out=2.0  buffer={1.0, 2.0, 3.0}
Design Flow: PRET-C to WCRT

Stages

1. PRET-C to Assembly: standard gcc based compilers can be used.
2. Assembly to TCCFG: our code analyser.
3. TCCFG to Model Checker: our FSM generator (XML).
4. CTL temporal logic property checking: bounded integer checking.

Overview

PRET-C

Assembly (Microblaze)

TCCFG

Model for Model Checker (UPPAAL)

WCRT value

mb-gcc

TCCFG gen

FSM gen

Execution (Microblaze)

Architecture Specifications

Verifying CTL properties

Roop (University of Auckland)

PRET

CASES'09
**Timed Concurrent Control Flow Graph**

**TCCFG**

An intermediate format to represent the control flow of the program.

- Easy to **visualise** the entire program flow.
- Supports **concurrency** using fork & join.
- Supports **preemption** using checkaborts.
- Easy to **analyse** the flow.

---

**Nodes**

- **Start/End**
- **Action node**
- **Conditional**
- **EOT**
- **Jump**
- **Fork**
- **Join**
- **Abort Start**
- **Abort End**
- **Chkabor**
Mapping PRET-C to TCCFG

PRET-C

```c
void sampler() {
    ...... 
    while (cnt==N) EOT;
    buffer[i] = sample;
    EOT;
    ...... 
}
```

Assembly

```
lwi r3,r0,cnt
xori r18,r3,1000
beqi r18,$L50
```

```
# code for buffer[i]=sample
......
$L50:
```

### Execution cost

3
6
7

TCCFG

- (3) \(\text{cnt==N}\)
- (7+5)
- (0) \(\text{chk}\)
- (5) \(\text{Jump}\)
- (6) \(\text{buffer[i]=sample}\)
- (7)
void sampler() {
    ....
    while (cnt==N) EOT;
    buffer[i] = sample;
    EOT;
    ....
}

---

Assembly

lwi r3,r0,cnt
xori r18,r3,1000
beqi r18,$L50

## code for ##

buffer[i]=sample

....

$L50:

## code for EOT##
void sampler() {
    ...... 
    while (cnt==N) EOT;
    buffer[i] = sample;
    EOT;
    ...... 
}

**Execution cost**

- Assembly
  - `lwi r3,r0,cnt`
  - `xori r18,r3,1000`
  - `beqi r18,$L50`
- `buffer[i]=sample`...
- `$L50:`
- `## code for EOT##`

Mapping PRET-C to TCCFG

**PRET-C**

**TCCFG**

- **(3) cnt==N**
- **(7+5) chk**
- **(0) chk**
- **(5) Jump**
- **(6) buffer[i]=sample**
- **(7)**
Mapping PRET-C to TCCFG

PRET-C

```c
void sampler() {
    ....
    while (cnt==N) EOT;
    buffer[i] = sample;
    EOT;
    ....
}
```

TCCFG

```
(3) cnt==N
(7+5)
(0) chk
(5) Jump
(6) buffer[i]=sample
(7)
```

```
Execution cost

3 lwi r3,r0,cnt
6 xori r18,r3,1000
6 beqi r18,$L50
## code for ##
6 buffer[i]=sample
7 ....
7 $L50:
7 ##code for EOT#
```
Mapping TCCFG to TFSM

**TCCFG**

1. out = 0.0, i = 0
2. cnt = 0
3. out = buffer[i]
4. i = (i + 1) % N
5. cnt = (cnt - 1) % N
6. WriteLCD()

**TFSM**

- EOT1
- EOT2
- EOT3
- EOT4
Mapping TCCFG to TFSM

TCCFG

out=0.0
i=0

cnt==0
Jump

out=
buffer[i]
i=(i+1)%N
cnt=(cnt-1)%N
WriteLCD()
Jump

(6)
(23)
(23+5)
(5)
(6)

TFSM

EOT1

EOT2

EOT3

EOT4

30
56
28
30
29
29
31
56
28
29

Roop (University of Auckland)
Mapping TCCFG to TFSM

TCCFG

1. out=0.0
2. i=0
3. cnt==0
4. Jump
5. out=buffer[i]
6. i=(i+1)%N
7. cnt=(cnt-1)%N
8. WriteLCD()
9. Jump

