Chapter 2

Worst-Case Execution Time Analysis
Objectives

In this chapter, you are supposed to learn:

- What is WCET, and why WCET
- How to obtain the WCET of a program
- Static analysis methods and measurement-based methods
- Practices on WCET analysis of RTOS
- New challenges and future trends on WCET analysis
Contents

- An Introduction to WCET Analysis
- Static Analysis
- Measurement-Based Methods
- WCET Analysis of RTOS
- New Challenges and Future Trends
- Recommended Readings
An Example of Distributed RTS
The Need for Timing Validation

- An Example in Car Industry
  - Today, a new car typically contains 80 ECUs
  - The car electronic systems are provided by multiple OEMs
  - The challenge of integration
  - Increasingly complex processors are used

- Related reports show that
  - Other Electronic Problems: 25%
  - Timing Problems: 30%
  - Other Problems: 45%
Void signal_processing (){ 
    curr_signal = read_signal();
    if (curr_signal < threshold){
        signal_transformation();    // some +-* ops.
    }
    else{
        error_handling_routine();
        // complex error handling operations
    }
}

In this signal processing task, the real operations performed depends on the inputted signals. Different signals lead to different operations, then different execution time.

Almost all real-life programs have variable execution time.
What is WCET?

Worst-case execution time of a task is NOT response time of a task, the latter contains not only execution time, but also the durations of preemptions and blockings.
Why WCET Analysis?

- Hard real-time systems must satisfy stringent timing constraints; whether the constraints are satisfied or not should be analyzed at design time.
- Real-time schedulability test requires WCET of each task, and an incorrect result leads to timing failure.
- On the right is an example of the result led by incorrectly estimated WCET.
WCET Analysis Quality

- **Safety:**
  - The estimated upper bound should always enclose the actual WCET

- **Tightness:**
  - The estimated upper bound should be as close as possible to the actual WCET

- **Complexity:**
  - There is a trade-off between accuracy and analysis complexity
  - Analyzers should balance it according to practical requirements
  - The trade-off between analysis complexity and the quality of results
Why Not Just Measure WCET?

Start Timing Measurement

Timer
Logic Analyzer
...

Execute Tasks on Target HW

Stop Timing Measurement

WCET Estimation?
Why Not Just Measure WCET?

Why NOT?
- It is *intractable* to cover all execution traces of a program (Think of a program with 10,000 loop iterations and an if-then-else as the loop body, $2^{10,000}$ traces)
- Hard to guarantee worst-case data input
- hard to simulate worst-case processor state
- Need real hardware

BUT
- Measurement-based methods are easy to implement
- Can get a rough estimation of the execution time
- Compliment with other analysis techniques to make the results trustworthy
Static Analysis Techniques

- **How it works?**
  - Given a program executable and the hardware the program is running, use mathematical methods to calculate the safe upper bound without any simulation.

- **Pros**
  - Math theorems guarantee safety.
  - So mandatory in safe-critical hard real-time systems.

- **Cons**
  - Need to build complex mathematical models.
  - Long analysis time for complex programs.
The Ingredients of WCET Analysis

- **Flow Facts**
  - Flow facts give us information on the control flow of the programs, such as infeasible paths and loop counts, etc.
  - Automatic flow facts extraction and manual annotation
  - How to annotate flow facts in the program
The Ingredients of WCET Analysis

The Representation Levels of Programs

Matlab/Simulink – Component-based Design

C/C++/Java – High-Level Language

Assembly or Machine Code

```
int filtez(int *bpl, int *dlt)
{
    int i;
    long int zl;
    zl = (long)(*bpl++)*(*dlt++);
    for (i = 1; i < 6; i++)
        zl+=(long)(*bpl++)*(*dlt++);
    return ((int) (zl >> 14)); /* x2 here */
}
```
The Ingredients of WCET Analysis

- The Target Hardware

<table>
<thead>
<tr>
<th>Simple Single-Core Processor</th>
<th>Complex Processors with Pipeline and Cache</th>
<th>Multi-core Processors</th>
</tr>
</thead>
</table>

Hardware is becoming more and more complex, hard to analyze!
Remarks on the Ingredients

- The Representation Levels of Programs
  - Precise timing analysis has to be done after all program transformations
  - Generally, it is much easier to extract or annotate flow facts in a higher representation level
  - The flow facts should be mapped from higher level to lower level correctly, probably this mapping is done in parallel to code transformation

- Hardware in real-time systems are becoming more and more complex with features to improve average-case performance (throughput), but less predictable, e.g. timing anomaly
Contents

- An Introduction to WCET Analysis
- Static Analysis
  - Path Analysis
  - Micro-Architecture Analysis
- Measurement-Based Methods
- WCET Analysis of RTOS
- New Challenges and Future Trends
- Recommended Readings
A Generic Workflow of Static Analysis

1. Compile source code into the binary of target hardware
2. Reconstruct the Control-Flow Graph from the binary
3. Model the HW architecture, calculate the execution time of each basic block in the CFG
4. Calculate the WCET using some DSE tools, e.g. ILP solvers, constraint solvers, model checkers

Figure from Chronos@NUS
An Example of the Workflow

/* p >= 0 */
q = p;
while(q<10)
  q++;
r = q;

Do micro-arch modeling to get the execution time of each BB

\[
\text{wcet} = \max \sum_{t_{k,i} \in T_g}(\text{cost}(t_{k,i}) \cdot x_{k,i}) \text{ s.t. flow facts (1)}
\]

\[
\forall BB_i, b_i = \sum_{\text{dst}(t_{k,i}) = BB_i} x_{k,i} = \sum_{\text{src}(t_{i,j}) = BB_i} x_{i,j} \text{ (2)}
\]

\[
\forall \text{Loop}_i, b_{\text{Tail}_i} \leq lp_{b_i} \cdot \sum_{BB_j \in BL_i} b_j
\]
What is Path Analysis?

- **Path Analysis**
  - To identify the execution trace that leads to the longest execution time
  - To identify infeasible paths of the program
  - Path analysis is a “Design Space Exploration” problem

- **Popular Techniques**
  - Tree-based methods (Timing Schema)
  - Path-based methods
  - Implicit Path Enumeration Technique (IPET)
Timing Schema

- Represent the program in a syntax tree
- Calculate the WCET of a program by folding the tree

Transformation rules:

- $T(\text{seq}(S1, S2)) = T(S1) + T(S2)$
- $T(\text{if}(\text{Exp}) \ S1 \ \text{else} \ S2) = T(\text{Exp}) + \max(T(S1), T(S2))$
- $T(\text{loop}(\text{Exp, Body})) = T(\text{Exp}) + (T(\text{Exp}) + T(\text{Body})) \times (\text{maxIter-1})$
Timing Schema

- Some General Assumptions
  - No recursion
  - Explicit function calls
  - No “goto”s
  - Bounded loop with single entry and single exit