TFSM

1. EOT1
2. cnt==0
3. Jump
4. EOT2
5. EOT3
6. EOT4
Mapping TCCFG to TFSM

TCCFG

(6) out=0.0
i=0

(23) EOT1

(2) cnt==0

(23+5) EOT2

(5) Jump

(6) out= buffer[i]

(23) EOT3

TFSM

29

28

30

31

56

29

2+6+23 = 31
Timing Analysis

\[ W_{CRT} \text{ min} \] is the sum of the minimum local tick lengths of the threads.

\[ W_{CRT} \text{ max} \] is the sum of the maximum local tick lengths of the threads.

The \( W_{CRT} \) value of the program lies in the interval\( [W_{CRT} \text{ min}, W_{CRT} \text{ max}] \).
\( WCRT_{\text{min}} \) is the sum of the minimum local tick lengths of the threads.
- \( WCRT_{\text{min}} \) is the sum of the minimum local tick lengths of the threads.
- \( WCRT_{\text{max}} \) is the sum of the maximum local tick lengths of the threads.
Timing Analysis

- $WCRT_{\text{min}}$ is the sum of the minimum local tick lengths of the threads.
- $WCRT_{\text{max}}$ is the sum of the maximum local tick lengths of the threads.
- The WCRT value of the program lies in the interval $[WCRT_{\text{min}}, WCRT_{\text{max}}]$. 
Timing Analysis

- $WCRT_{min}$ is the sum of the minimum local tick lengths of the threads.
- $WCRT_{max}$ is the sum of the maximum local tick lengths of the threads.
- The WCRT value of the program lies in the interval $[WCRT_{min}, WCRT_{max}]$. 

![Diagram showing EOT1, EOT2, EOT3, and EOT4 with tick lengths and values]
Timing Analysis

- \( WCRT_{\text{min}} \) is the sum of the minimum local tick lengths of the threads.
- \( WCRT_{\text{max}} \) is the sum of the maximum local tick lengths of the threads.
- The WCRT value of the program lies in the interval \([WCRT_{\text{min}}, WCRT_{\text{max}}]\).

\[
WCRT_{\text{min}} = 15 + 28 = 43
\]
Timing Analysis

- $WCRT_{min}$ is the sum of the minimum local tick lengths of the threads.
- $WCRT_{max}$ is the sum of the maximum local tick lengths of the threads.
- The WCRT value of the program lies in the interval $[WCRT_{min}, WCRT_{max}]$.

\[ WCRT_{min} = 15 + 28 = 43 \]
\[ WCRT_{max} = 30 + 56 = 86 \]
The goal of static analysis is to search for a value of WCRT in this interval that is as close to the actual WCRT of the program as possible. We call this value the tight WCRT of the program, denoted $WCRT_{tight}$. 
Motivation for Model Checking

Model checking is a reachability analysis process for efficiently searching over very large state spaces. Model checking can effectively deal with the well-known problem of state-space explosion, when the number of concurrent threads is large. In the past, model checking has already been shown to be an effective tool for WCET analysis [Metzner, CA V 2004].

Since we need to operate over unbounded integers, we have selected the UPPAAL model checker that supports this effectively.
Motivation for Model Checking

- Model checking is a reachability analysis process for efficiently searching over very large state spaces.
Model checking is a reachability analysis process for efficiently searching over very large state spaces.

Model checking can effectively deal with the well known problem of state-space explosion, when the number of concurrent threads is large.
Motivation for Model Checking

- Model checking is a reachability analysis process for efficiently searching over very large state spaces.
- Model checking can effectively deal with the well known problem of state-space explosion, when the number of concurrent threads is large.
- In the past, model checking has already been shown to be an effective tool for WCET analysis [Metzner, CAV2004].
Motivation for Model Checking

- Model checking is a reachability analysis process for efficiently searching over very large state spaces.
- Model checking can effectively deal with the well known problem of *state-space explosion*, when the number of concurrent threads is large.
- In the past, model checking has already been shown to be an effective tool for WCET analysis [Metzner, CAV2004].
- Since we need to operate over bounded integers, we have selected the UPPAAL model checker that supports this effectively.
Model checking formulation

- We convert TFSMs to the input format of the model checker UPPAAL called timed automata (TA). Our timed automaton do not use any clocks and instead have only bounded integers.
Model checking formulation

- We convert TFSMs to the input format of the model checker UPPAAL called timed automata (TA). Our timed automaton do not use any clocks and instead have only bounded integers.
- We compose all TAs in parallel and introduce a barrier TA to emulate the synchronous composition.

\[ \text{We perform binary search in the interval } [W_{CRT}\text{min}, W_{CRT}\text{max}] \]
Model checking formulation

- We convert TFSMs to the input format of the model checker UPPAAL called timed automata (TA). Our timed automaton do not use any clocks and instead have only bounded integers.
- We compose all TAs in parallel and introduce a barrier TA to emulate the synchronous composition.
- We then evaluate a set of queries of the form $AG(gtick \Rightarrow x \leq val)$.
We convert TFSMs to the input format of the model checker UPPAAL called timed automata (TA). Our timed automaton do not use any clocks and instead have only bounded integers.

We compose all TAs in parallel and introduce a barrier TA to emulate the synchronous composition.

We then evaluate a set of queries of the form $AG(gtick \Rightarrow x \leq val)$.

We perform binary search in the interval $[WCRT_{min}, WCRT_{max}]$ by successively trying different $val$ until the tight value is found.
Mapping TFSM to Timed Automata

Thread 1

- **EOT1**
  - 15
  - ![gtick](x = x + 30)  
  - ![lt1](true)

- **EOT2**
  - 30
  - ![gtick](x = x + 30)  
  - ![lt1](false)

- **b01**
  - ![gtick](x = x + 15)  
  - ![lt1](false)

- **b12**
  - ![gtick](x = x + 30)  
  - ![lt1](false)

Roop (University of Auckland)
Mapping TFSM to Timed Automata

Thread 1

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 EOT1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 EOT2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>!gtick</td>
<td></td>
<td>x = x + 30</td>
</tr>
<tr>
<td>gtick</td>
<td>lt1 = true</td>
<td>x = x</td>
</tr>
<tr>
<td>EOT1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b01</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gtick</td>
<td>lt1 = false</td>
<td>x = x</td>
</tr>
<tr>
<td>b12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOT2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thread 2

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 EOT1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25 EOT2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>!gtick</td>
<td></td>
<td>x = x + 20</td>
</tr>
<tr>
<td>gtick</td>
<td>lt2 = true</td>
<td>x = x</td>
</tr>
<tr>
<td>EOT1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b01</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gtick</td>
<td>lt2 = false</td>
<td>x = x</td>
</tr>
<tr>
<td>b12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOT2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mapping TFSM to Timed Automata

\[ lt1 = false \]

- \( ! \text{gtick} \)
- \( x = x + 30 \)
- \( lt1 = true \)

- \( \text{gtick} \)
- \( \text{lt1} = false \)

- \( \text{gtick} \)
- \( \text{lt1} = false \)

\[ lt2 = false \]

- \( ! \text{gtick} \)
- \( x = x + 20 \)
- \( lt2 = true \)

- \( \text{gtick} \)
- \( \text{lt2} = false \)

- \( \text{gtick} \)
- \( \text{lt2} = false \)

\[ gtick = false \quad x = 0 \]

- \( \text{lt1} \land \text{lt2} \)
- \( ! \text{lt1} \land ! \text{lt2} \)
- \( \text{WaitLT} \)
- \( \text{U} \)
- \( \text{GTReached} \)
- \( \text{gtick} = true \)
- \( \text{lt1} \land \text{lt2} \)
- \( \text{gtick} = false \)
- \( x = 0 \)

\[ \text{lt1} = false \]
\[ \text{lt2} = false \]
\[ \text{gtick} = false \]
\[ x = 0 \]
Mapping TFSM to Timed Automata

\[
\begin{align*}
\text{lt1} &= \text{true} \\
\text{lt2} &= \text{false}
\end{align*}
\]
Mapping TFSM to Timed Automata