- The Rules

<table>
<thead>
<tr>
<th>construct</th>
<th>MAXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>primitive</td>
<td>maxt(primitive) = ( \tau(\text{primitive}) )</td>
</tr>
<tr>
<td>sequence</td>
<td>maxt(sequence) = ( \sum_i \text{maxt}(\text{construct}_i) )</td>
</tr>
<tr>
<td>alternative</td>
<td>maxt(alternative) = ( \text{maxt}(\text{condition}) + \max(\text{maxt}(\text{construct}_1), \text{maxt}(\text{construct}_2)) )</td>
</tr>
</tbody>
</table>
| loop. number   | maxt(loophead) = \( \text{maxt}(\text{init}) + \text{maxt}(\text{condition}) + \)  
                  | \( \text{count} \times (\text{maxt}(\text{body}) + \text{maxt}(\text{condition})) + \)  
                  | \( \text{maxt}(\text{overrun} \_ \text{statement}) \)                      |
|                | maxt(looptail) = \( \text{count} \times (\text{maxt}(\text{body}) + \text{maxt}(\text{condition})) + \)  
                           | \( \text{maxt}(\text{overrun} \_ \text{statement}) \)                    |
| loop.time      | maxt(looptime) = \( \text{time} + \text{maxt}(\text{timeout} \_ \text{statement}) \) |
| subroutine     | maxt(subroutine) = \( \tau(\text{organization}) + \text{maxt}(\text{body}) \) |
An Example

```c
int calc_center(image, x_center, y_center) /* 44 */
{ char image[MAX_ROWS][MAX_COLS];
  int *x_center, *y_center;
  
  int pixel_count, x_coord, y_coord, x_sum, y_sum; /* 48 */
  
  pixel_count = x_sum = y_sum = 0;

  FOR(y_coord = 0; y_coord < MAX_ROWS; y_coord++) SCOPE MAX_COUNT(MAX_ROWS) /* (42+26;78;32) */
  { /* loop3 */
    FOR(x_coord = 0; x_coord < MAX_COLS; x_coord++) MAX_COUNT(MAX_COLS) /* (42+26;78;32) */
    { /* loop4 */
      if (image[x_coord][y_coord]) /* alt2 */ /* 146 */
      {
        int weight;

        MAX_COUNT(MAX_AREA); /* marker */ /* 40 */
        weight = calc_weight(image, x_coord, y_coord); /* 310 */
        x_sum += x_coord * weight;
        y_sum += y_coord * weight;
        pixel_count += weight;
      }
    }
  }

  if (pixel_count) /* alt3 */ /* 22 */
  {
    *x_center = x_sum / pixel_count; /* 424 */
    *y_center = y_sum / pixel_count;
  }
  else
    *x_center = *y_center = 0; /* 56 */

  return 1; /* 14 */
}
```
An Example (2)

\[
\begin{align*}
\text{maxt}(\text{calc\_center}) & = 44 + 48 + \text{maxt}(\text{loop}_3) + \text{maxt}(\text{alt}_3) + 14 = \\
& = 551\,475\,096 \\
\text{maxt}(\text{loop}_3) & = 42 + 76 + 200 \ast ((\text{maxt}(\text{loop}_4) + 32) + 76) + 26 = \\
& = 551\,474\,544 \\
\text{maxt}(\text{loop}_4) & = 42 + 76 + 640 \ast ((\text{maxt}(\text{alt}_2) + 32) + 76) + 26 = \\
& = 2\,757\,264 \\
\text{maxt}(\text{alt}_2) & = 146 + \max(310 + \text{maxt}(\text{calc\_weight}), 0) = 4\,200 \\
\text{maxt}(\text{alt}_3) & = 22 + \max(424, 56) = 446
\end{align*}
\]

\[
\begin{align*}
\text{maxt}(\text{calc\_weight}) & = 44 + 72 + \text{maxt}(\text{loop}_1) + 122 = 3\,744 \\
\text{maxt}(\text{loop}_1) & = 54 + 84 + 3 \ast ((\text{maxt}(\text{loop}_2) + 32) + 84) + 26 = 3\,506 \\
\text{maxt}(\text{loop}_2) & = 54 + 84 + 3 \ast ((\text{maxt}(\text{alt}_1) + 32) + 84) + 26 = 998 \\
\text{maxt}(\text{alt}_1) & = 146 + \max(16, 0) = 162
\end{align*}
\]
The Workflow of Timing Schema

- **Decomposition**
  - Decompose a statement into its primitive components (atomic blocks)

- **Code Prediction**
  - Predict the implementation (compiled instructions) of each atomic block

- **Execution Time of the Atomic Blocks**
  - Calculate the execution times of the atomic blocks according to the execution times of the instructions

- **Execution Time of the Statements**
  - Calculate the execution times of the statements according to the execution times of the atomic blocks
An Evaluation of Timing Schema

- **Pros**
  - Simple method with cheap computation effort
  - Scale very well with program size

- **Cons**
  - Cannot deal with generic flexible program structures
  - Limited ability on specifying flow facts
  - Suffers compiler optimization
Path-Based Methods

- The upper bound is determined by: first calculating the bounds of all paths, and then searching the path with longest execution time.

- Possible paths are represented explicitly.

```
// Unit timing
\( t_{\text{path}} = 31 \)
\( t_{\text{header}} = 3 \)

// WCET Calc
WCET =
\( t_{\text{header}} + t_{\text{path}} \times (\text{maxiter}-1) = 3 + 31 \times 99 = 3072 \)
```
Model Checking

- Model Checking of WCET is Path Based
  - The state space is all the possible program paths
  - The model checkers deal with paths explicitly

- Basic Idea
  - Construct the CFG of a program as input
  - Transform the CFG into the MC model
  - Search the path with the longest execution time
CFG Reconstruction – An Example

```c
void main()
{
    int b;
    int i = 0, j = 0;
    while (i < 10){
        if (b)
            j++;
        else
            j--;
        i++;
    }
}
```

(a) A motivating example  (b) The CFG
The model checker runs an FSM, where each box represents a state in the FSM, and the arcs represent the transitions. Labels on arcs specify the transition conditions.
The Optimization Procedure

- We can ask the model checker “is it YES or NO that ‘for all execution paths starting from the initial state, globally WCET is not greater than N’.

- Additional procedures are needed to find the actual value of N

```
Algorithm 1 Finding the WCET using binary search

input: The model M of a model checker, initial value of N
output: The optimal value found

set the upper and lower bound of binary search
while (lower bound < upper bound - 1)
    middle = (lower bound + upper bound) / 2;
    check the property [] φ(middle)
    if ( [] φ(middle) is satisfied)
        upper bound = middle;
    else
        lower bound = middle;
end while
return upper bound
```

For example,
- If the actual WCET is 100, then TRUE, for N = 100
- FALSE, for N = 99
Evaluation of the Path-Based Methods

- **Pros**
  - Allows simple integration of HW modeling in the analysis (expressiveness)
  - Guaranteed exact results

- **Cons**
  - Scalability problems (exponential state space)
  - If you use model checkers, some unknown performance bottlenecks may occur
Implicit Path Enumeration Technique

- Can obtain exact answer without exhaustive search of all the paths

- Hint: the objective is to determine the worst-case execution time, not the worst-case execution path

- Idea: finding the worst-case execution time $\rightarrow$ finding the worst-case execution count of each basic block
Implicit Path Enumeration Technique

Solutions

- The problem of finding the worst-case execution counts can be formulated as an **Integer Linear Programming (ILP)** problem or a constraint programming problem.
- The more constraints, the more accurate results.

\[
\begin{align*}
    t_{\text{max}} &= \max (\sum_i c_i x_i)
    \\
    \text{execution time of basic block } B_i &\quad \text{(constant)}
    \\
    \text{execution count of basic block } B_i &\quad \text{(variable)}
\end{align*}
\]

subject to a set of constraints:

\[
Ax \leq B
\]
Implicit Path Enumeration Technique

- Constraints – Restrictions on x-variables
  - Structural constraints: extracted directly from the CFG

If statement

(a) Source code

if (p)
  q = 1;
else
  q = 2;
  r = q;

(b) Control flow graph

At each node:

- Basic block count = Σ inputs
  = Σ outputs

Structural Constraints:

- $x_1 = d_1 = d_2 + d_3$
- $x_2 = d_2 = d_4$
- $x_3 = d_3 = d_5$
- $x_4 = d_4 + d_5 = d_6$