lt1 = true

! gtick
x = x + 30
lt1 = true

lt1 = false

gtick

lt1 = false

lt1 = true

lt2 = false

! gtick
x = x + 20
lt2 = true

lt2 = true

gtick

lt2 = false

lt1 ∧ lt2
GTRReached

gtick = true

lt1 ∧ ! lt2

gtick = false

x = 15

x = 0

Roop (University of Auckland)
Mapping TFSM to Timed Automata

\[ \text{lt1} = \text{true} \]

\[ \begin{align*}
! \text{gtick} \\
& \quad x = x + 30 \\
& \quad \text{lt1} = \text{false} \\
\end{align*} \]

\[ \begin{align*}
\text{gtick} \\
& \quad \text{lt1} = \text{false} \\
\end{align*} \]

\[ \text{EOT1} \]

\[ \begin{align*}
! \text{gtick} \\
& \quad x = x + 15 \\
& \quad \text{lt1} = \text{true} \\
\end{align*} \]

\[ \begin{align*}
\text{gtick} \\
& \quad \text{lt1} = \text{false} \\
\end{align*} \]

\[ \text{EOT2} \]

\[ \begin{align*}
\text{lt1} = \text{true} \\
\end{align*} \]

\[ \begin{align*}
\text{lt2} = \text{true} \\
\end{align*} \]

\[ \begin{align*}
! \text{gtick} \\
& \quad x = x + 20 \\
& \quad \text{lt2} = \text{false} \\
\end{align*} \]

\[ \begin{align*}
\text{gtick} \\
& \quad \text{lt2} = \text{false} \\
\end{align*} \]

\[ \text{EOT1} \]

\[ \begin{align*}
\text{lt2} = \text{true} \\
\end{align*} \]

\[ \begin{align*}
! \text{gtick} \\
& \quad x = x + 25 \\
& \quad \text{lt2} = \text{true} \\
\end{align*} \]

\[ \begin{align*}
\text{gtick} \\
& \quad \text{lt2} = \text{false} \\
\end{align*} \]

\[ \text{EOT2} \]

\[ \text{gtick} = \text{false} \quad x = 40 \]
Mapping TFSM to Timed Automata

\[
\begin{align*}
\text{lt1} &= \text{true} \\
\text{lt2} &= \text{true} \\
\text{gtick} &= \text{false} \\
\end{align*}
\]

\[
\begin{align*}
! \text{gtick} \\
x &= x + 30 \\
\text{lt1} &= \text{false} \\

gtick \\
\text{lt1} &= \text{false} \\

! \text{gtick} \\
x &= x + 15 \\
\text{lt1} &= \text{true} \\

gtick \\
\text{lt1} &= \text{false} \\

! \text{gtick} \\
x &= x + 25 \\
\text{lt2} &= \text{true} \\

gtick \\
\text{lt2} &= \text{false} \\

\end{align*}
\]
Mapping TFSM to Timed Automata

\[ \text{lt1} = \text{true} \]

- \( ! \text{gtick} \)
- \( x = x + 15 \)
- \( \text{lt1} = \text{false} \)

\[ \text{gtick} \]

\[ \text{lt1} = \text{true} \]

\[ \text{lt2} = \text{true} \]

- \( ! \text{gtick} \)
- \( x = x + 20 \)
- \( \text{lt2} = \text{false} \)

\[ \text{gtick} \]

\[ \text{lt2} = \text{true} \]

- \( ! \text{gtick} \)
- \( x = x + 30 \)
- \( \text{lt1} = \text{false} \)

\[ \text{gtick} \]

\[ \text{lt1} = \text{false} \]

\[ \text{lt2} = \text{false} \]

\[ \text{lt1} \land \text{lt2} \]

\[ \neg \text{lt1} \land \neg \text{lt2} \]

\[ \text{WaitLT} \]

\[ \text{U} \]

\[ \text{GTReached} \]

\[ \text{gtick} = \text{true} \]

\[ x = 40 \]
Mapping TFSM to Timed Automata

\[ \text{lt1 = true} \]
\[ \text{lt2 = true} \]
\[ \text{gtick = true} \]

\[ x = x + 30 \]
\[ \text{lt1 = false} \]
\[ \text{gtick} \]