Linear constraints
Implicit Path Enumeration Technique

- Constraints – Restrictions on x-variables
  - Functional constraints: telling how the program works, e.g. how many times a loop iterates

```c
int check_data()
{
    int i, morecheck, wrongone;
    morecheck=1; i=0; wrongone= -1;
    while(morecheck) {
        x1
        if (data[i] < 0)
            {wrongone=i; morecheck=0;}
        else
            x3
            if (++i >= datalize)
            x5
                morecheck=0;
            x6
        }
    if (wrongone >= 0)
        x7
        return 0;
    else
        x8
        return 1;
}
```

- Loop bound information (mandatory) (loop 1-10 times)
  \[ x_1 \leq x_2 \leq 10 \]

- Basic blocks \( B_3 \) and \( B_5 \) are mutually exclusive and either one of them is executed once.
  \[ (x_3 = 0 \& x_5 = 1) \text{ or } (x_3 = 1 \& x_5 = 0) \]

- Basic blocks \( B_5 \) and \( B_9 \) are executed together.
  \[ (x_5 \geq 1 \& x_9 \geq 1) \text{ or } (x_5 = 0 \& x_9 = 0) \]

*Sets of linear constraints*
An Example of ILP Formulation

Maximize
\[ 11 \ dSta_0 + 5 \ d0_1 + 1 \ d1_2 + 3 \ d1_3 \]
\[ + 5 \ d2_7 + 7 \ d3_4 + 5 \ d3_5 + 7 \ d4_6 + 8 \ d5_6 + 4 \ d6_1 \]

Subject to
\[
\text{\"==tcfg constraints\"}
\]
\[ dSta_0 = 1 \]
\[ b0 - d0_1 = 0 \]
\[ b0 - dSta_0 = 0 \]
\[ b1 - d1_2 - d1_3 = 0 \]
\[ b1 - d0_1 - d6_1 = 0 \]
\[ b2 - d2_7 = 0 \]
\[ b2 - d1_2 = 0 \]
\[ b3 - d3_4 - d3_5 = 0 \]
\[ b3 - d1_3 = 0 \]
\[ b4 - d4_6 = 0 \]
\[ b4 - d3_4 = 0 \]
\[ b5 - d5_6 = 0 \]
\[ b5 - d3_5 = 0 \]
\[ b6 - d6_1 = 0 \]
\[ b6 - d4_6 - d5_6 = 0 \]
\[ b7 - d2_7 = 0 \]
\[ b0 \leq 1 \]
\[ b7 = 1 \]
\[ b6 - 10 \ b0 \leq 0 \]

// Definition of integers is omitted
An Evaluation of IPET

Pros

- Allows to consider complex flow facts
- Generation of constraints is simple and direct
- Efficient tools

Cons

- Solving ILP is generally NP-hard (luckily, the WCET problem can be reduced to network flow problem, which requires less solving time)
- Still difficult to encode the flow facts that specify execution ordering
Contents

- An Introduction to WCET Analysis
- Static Analysis
  - Path Analysis
  - Micro-Architecture Analysis
- Measurement-Based Methods
- WCET Analysis of RTOS
- New Challenges and Future Trends
- Recommended Readings
Micro-Architecture Analysis

- Why Micro-Architecture Analysis?
  - The execution time depends not only on the program itself, but also on the hardware where the program executes.
  - Modern processors have lots of complex features that can result in unpredictable execution time variation, which is very hard to analyze.
  - Timing Anomaly

- What Are Included in Micro-Architecture Analysis?
  - Cache analysis
  - Pipeline analysis (multiple issue, out-of-order pipelines)
  - Branch prediction and speculative execution
  - ……
Cache in a Nutshell

- Why Cache?
  - The “memory wall”

```
Relative Performance

10000

1000

100

10

1


CPU Frequency
DRAM Speeds

CPU -- 2x Every 2 Years
DRAM -- 2x Every 6 Years

Gap
```
Cache in a Nutshell

- Why Cache?
  - Cost-speed trade-off
  - Program temporal/spatial locality
  - Memory hierarchy
Cache in a Nutshell

- Types of Caches
  - L1 Instruction Cache (32KB)
  - L1 Data Cache (32KB)
  - L2/L3 Unified Cache (512KB ~ 6MB)
  - Shared cache in multicores

- Associativity
  - Cache are organized in terms of “cache lines”
  - Associativity specifies how the cache lines are organized and how to map a memory block into the cache
  - Direct-mapped
  - Full-associative
  - Set-associative
Cache in a Nutshell

- Direct-mapped Cache

\[
i = x \% n;
\]

Easy to implement

Fast scan

But high miss ratio!
Cache in a Nutshell

- **Full-associative Cache**

A memory block can be mapped to any cache line if not occupied.

Efficient use of the cache

But notorious scan and replacement overhead!
Cache in a Nutshell

- **Set-associative Cache**

  \[ i = (x \mod \#\text{sets}) + A \quad (0 \leq A \leq \text{set size}) \]

  A clever trade-off between direct-mapped caches and full-associative caches

  Much less overhead than FA, but still harder to analyze than DM

  Good news to GP-architecture guys, but not so good to Real-Time guys
Cache in a Nutshell

- **Replacement Policy**
  - If cache miss occurs, kick out which cache line?
  - Round-robin, LRU, pseudo-LRU
  - Different cache replace policies have different predictability

- **Write Policy**
  - Write-through: whenever there is a write to the cache content, the data is immediately written to the corresponding main memory address, regardless of hits or misses
  - Write-back: only write dirty cache data to main memory when the cache block is replaced, requires special bits in cache to tag dirty data
Cache Analysis in WCET Analysis

- Without cache analysis
  - In each BB, all memory accesses take fixed cycles, no variation
  - The execution time of a BB is not affected by the execution history
  - When there is cache, all the situations are different

- Analysis of different types of caches
  - I-cache with different replacement policy
  - I-cache or D-cache?
  - Single-level or multi-level?
  - Dedicated cache or shared cache?
Cache Analysis in the IPET Framework

- **Idea**
  - Model new constraints related to cache behavior into the original ILP problem
  - No fundamental changes to the structure of the ILP problem

- **How to?**
  - For each instruction, determine
    - Cache hit execution counts, time
    - Cache miss execution counts, time
    - go into the basic blocks
The objective cache analysis is to determine how many misses and hits in each BB. Analyze conflicting memory blocks.
Modified ILP Formulation

Let:

- $x_{i,j}^{hit}$ – cache hit count of l-block $B_{i,j}$
- $x_{i,j}^{miss}$ – cache miss count of l-block $B_{i,j}$
- $c_{i,j}^{hit}$ – exec. time of l-block $B_{i,j}$ given that it is a cache hit
- $c_{i,j}^{miss}$ – exec. time of l-block $B_{i,j}$ given that it is a cache miss
- $n_i$ – number of l-blocks of basic block $B_i$

Maximize:

$$\sum_{i} \sum_{j} N n_i \left( c_{i,j}^{hit} x_{i,j}^{hit} + c_{i,j}^{miss} x_{i,j}^{miss} \right)$$

subject to:

- $x_i = x_{i,j}^{hit} + x_{i,j}^{miss}$ for $j = 1, 2, \ldots, n_i$
- structural constraints
- functionality constraints
- cache constraints
New Cache Constraints

- only one l-block Bk.1 maps to the same cache line (first access is miss):
  \[ x_{k,l}^{\text{miss}} \leq 1 \]

- only two or more non-conflicting l-blocks map to the same cache line (first access is miss):
  \[ x_{k,l}^{\text{miss}} + x_{m,n}^{\text{miss}} \leq 1 \]

- two or more conflicting l-blocks \( \Rightarrow \) use CCG
Cache Conflict Graph (CCG)

Control Flow Graph (CFG)  
 Cache Conflict Graph (CCG)

start

\[ B_k \]

\[ B_{k,l} \]

\[ B_m \]

\[ B_{m,n} \]

end

\[ s \]

\[ e \]
Generating Constraints from CCG

Flow at node $B_{k,l}$:
\[ x_k = p_{(s,k,l)} + p_{(m,n,k,l)} + p_{(k,l,k,l)} = p_{(k,l,e)} + p_{(k,l,m,n)} + p_{(k,l,k,l)} \]

Cache hit count for $l$-block $B_{k,l}$:
\[ p_{(k,l,k,l)} \leq x_{k,l}^{hit} \leq p_{(s,k,l)} + p_{(k,l,k,l)} \]

Starting Condition:
\[ p_{(s,k,l)} + p_{(s,m,n)} + p_{(s,e)} = 1 \]
Tightening the Constraints

Assumptions for the Example
- Each BB is mapped to a single cache line
- BB1 conflicts with BB6, BB4 conflicts with BB5

```c
/* k >= 0 */
s = k;
while (k < 10) {
    if (ok)
        j++;
    else {
        j = 0;
        ok = true;
    }
    k++;
}  
r = j;
```

(i) Source code

```
\begin{figure}
\centering
\begin{tikzpicture}

\node (x1) at (0,0) [state] {B1 \textit{s = k;}};
\node (x2) at (1,0) [state] {B2 \textit{while(k<10)}};
\node (x3) at (2,0) [state] {B3 \textit{if(ok)}};
\node (x4) at (3,0) [state] {B4 \textit{j++;}};
\node (x5) at (4,0) [state] {B5 \textit{j = 0; ok=true;}};
\node (x6) at (5,0) [state] {B6 \textit{k++;}};
\node (x7) at (6,0) [state] {B7 \textit{x = j;}};
\node (x8) at (7,0) [state] {e};

\path[->]
(x1) edge node {$d_1$} (x2)
(x2) edge node {$d_2$} (x3)
(x3) edge node {$d_3$} (x4)
(x4) edge node {$d_4$} (x3)
(x4) edge node {$d_5$} (x5)
(x5) edge node {$d_6$} (x4)
(x5) edge node {$d_7$} (x6)
(x6) edge node {$d_8$} (x5)
(x6) edge node {$d_{10}$} (x7)
(x7) edge [red] node {$p_{(4,1)}$} (x8)
(x8) edge [red] node {$p_{(5,1)}$} (x8)
(x8) edge node {$p_{(4,1,5,1)}$} (x8)
(x8) edge node {$p_{(5,1,5,1)}$} (x8);

\end{tikzpicture}
\end{figure}
```

(ii) Control flow graph (CFG)

\(p_{(4,1,5,1)} = 0\)
Tightening the Constraints

We already know:

But this needs to be tightened:

\[ x_3 = 10 \times 1 \]
\[ x_7 = 10 \times x_5 \]
\[ x_4 = 9 \times 1 \]
Inter-Procedure Calls

- $d_1 = 1$, $x_1 = d_1 = f_1$, $x_2 = f_1 = f_2$, $d_2.f_1 = f_1$
- $x_3.f_1 = d_2.f_1 = d_3.f_1$, $d_2.f_2 = f_2$
- $x_3.f_2 = d_2.f_2 = d_3.f_2$, $x_3 = x_3.f_1 + x_3.f_2$
- $X^{hit}3.1 = p(3.1.f_1, 3.1.f_2)$

void main()
{
    B_1 inc(&i);
    B_2 inc(&j);
}

void inc(int *pi)
{
    B_3 *pi++;
}

(i) Code fragment  (ii) CFG with 2 instances of function inc  (iii) CCG
Direct-Mapped $\rightarrow$ Set-Associative

- What’s the Difference?
  - Since conflicting domains are set-associative sets, there are more potential conflicts to be analyzed
  - Cache replacement policy affects analysis

- What to do?
  - We need to maintain cache states
    $$ \sum_{i=0}^{n} \frac{m!}{(m-i)!} $$
    
    - $[X,X], [X,B_{i,j}], [X,B_{k,i}], [X,B_{m,n}], [B_{i,j},B_{k,i}], [B_{i,j},B_{m,n}], [B_{k,i},B_{m,n}], [B_{m,n},B_{i,j}]$ and $[B_{m,n},B_{k,i}]$.
  - CCG $\rightarrow$ CSTG (a more concrete form of CCG)
  - Cost function is unchanged, but cache constraints are different now
Cache State Transition Graph

```
\text{CFG}

\text{CCG}

\text{Cache state transition graph}
```
New Cache Constraints

1. The execution count of $B_{m,n}$ = the sum of inflow with $B_{m,n}$ in the right most line entry
2. For each node, sum of inflow = sum of outflow
3. Starting condition
4. Cache hit lower bound:

$$x_{hit}^{y,z} \geq \sum_{u,v} P([u,v,y,z], [u,v,y,z]) + \sum_{u,v} P([y,z,u,v], [u,v,y,z])$$
Data Cache Analysis

- Two sub-problems
  - Determine load/store addresses
  - Model worst case data cache hit/miss counts

- Difficulties
  - L/S addresses may be ambiguous or may change, usually dynamic data structures are banned for static analysis
  - Data flow analysis is required

- Solutions
  - Extend cost functions to include data cache miss penalties
  - Use linear constraints to solve address ambiguity problems
Two-Level Analysis

- Data flow analysis
  - To determine the absolute data addresses of LD/ST instructions
  - Very difficult, but algorithms already established

- Data cache conflict analysis
  - Given the results of data flow analysis, construct a data cache conflict graph, and use ILP techniques to bound the data cache hit and miss counts

- Cinderella works on the second sub-problem
Modified Cost Functions

\[ x_i = m_{addr}^{hit} + m_{addr}^{miss} \]

\[
\text{Exec. time} = \sum_i \sum_j (c_{i,j}^{hit} x_{i,j}^{hit} + c_{i,j}^{miss} x_{i,j}^{miss}) \\
+ \sum_{addr} (c_{addr}^{hit} m_{addr}^{hit} + c_{addr}^{miss} m_{addr}^{miss})
\]
Data Cache Conflict Graph

- **Idea**
  - By data flow analysis, we can identify a set of possible data addresses accessed by LD/ST instr.
  
  - Different LD/ST instructions that access the addresses in the same data cache set may lead to data cache misses.
  
  - Similar to I-cache analysis, use data cache conflict graph to capture the control flow of LD/ST instructions to analyze potential data hits and misses.
Assume data cache is direct-mapped, and each cache line has 4 bytes.

Data address range [0x100, 0x124] span 10 data cache lines.

Take the set at 0x100 for example, see the graph on the left.

Data Cache Conflict Graph

Assume data cache is direct-mapped, and each cache line has 4 bytes.

Data address range [0x100, 0x124] span 10 data cache lines.

Take the set at 0x100 for example, see the graph on the left.
New Constraints

- In D-CCG, sum of inflow = sum of outflow
  \[ m_{0x014.0x100} = p_1 = p_2 \]
  \[ m_{0x01c.0x100} = p_2 = p_3 \]
  \[ p_1 = 1 \]

- The bounds on the execution counts of each LD/ST instruction instance
  \[ \sum_{addr_j} m_{0x014.addr_j} = x_2 \]
  \[ \sum_{addr_j} m_{0x01c.addr_j} = x_2 \]

- Hit and miss relation
  - LD-incurred cache miss is similar to instruction cache
  - ST-incurred cache miss depends on write policies: write through or write back, with/without write allocate
An Evaluation of the Above Analysis

- **Pros**
  - An elegant way to integrate hardware modeling into WCET calculation

- **Cons**
  - The number of ILP constraints grows greatly, because the CCG is a fine-grained representation of cache states
  - So the time to solve the ILP problem may be very long, not feasible for real-life programs

- **Solutions**
  - Try some other methods that can do cache analysis in a more coarse-grained way by sacrificing some precision
Timing Anomaly

- **Counterintuitive Behaviors**

Latency of instruction A varies by $\Delta t = -7$ cycles.