\[ x = x + 20 \]
\[ \text{lt2 = false} \]
\[ \text{gtick} \]

\[ x = x + 15 \]
\[ \text{lt1 = false} \]
\[ \text{gtick} \]

\[ x = x + 25 \]
\[ \text{lt2 = false} \]
\[ \text{gtick} \]

\[ \text{lt1 \land lt2} \]
\[ \text{gtick = true} \]
\[ \text{lt1 = true} \]
\[ \text{lt2 = true} \]

\[ \text{x = 40} \]

\[ \text{WaitLT} \]

\[ \text{U} \]

\[ \text{GTReached} \]
Mapping TFSM to Timed Automata

\[
\begin{align*}
\text{lt1} &= \text{false} \\
! \text{gtick} \\
x &= x + 30 \\
\text{lt1} &= \text{true} \\
\text{gtick} \\
\text{lt1} &= \text{false} \\
EOT1 \\
! \text{gtick} \\
x &= x + 15 \\
\text{lt1} &= \text{true} \\
\text{gtick} \\
\text{lt2} &= \text{false} \\
b01 \\
\text{gtick} \\
\text{lt2} &= \text{true} \\
b12 \\
EOT2
\end{align*}
\]

\[
\begin{align*}
\text{lt2} &= \text{true} \\
! \text{gtick} \\
x &= x + 20 \\
\text{lt2} &= \text{true} \\
\text{gtick} \\
\text{lt2} &= \text{false} \\
b01 \\
\text{gtick} \\
\text{lt2} &= \text{true} \\
b12 \\
EOT2
\end{align*}
\]

\[
\begin{align*}
\text{gtick} &= \text{true} \\
x &= 40 \\
\text{lt1} &\land \text{lt2} \\
!\text{lt1} &\land !\text{lt2} \\
\text{gtick} &= \text{true} \\
\text{WaitLT} \\
U \\
\text{GTReached} \\
\text{gtick} &= \text{false} \\
x &= 0 \\
\text{lt1} &= \text{false} \\
\text{lt2} &= \text{true} \\
\text{gtick} &= \text{true} \\
x &= 40
\end{align*}
\]
Mapping TFSM to Timed Automata

\[
\begin{align*}
\text{lt1} = \text{false} & \quad \text{lt2} = \text{true} & \quad \text{gtick} = \text{true} & \quad x = 40 \\
! \text{gtick} & \quad x = x + 30 & \quad \text{lt1} = \text{true} & \\
\text{gtick} & \quad \text{lt1} = \text{false} & \quad x = x + 20 & \quad \text{lt2} = \text{true} \\
! \text{gtick} & \quad x = x + 15 & \quad \text{lt1} = \text{true} & \\
\text{gtick} & \quad \text{lt2} = \text{false} & \quad x = x + 25 & \quad \text{lt2} = \text{true} \\
\end{align*}
\]
Mapping TFSM to Timed Automata

\( \text{lt1} = \text{false} \)

- \text{!gtick}
  - \text{x} = x + 30
- \text{lt1} = \text{true}

- \text{gtick}
  - \text{lt1} = \text{false}

- \text{EOT1}

- \text{!gtick}
  - \text{x} = x + 15
- \text{lt1} = \text{true}

- \text{gtick}
  - \text{lt1} = \text{false}

- \text{EOT2}

\( \text{lt2} = \text{false} \)

- \text{!gtick}
  - \text{x} = x + 20
- \text{lt2} = \text{true}

- \text{gtick}
  - \text{lt2} = \text{false}

- \text{EOT2}

\( \text{gtick} = \text{true} \quad x = 40 \)

- \text{lt1} \land \text{lt2}

- \text{lt1} \land \text{!lt2}

- \text{WaitLT}

- \text{U}

- \text{GTReached}

- \text{gtick} = \text{true}

- \text{lt1} \land \text{lt2}

- \text{gtick} = \text{false}

- \text{x} = 0

- \text{GTReached}

\text{Roop (University of Auckland)}

\text{PRET}

\text{CASES'09}
Mapping TFSM to Timed Automata

\[ \text{lt1} = \text{false} \]

\[ \text{lt2} = \text{false} \]

\[ \text{gtick} = \text{true} \quad x = 40 \]

\[ \text{lt1} \land \text{lt2} \]

\[ \text{UGTRenched} \]

\[ \text{b01} \]

\[ \text{EOT1} \]

\[ \text{b12} \]

\[ \text{EOT2} \]
Mapping TFSM to Timed Automata

\( \text{lt1} = \text{false} \)