<table>
<thead>
<tr>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
</tbody>
</table>

Instruction A Cache Miss

Instruction A Cache Hit
Timing Anomaly

- A Formal Definition
  - $\Delta t$ – Latency variations of several instructions $S'$ (the whole instruction sequence is $S$)
  - $\Delta C$ – execution time change of the whole instruction sequence

As long as one of the following conditions hold, we say that a timing anomaly occurs

- $\Delta t > 0 \rightarrow \Delta C < 0$
- $\Delta t < 0 \rightarrow \Delta C > 0$
- $\Delta t > 0 \rightarrow \Delta C > \Delta t$
- $\Delta t < 0 \rightarrow \Delta C < \Delta t$
Domino Effect

Instructions

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ADD</td>
<td>r4, r3, r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>SW</td>
<td>r4, 0x0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>MUL</td>
<td>r10, r4, r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>LW</td>
<td>r3, 0x8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>ADD</td>
<td>r11, r10, r10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

extra delay of 1 cycle each iteration !!!

First instruction One cycle delayed

Initially empty Pipeline

in-order resource

out-of-order resource
Possible Solutions

- Occurrence of timing anomalies depends on both hardware features and code structure
- How to eliminate timing anomalies?
  - De-active caches
  - Use synchronization points
  - Choose more predictable hardware platform
  - Code reordering
Contents

- An Introduction to WCET Analysis
- Static Analysis
- Measurement-Based Methods
- WCET Analysis of RTOS
- New Challenges and Future Trends
- Recommended Readings
A Review of Problems of Static Analysis

- **Problems of Static Analysis**
  - Computation efforts exerted to cover all possible situations → possible scalability problems
  - Hard to conduct micro-architecture models
  - Micro-arch analysis of complex hardware may encounter scalability problems

- **So Measurement-Based Methods**
  - What can we benefit from it?
  - How to do measurement-based analysis?
  - What are the technical issues?
Measurement-Based Methods – The Big Picture

Analyzer tool

C-Source

Analysis phase

Execution time measurement framework

Measurement phase

Calculation tool

Calculation phase

WCET bound

Chapter 2: WCET Analysis
Tool Architecture
Issues in Measurement-Based Methods

- **How to measure?**
  - Measurement tools: HW, SW
  - End-to-end, or just measure code segments?

- **How to cover more execution traces?**
  - Due to worst-case input
  - Due to worst-case hardware states
  - Path/Trace coverage

- **What do the results reveal?**
  - Single WCET value, or a ET distribution?
  - This issue equals “what’s the use of measurement-based methods?”
How to Measure?

- End-to-end or measuring code segments?
  - End-to-end is easy, but inaccurate, intractable
  - Measurement of code segments + Calculation

- How to Measure?
  - Software instrumentation
    - Put time recording in the analyzed codes
    - Accuracy?
  - Hardware instrumentation
    - Logic Analyzers, oscilloscopes, …
Hardware Instrumentation

System Under Test

Code

Hardware

Hardware Interfaces
- Simple I/O ports
- Address lines
- Debug interfaces
- Communication devices

Instrumentation Interface

Execution Time Measurement System

FST: set 2
dcall df
ldab #1
ldab OFST-1,s
bitb #15
bne L22
ldab #1
stab L5r
L22:
leas 2,s
rti

quit:
Execution Time Measurement Framework

C-Code -> Analyzer Tool
*.testdata.xml
*.target_ps002.c
*.target_ps001.c

MF Tool
Target make
target.bin

Link
Target Hardware
USB
2 lines

Runtime Measurement Device
*.etime.xml

Host PC
Instrumentation Methods

- **Requirements**
  - Instrumentations (IPs) may not alter program flow or execution time in an unknown or unpredictable way. IPs have to be persistent if changing either.
  - Execution always starts with the same (known) state (cache, pipeline, branch prediction, ...)

- **Design Decisions**
  - Control flow manipulation? Input data generation?
  - Number of measurement runs?
  - Resource consumption?
  - Required devices?
  - Installation effort?
The Steps of Measurement-Based Analysis

1. Static analysis: reconstruct CFG from the code
2. Program partitioning
3. Test data generation
4. Execution time instrumentation
5. WCET calculation

- This is only one exemplary workflow, other measurement-based methods may have different workflow
Program Partitioning

- **What is a program segment?**
  - Roughly a sub-graph of the CFG

- **Why program partitioning?**
  - Reduce problem state space → reduce analysis efforts
  - Precision is sacrificed

- **Partitioning granularity**
  - Fewer segments → less instrumentation efforts but higher analysis computation overhead

- **“Good” partitioning**
  - Balance “the # of program segments” and “the average # of paths per segment”
Program Partitioning

- An Example of Program Partitioning

(a) Partitioning Results

| Path Bound | |PSG| | #Paths (Σ|πj|) |
|---|---|---|---|
| 1 | 30 | 30 |
| 5 | 6 | 14 |
| 10 | 3 | 14 |
| 20 | 2 | 18 |
| 100 | 1 | 72 |

(b) Dependency between |PSG| and ∑|Πj|

```
int x;
int main_nice_partitioning()
    int y, int i, int a, int b
{
    if (x == 1) {
        x++; // BB 2
    } else {
        x--; // BB 4
    } // BB 3
    if (b == 1) {
        // BB 5
        if (a == 1) {
            x++; // BB 9
        } else {
            // BB 11
        } // BB 7
        if (x == 3) {
            x++; // BB 9
        } else {
            // BB 11
        } // BB 12
        if (x == 2) {
            x++; // BB 13
        } else {
            // BB 14
        } // BB 11
        if (x == 4) {
            x++; // BB 15
        }
    }
    } else {
    x++; // BB 17
    } // BB 13
```
Test Data Generation

- What is the so-called “test data”?
  - Roughly, the values of a set of variables that leads to one of the paths of a program segment

- What is the use of “test data”?
  - Put code instrumentations at the segment boundaries, and set the test data to some specific values, which can leads the program to the desired path

- How to obtain “test data”? – model checking
Test Data Generation
Test Data Generation

- **Execution Time Measurement**
  - Use software instrumentation to guide the program
  - Use hardware instrumentation to measure execution time

- **Enforcing Predictable Hardware States**
  - Challenge: on complex hardware where the instruction timing depends on the execution history
  - Code instrumentations can be used to enforce an a-priori known state at the beginning of a program segment, thus avoiding the need for considering the execution history

- **WCET Calculation**
  - Use ILP, Model Checking, or any optimization tools to do longest path search
Probabilistic WCET Analysis

- **What is probabilistic WCET analysis?**
  - It gives you a distribution of the execution time of a program, instead of single WCET value

- **Why probabilistic WCET analysis?**
  - To determine the probability distribution of the execution times of tasks, then used to do probabilistic schedulability analysis in soft real-time systems
  - Helping to detect the “WCET hotspot”, used for WCET reduction
  - Helping to analyze the execution behaviors of a program
Solution: Probabilistic Timing Schema

Timing Schema
- $W(A) = \text{exec time } A$
- $W(A;B) = W(A) + W(B)$
- $W(\text{if } E \text{ then } A \text{ else } B) = W(E) + \max(W(A), W(B))$

Probabilistic Timing Schema

Sequential execution: $Z = X + Y$

Distribution functions: $F(x) = P[X \leq x]$, $G(y) = P[Y \leq y]$

To compute $H(z) = P[X + Y \leq z]$.

If $X$ and $Y$ are independent
$$H(z) = \int_{x} F(x)G(z - x)dx$$

If joint distribution between $X$ and $Y$ is given as $J(x, y)$
$$H(z) = \int_{x+y=z} j(x, y)$$

If the joint distribution is unknown
$$H(z) = \int_{x+y=z} \frac{\partial^2 \min(F(x), G(y))}{\partial x \partial y}$$
Probabilistic WCET Analysis

- **Probabilistic Timing Schema**
  - **Conditional execution**: $Z = \max(X, Y)$
  - $Z = E + \max(X, Y)$, $\max(X, Y)$ has the distribution $H(z)$

$$H(z) = \int_{\max(x,y)=z} \frac{\partial^2 \min((F(x), G(y)))}{\partial x \partial y}$$

- **Iteration**: can be analyzed as a combination of sequence execution and conditional execution, loop bounds should be known

- **Determining Probability Distributions**
  - To determine the actual distribution of the execution times of individual units (basic blocks)
  - Run the units under a large number of test scenarios
The pWCET Analysis Tool

Obtaining execution traces. This is done by manually or automatically inserting instrumentation calls into the source code, or by automatically adding instrumentation codes into the compiled assembly code.

In this step, the CFG of the assembly code is reconstructed, and then converted into a syntax tree.

Compute the distribution functions of each node from the traces; Determine the joint distribution function of pairs of nodes; Loop identification, loop iteration extracted; This step is VERY computation expensive!!

Generate a program for WCET calculation, this is based on separating the timing analysis into a program generation part and an execution part. The generator traverses the tree in reversed order and applies the timing schema rules, and the results is a set of commands on how to compute the timing program for the given tree.

Run the generated program with the program to be analyzed, and calculate the probabilistic distribution of the execution times of the program.
## A Survey of WCET Tools

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>aiT</td>
<td>value analysis</td>
<td>static program analysis</td>
<td>IPET</td>
</tr>
<tr>
<td>Bound-T</td>
<td>linear loop-bounds and constraints by Omega test</td>
<td>static program analysis</td>
<td>IPET per function</td>
</tr>
<tr>
<td>RapiTime</td>
<td>n.a.</td>
<td>measurement</td>
<td>structure-based</td>
</tr>
<tr>
<td>SymTA/P</td>
<td>single feasible path analysis</td>
<td>static program analysis for I/D cache, measurement for segments</td>
<td>IPET</td>
</tr>
<tr>
<td>Heptane</td>
<td>-</td>
<td>static prog. analysis</td>
<td>structure-based, IPET</td>
</tr>
<tr>
<td>Vienna S.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vienna M.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vienna H.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vienna H.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWEET</td>
<td>value analysis, abstract execution, syntactical analysis</td>
<td>static program analysis for instr. caches, simulation for the pipeline</td>
<td>path-based, IPET-based, clustered</td>
</tr>
<tr>
<td>Florida</td>
<td></td>
<td>static program analysis</td>
<td>path-based</td>
</tr>
<tr>
<td>Chalmers</td>
<td></td>
<td>modified simulation</td>
<td></td>
</tr>
<tr>
<td>Chronos</td>
<td></td>
<td>static prog. analysis</td>
<td>IPET</td>
</tr>
</tbody>
</table>
## Support of Architectural Features

<table>
<thead>
<tr>
<th>Tool</th>
<th>Caches</th>
<th>Pipeline</th>
<th>Periphery</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiT</td>
<td>I/D, direct/set associative, LRU, PLRU, pseudo round robin</td>
<td>in-order/out-of-order</td>
<td>PCI bus</td>
</tr>
<tr>
<td>Bound-T</td>
<td>-</td>
<td>in-order</td>
<td>-</td>
</tr>
<tr>
<td>RapiTime</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>SymTA/P</td>
<td>I/D, direct/set-associative, LRU</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>Heptane</td>
<td>I-cache, direct, set associative, LRU, locked caches</td>
<td>in-order</td>
<td>-</td>
</tr>
<tr>
<td>Vienna S.</td>
<td>jump-cache</td>
<td>simple in-order</td>
<td>-</td>
</tr>
<tr>
<td>Vienna M.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>Vienna H.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>SWEET</td>
<td>I-cache, direct/set associative, LRU</td>
<td>in-order</td>
<td>-</td>
</tr>
<tr>
<td>Florida</td>
<td>I/D, direct/set associative</td>
<td>in-order</td>
<td>-</td>
</tr>
<tr>
<td>Chalmers</td>
<td>split first-level set-associative, unified second-level cache</td>
<td>multi-issue superscalar</td>
<td>-</td>
</tr>
<tr>
<td>Chronos</td>
<td>I-cache, direct, LRU</td>
<td>in-order/out-of-order, dyn. branch prediction</td>
<td>-</td>
</tr>
</tbody>
</table>
Contents

- An Introduction to WCET Analysis
- Static Analysis
- Measurement-Based Methods
- WCET Analysis of RTOS
- New Challenges and Future Trends
- Recommended Readings
Introduction of This Research Topic

- Real-Life Real-Time Systems are Composed of
  - RTOS
  - Applications

- Timing Correctness of a Real-Time System is guaranteed by
  - Schedulability analysis in the high level
  - WCET analysis in the low level

- Applying WCET tools for application programs to RTOS
  - Poor results are reported (up to 86% pessimism)
  - Hard to handle some RTOS specific programs

- Additional analysis techniques are required!
WCET Analysis of RTEMS

- **Research Group**
  - Antoine Colin & Isabelle Puaut @ IRISA

- **Experiment Setup**
  - WCET tool: Heptane (tree-based)
  - RTOS: RTEMS
  - Manual revision to codes
  - 12 out of 85 system calls, span across 91 files, 14,532 LOC
WCET Analysis of RTEMS

- Problem 1: unstructured control flow
  - Such as goto statements, multiple loop exits, …
  - Because Heptane is a tree-based WCET analysis tool
  - Consequences: (1) rewriting the codes; (2) only a small subset of RTEMS system calls are analyzed

- Problem 2: Dynamic function calls implemented through function pointers
  - Real called functions are determined at runtime
  - Solutions: replace them with static ones
Problem 3: Hard to determine loop bounds since the loop bounds are related to dynamic runtime behaviors
- Task queue, message queue manipulation
- Solution: Manually bound loops by an investigation of RTOS codes

Problem 4: Blocking system calls

Problem 5: Context switch overhead

Putting them all together, an average of 86% pessimism in the estimated results is reported
Challenges to WCET Analysis – Side Effects

- It is apparent that the state space can be reduced via composable or hierarchical design/analysis.

- Side effects are defined as task interactions that cannot be traced back to task interface. For example, the shared cache may enable task A to influence the execution time of task B by displacing B’s data in the shared region.