- \( ! \text{gtick} \)
- \( x = x + 30 \)
- \( \text{lt1} = \text{true} \)

- \( \text{gtick} \)
- \( \text{lt1} = \text{false} \)

- \( \text{EOT1} \)

\( \text{lt2} = \text{false} \)

- \( ! \text{gtick} \)
- \( x = x + 20 \)
- \( \text{lt2} = \text{true} \)

- \( \text{gtick} \)
- \( \text{lt2} = \text{false} \)

- \( \text{EOT2} \)

\( \text{gtick} = \text{false} \quad x = 0 \)

- \( \text{lt1} \land \text{lt2} \)
- \( ! \text{lt1} \land ! \text{lt2} \)

- \( \text{WaitLT} \)

- \( \text{U} \)

- \( \text{GTReached} \)

- \( ! \text{lt1} \land ! \text{lt2} \)
- \( \text{gtick} = \text{false} \quad x = 0 \)

- \( \text{gtick} = \text{true} \)

Roop (University of Auckland)
Mapping TFSM to Timed Automata

\[ \text{lt1} = \text{false} \]

- \( \lnot \text{gtick} \)
- \( x = x + 15 \)
- \( \text{lt1} = \text{true} \)

- \( \text{gtick} \)
- \( \text{lt1} = \text{false} \)

- \( \lnot \text{gtick} \)
- \( x = x + 30 \)
- \( \text{lt1} = \text{true} \)

\[ \text{lt2} = \text{false} \]

- \( \lnot \text{gtick} \)
- \( x = x + 25 \)
- \( \text{lt2} = \text{true} \)

- \( \text{gtick} \)
- \( \text{lt2} = \text{false} \)

- \( \lnot \text{gtick} \)
- \( x = x + 20 \)
- \( \text{lt2} = \text{true} \)

\[ \text{gtick} = \text{false} \quad x = 0 \]

- \( \text{WaitLT} \)
- \( \text{lt1} \land \text{lt2} \)
- \( \text{gtick} = \text{true} \)

- \( \lnot \text{lt1} \land \lnot \text{lt2} \)
- \( \text{GTRReached} \)
- \( \text{gtick} = \text{false} \quad x = 0 \)
Complexity

We only need one integer variable ($x$), to estimate the cost of the global tick. Model checking a single query ($AG (gtick \Rightarrow x \leq val)$) is $O(|val| \times |M| \times |\phi|)$.

Since the value of $x$ ranges between $[W_{CRIT}^{\min}, W_{CRIT}^{\max}]$.

Overall Complexity $O(\log_2(W_{CRIT}^{\max} - W_{CRIT}^{\min}) \times (W_{CRIT}^{\max} - W_{CRIT}^{\min}) \times |M| \times |\phi|)$. 

Roop (University of Auckland)
We only need one integer variable ($x$), to estimate the cost of the global tick.
We only need one integer variable ($x$), to estimate the cost of the global tick.

Model checking a single query

$\text{(AG(gtick } \Rightarrow x \leq val))$ is $O(|val| \times |M| \times |\phi|)$
We only need one integer variable ($x$), to estimate the cost of the global tick.

**Model checking a single query**

$$(AG(gtick \Rightarrow x \leq val)) \text{ is } O(|val| \times |M| \times |\phi|)$$

Since the value of $x$ ranges between $[WCRT_{min}, WCRT_{max}]$.
We only need one integer variable \(x\), to estimate the cost of the global tick.