- Side effects are a big problem to composable timing analysis.
Predictable Architecture Design @ TuWein

- **Side Effects in Simple Hardware Architectures**
  - Variable program execution time due to
    - Unpredictable data input
    - Instructions with variable execution cycles dependent on operands

- **In Complex Hardware Architectures**
  - Different task instances may have different execution time
  - Scheduling without preemption: task instances from different tasks may execute alternatively, creating complex hardware states which are hard to predict
  - Scheduling with preemption: HW states change at preemption points, hard to predict when preemption will happen
  - Modern complex pipelines $\rightarrow$ flush not practical
Side Effects in Multicore Processors

- Shared cache: if two tasks on two different cores share the same cache lines, it is hard to bound the effects of mutual replacement of cache contents
- Other shared resources have similar problems
- Simultaneous Multi-Threading (SMT): also called hyper-threading by Intel, multiple tasks on the same core share the function units at instruction level, hard to analyze the execution time of each task with good precision
Solutions

The basic philosophy of Puschner’s solutions is to try every possibility to **AVOID** unwanted interactions

- (1) The use of single-path code in all tasks
- (2) The execution of a single task/thread per core
- (3) The use of simple in-order pipelines
- (4) Statically scheduled access to shared memory in CMPs

The solutions require redesign in both hardware and software (at both system level and application level)
Predictable Architecture Design @ TuWein

- An RTOS for a Time-Predictable Computing Node
Requirements on Hardware Architectures

- The execution times of instructions are independent of the operand values
- The CPU supports a conditional move instruction having invariable execution times
- Direct-mapped or set-associative caches with LRU
- Memory access times are invariable for all data items
- The CPU has a programmable instruction counter that can generate an interrupt when a given number of instructions has been completed
The SW Architecture – Task Model

Simple Task Model
- I/O operations will never block a task
- No statements for explicit I/O or synchronization within a task
- All inputs are ready at task startup
- Outputs are ready in the output variables when the task completes

Single-path Tasks
- Transformation techniques
**Single-Path Transformation**

```
if cond
then result := expr1;
else result := expr2;
```

```
tmp1 := expr1;
tmp2 := expr2;
test cond;
movt result, tmp1;
movf result, tmp2;
```

```
if cond
then \((v_1,\ldots,v_n) := F1(v'_1,\ldots,v'_m)\)
else \((v_1,\ldots,v_n) := F2(v'_1,\ldots,v'_m)\)
```

\[
(h_1,\ldots,h_n) := F1(v'_1,\ldots,v'_m)
\]

\[
(h'_1,\ldots,h'_n) := F2(v'_1,\ldots,v'_m)
\]

\[
\text{cond: } (v_1,\ldots,v_n) := (h_1,\ldots,h_n)
\]

\[
\text{not cond: } (v_1,\ldots,v_n) := (h'_1,\ldots,h'_n)
\]

```
-- conditions so far: cond-old
while cond-new do max expr times stmts;
```

```
finished_x := false;
for \(i_x := 1\) to expr do
begin
if not cond-new
then finished_x := true;
if cond-old and not finished_x
then stmts;
end
```
Predictable Architecture Design @ TuWein

- The SW Architecture – RTOS
  - There must be no jitter in the execution times of the RTOS routines
  - Kernel designed using the single-path techniques
  - Communications: messages are scheduled at fixed time off-line

1. Local buffer accessed by tasks
2. Global buffer managed by IPC
3. Inter-node communication
4. Message schedule defined off-line
Predictable Architecture Design @ TuWein

- The SW Architecture – RTOS
  - Scheduler
    - Time-triggered
    - Schedule is determined off-line
    - Scheduler invoked at each global clock tick
    - Mode-switch is implemented by schedule switch, also determined off-line
    - Tasks are divided into “initialization phase” and “real-time phase”, the former is non-real-time, the latter is managed by the RTOS
Predictable Architecture Design @ TuWein

- An Example

<table>
<thead>
<tr>
<th>Task</th>
<th>Time of activation</th>
<th>Execution time</th>
<th>Termination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Scheduler</td>
<td>0</td>
<td>1341</td>
<td>1341</td>
</tr>
<tr>
<td>Kernel</td>
<td>1341</td>
<td>1341</td>
<td>1341</td>
</tr>
<tr>
<td>$T_A$</td>
<td>8644</td>
<td>5634</td>
<td>6975</td>
</tr>
<tr>
<td>Kernel</td>
<td>9255</td>
<td>1626(+43)</td>
<td>9255</td>
</tr>
<tr>
<td>IPC</td>
<td>10924</td>
<td>628</td>
<td>11552</td>
</tr>
<tr>
<td>Kernel</td>
<td>11552</td>
<td>1626(+43)</td>
<td>13221</td>
</tr>
<tr>
<td>$T_B$</td>
<td>13221</td>
<td>1000</td>
<td>14221</td>
</tr>
<tr>
<td>Kernel</td>
<td>14221</td>
<td>1626(+43)</td>
<td>15890</td>
</tr>
<tr>
<td>$T_C$</td>
<td>15980</td>
<td>37</td>
<td>15927</td>
</tr>
<tr>
<td>Kernel</td>
<td>15927</td>
<td>1626(+43)</td>
<td>17596</td>
</tr>
<tr>
<td>$T_B(resumed)$</td>
<td>17596</td>
<td>1771</td>
<td>19369</td>
</tr>
</tbody>
</table>
Predictable Architecture Design @ TuWein

Evaluations
- Puschner has posed insights on design for predictability
- Single-path technique is too costly and rigid
- Requiring both specialized hardware and software (RTOS) may be impractical
- In all, the ultimate predictability is achieved at the cost of system flexibility
Schneider studied combined schedulability & WCET analysis in his Ph.D. thesis, issues discussed in his work include:

- The quality of WCET analysis of RTOS can be improved by considering both the applications and the RTOS.
- In real-life multitasking real-time systems, tasks are executed in an interleaving manner (interrections), but this is not considered in traditional WCET analysis, under such a circumstance, both scheduling and WCET must be re-think.
Combined Schedulability & WCET Analysis

Why Combined Schedulability & WCET Analysis?

- Traditional schedulability and WCET analysis are performed in a hierarchical manner where the WCETs of the tasks are calculated first, then the results are fed to schedulability analysis.
- It is implied that even a task is interrupted, the WCET of all its segments equals the WCET of the task without interruptions.
- In multi-tasking systems running on complex hardware, the assumptions for hierarchical analysis is invalidated.
Why the assumption is invalidated?

As we have discussed in previous slides, the WCET of a program highly depends on the processor states in presence of complex hardware.

If a program is interrupted during execution, when it resumes, the hardware state is not identical to that at the interruption point, the influences are complex:

- Some needed cache contents are swapped out, so the WCET in presence of interruption is larger than that without interruption.
- If timing anomaly occurs, the displacement of cache contents may leads to a smaller WCET.
Combined Schedulability & WCET Analysis

- How to deal with these problems?
  - Consider the scheduling behavior within the WCET analysis process, and capture the state change at the interruption points
  - Re-calculate the WCET by considering the state change
  - Re-do schedulability analysis with new WCET values
Combined Schedulability & WCET Analysis

- The Old and New Analysis Framework

Diagram:

1. Tasks → WCET Analysis → WCETs → Schedulability Analysis
   
2. Task parameters e.g. T, D, i
   
3. Yes, or don't know
   
4. Cache related preemption costs

Interfering Tasks:

1. Tasks → WCET Analysis → WCETs
   
2. Task parameters e.g. T, D, i
   
3. Yes, or don't know
## A Summary of Research Practices in WCET Analysis of RTOS

<table>
<thead>
<tr>
<th>Problem</th>
<th>Colin</th>
<th>Schneider</th>
<th>Sandell</th>
<th>Petters</th>
<th>Puschner</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTOS Analyzed</td>
<td>RTEMS</td>
<td>OSE</td>
<td>OSE</td>
<td>L4</td>
<td></td>
</tr>
<tr>
<td>Analysis Tool</td>
<td>HEPTANE</td>
<td>ait</td>
<td>ait/SWEEET</td>
<td>Petters’ Tool</td>
<td></td>
</tr>
<tr>
<td>Average Overestimation</td>
<td>86%</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