Model checking a single query

\[
(AG(gtick \Rightarrow x \leq val)) \text{ is } O(|val| \times |M| \times |\phi|)
\]

Since the value of \(x\) ranges between \([WCRT_{\text{min}}, WCRT_{\text{max}}]\)

Overall Complexity

\[
O(log_2(WCRT_{\text{max}} - WCRT_{\text{min}}) \times (WCRT_{\text{max}} - WCRT_{\text{min}}) \times |M| \times |\phi|).
\]
### Comparing $\text{WCRT}_{\text{max}}$ and the $\text{WCRT}_{\text{tight}}$

<table>
<thead>
<tr>
<th>Example</th>
<th>WCRT (Max Plus)</th>
<th>WCRT (Model Checker)</th>
<th>Gain %</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABRO</td>
<td>89</td>
<td>89</td>
<td>0</td>
</tr>
<tr>
<td>Channel Protocol</td>
<td>174</td>
<td>152</td>
<td>12.64</td>
</tr>
<tr>
<td>Reactor Control</td>
<td>121</td>
<td>118</td>
<td>2.47</td>
</tr>
<tr>
<td>Producer-Consumer</td>
<td>110</td>
<td>92</td>
<td>16.36</td>
</tr>
<tr>
<td>Smokers</td>
<td>531</td>
<td>449</td>
<td>15.44</td>
</tr>
<tr>
<td>Robot Sonar</td>
<td>419</td>
<td>346</td>
<td>17.42</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td><strong>10.72</strong></td>
</tr>
</tbody>
</table>

- Using Model Checker, we yields about 10% tighter value than the Max Plus approach.
### Execution Time

<table>
<thead>
<tr>
<th>Example</th>
<th>WCRT (Model Checker)</th>
<th>WCRT (Actual Execution)</th>
<th>Gain %</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABRO</td>
<td>89</td>
<td>87</td>
<td>97.75</td>
</tr>
<tr>
<td>Channel Protocol</td>
<td>152</td>
<td>149</td>
<td>98.03</td>
</tr>
<tr>
<td>Reactor Control</td>
<td>118</td>
<td>114</td>
<td>96.61</td>
</tr>
<tr>
<td>Producer-Consumer</td>
<td>92</td>
<td>88</td>
<td>95.65</td>
</tr>
<tr>
<td>Smokers</td>
<td>449</td>
<td>430</td>
<td>95.77</td>
</tr>
<tr>
<td>Robot Sonar</td>
<td>365</td>
<td>339</td>
<td>97.40</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td><strong>96.87</strong></td>
</tr>
</tbody>
</table>

- On an average, the actual value is approximately 96% of the value obtained from UPPAAL.
Conclusions

Contributions
Conclusions

Contributions

- First model checking based static analysis tool for synchronous programs developed.
Conclusions

Contributions

- First model checking based static analysis tool for synchronous programs developed.
- Takes both state-dependencies and data-dependencies into account while computing the tick length.
Conclusions

Contributions

- First model checking based static analysis tool for synchronous programs developed.
- Takes both state-dependencies and data-dependencies into account while computing the tick length.
- Computed values are very close to the measured WCRT values.
Future Work

Tighter

- Use of abstraction-refinement to deal with the environment non-determinism.
Future Work

Tighter

- Use of abstraction-refinement to deal with the environment non-determinism.

Extend

- Analysis of pure Software Implementation.
- Model any processor (with speculative features), for timing analysis over GPPs.
- Model the memory hierarchy.

Comparisons

ILP formulation [Juet al., CODES+ISSS'08].

Roop (University of Auckland)
Future Work

Tighter

- Use of abstraction-refinement to deal with the environment non-determinism.

Extend

- Analysis of pure Software Implementation.
- Model any processor (with speculative features), for timing analysis over GPPs.
- Model the memory hierarchy.

Comparisons

- ILP formulation [Ju et al., CODES+ISSS’08].
More Information..
Hardware Support

Thread Table
- PC (32bits)
- TDA (1bit)
- TSP (1bit)
- TLT (1bit)
- PID \( \log_2(N) \) bits
- ALC \( \log_2(M) \) bits
- SC (32bits)

WCRT Timer

Controller Logic

Abort Table
- Valid (1bit)
- TID \( \log_2(N) \) bits
- PVI \( \log_2(X) \) bits
- WS (1bit)
- PA (32bits)
- AL \( \log_2(M) \) bits
- PV (X bits)

Scheduler

Control and data from MB

Control and data to MB