### Problems Due to Program Features

<table>
<thead>
<tr>
<th>Problem</th>
<th>Colin</th>
<th>Schneider</th>
<th>Sandell</th>
<th>Petters</th>
<th>Puschner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Irreducible Program Structure</td>
<td>P2</td>
<td>P2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indexed Jumping</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td></td>
</tr>
</tbody>
</table>

### Problems Due to Lack of Application Information

<table>
<thead>
<tr>
<th>Problem</th>
<th>Colin</th>
<th>Schneider</th>
<th>Sandell</th>
<th>Petters</th>
<th>Puschner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard to Bound Loops Due to Runtime Properties</td>
<td>P2,3</td>
<td>P2,3</td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Dynamic Function Calls</td>
<td>P4</td>
<td></td>
<td></td>
<td>P4</td>
<td></td>
</tr>
<tr>
<td>Blocking System Calls</td>
<td>N</td>
<td></td>
<td></td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>Lack of Knowledge on System Call Contexts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of Knowledge on RTOS Running Mode</td>
<td></td>
<td></td>
<td></td>
<td>P2</td>
<td></td>
</tr>
</tbody>
</table>

### Problems Due to Task Switching and Inter-Task Interference

<table>
<thead>
<tr>
<th>Problem</th>
<th>Colin</th>
<th>Schneider</th>
<th>Sandell</th>
<th>Petters</th>
<th>Puschner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing Effects Due to Task Switching</td>
<td>P1</td>
<td></td>
<td></td>
<td>P4</td>
<td></td>
</tr>
<tr>
<td>Timing Anomalies Due to Preemption</td>
<td>P1</td>
<td></td>
<td></td>
<td>P4</td>
<td></td>
</tr>
<tr>
<td>Inaccurate Execution Time of Context Switches</td>
<td>N</td>
<td>S</td>
<td>N</td>
<td>P4</td>
<td></td>
</tr>
<tr>
<td>Inter-Task Interference Due to Resource Sharing on Multicores</td>
<td>N</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

* “S”: the problem is properly solved
* “N”: the problem is circumvented in related research
* “P”: the problem is partially solved, but needs further development. Possible problems may be low scalability of the analysis (P1), too much user intervention required (P2), low quality of the results (P3), or the adopted techniques are too restrictive (P4)
A Summarization of Problems

- Problem 1: Irreducible program structures
  Solution: choose a proper WCET tool

- Problem 2: Lack of application information greatly affects analyzability and the precision of the results
  - Bounding loops
  - Dynamic function calls and blocking system calls
  - System call context and RTOS working mode
  Solution: extract helpful information from applications

- Problem 3: multi-tasking
  Solution: develop analysis techniques that can safely bound the effects of task switching
Challenges on WCET Analysis of RTOS

- Does Single WCET Value Suffice?
  - The running of RTOS is mode-based, so a single WCET value regardless of execution mode is not sensible
  - Related techniques, such as parametric ILP should be developed

- Considering Both Applications and RTOS
  - Application information may be very useful to RTOS analysis, e.g. bounding loops
  - What kinds of application information should be communicated to the analyzer?
  - How can these information be communicated to the analyzer?
Challenges on WCET Analysis of RTOS

- Combined Schedulability and WCET Analysis
  - There is a mutual communication between schedulability analysis and WCET analysis
  - Control of the state space explosion

- Raising the Degree of Automation
  - Almost all related research practices reported low degree of automation in the analysis
  - WCET tool designers must always keep the issue of “automation” in mind when designing tools
  - The degree of automation is the largest factor that affects the usability of a WCET tool
Challenges on WCET Analysis of RTOS

- Managing Analysis Complexity in the Multicore Era
  - Problem: fine-grained access to shared resources (L2 cache, on-chip bus, ...), and for most existing architectures, we have very limited ability to control the behavior of these shared resources
  - Solution: Performance isolation techniques (cache partitioning), since such techniques can “create” an isolated environment for each core, and at the same time still maintains the flexibility that shared resources provide with
Challenges on WCET Analysis of RTOS

- To Design or to Analyze?
  - Analyze
    - No need to change existing hardware or system; analysis must be done if you’re to analyze fabricated systems
    - But lots of hardware features or management policies are not designed for real-time, these features make the analysis very hard
    - To guarantee predictability on unpredictable hardware, a lot of pessimism is introduced into the results → system over design
  - Design
    - To design hardware or software with the consideration of real-time from scratch can yield very predictable systems
    - Predictability is achieved by sacrificing flexibility
    - New hardware requires re-design of the system, from hardware, to programming tools, to OS and applications
  - A Graceful Balance!
Partitioning is used to avoid inter-task interference regardless of single- or multi-core. Locking is used to enforce predictability in terms of cache hits/misses.
Contents

- An Introduction to WCET Analysis
- Path Analysis
- Micro-architecture Analysis
- A Survey of Academic and Industrial WCET Tools
- WCET Analysis of RTOS
- New Challenges and Future Trends
- Recommended Readings
Trends in Hardware

- More software control
  - Software-controlled cache locking
  - Scratchpad memory
  - More predictable caches or pipelines

- Multi-core processors
  - + multiple simple cores
  - - Shared cache $\rightarrow$ inter-task interference
  - - Share whatever, on-chip buses or networks

- Execution Behavior
  - Traditionally, researchers assume single task execute on single core, but this is not necessarily the whole story
  - A big gap between WCET and ACET
Trends in Software

- Levels of Abstraction
  - Traditionally C code or assembly code
  - A trend towards higher-level abstraction, e.g. OO languages, model-based design
  - More dynamic control structure, hard to reconstruct CFG
  - More dynamic data structure, memory access
  - Java VM, JIT compilation

- Component-based design
  - FSM synthesize highly unstructured code
  - Parameterized execution time/WCET
Trends in Analysis Techniques

- **WCET-aware Compilation**
  - Try to tackle the analysis complexity problem in compilers
  - Develop compilers that can generate predictable codes

- **Raise Automation Level**
  - Automatic extraction of flow facts, less user intervention
  - Flow facts mapping across different representation levels

- **Parametric WCET Analysis**
  - Obtain a function for WCET results, instead of a single WCET value

- Integrate WCET analysis with power-aware techniques
- Integrate WCET analysis with scheduling analysis
Contents

- An Introduction to WCET Analysis
- Path Analysis
- Micro-architecture Analysis
- A Survey of Academic and Industrial WCET Tools
- WCET Analysis of RTOS
- New Challenges and Future Trends
- Recommended Readings
Recommended Readings

- **Books**

- **Related Course Pages**
  - http://ti.tuwien.ac.at/rts/teaching/courses/wcet-ss08

- **Referenced Papers**
  - Surveys and Overview Papers
Recommended Readings


**Static Analysis**


**Measurement-Based Analysis**

Recommended Readings


**WCET Analysis of RTOS**

Recommended Readings


Tools & Projects

- aiT: [www.ait.com](http://www.ait.com)
- Bound-T: [www.tidorum.fi/bound-t/](http://www.tidorum.fi/bound-t/)
- RapiTime: [www.rapitasystems.com](http://www.rapitasystems.com)
- SymTA/P:
  - Vienna: [http://www.wcet.at/](http://www.wcet.at/)
  - Chalmers: [http://www.ce.chalmers.se/research/group/hpcag/project/wcet.html](http://www.ce.chalmers.se/research/group/hpcag/project/wcet.html)
Visit Our Website 😊

- The Website of Real-Time Embedded Systems Laboratory, Northeastern University
  - http://www.neu-rtes.org
  - http://www.neu-rtes.org/courses/spring2009/

- You can find
  - General information on the projects conducted in our lab
  - Research and publications
  - Research information and contacts of the members
  - Some useful research links

- Write me emails if you have questions in WCET or RTS
  - mingsong@research.neu.edu.cn