WCET Analyzers for Industry

Christian Ferdinand
AbsInt Angewandte Informatik GmbH
AbsInt Angewandte Informatik GmbH

- Provides advanced development tools for embedded systems, and tools for validation, verification, and certification of safety-critical software
- Founded in February 1998 by six researchers of Saarland University, Germany
- Privately held by the founders
- Selected Customers:
Hard Real-Time Systems

- Controllers in planes, cars, plants, ... are expected to finish their tasks within reliable time bounds.
- Schedulability analysis must be performed.
- Hence, it is essential that an upper bound on the execution times of all tasks is known.
- Commonly called the Worst-Case Execution Time (WCET).
The Timing Problem

- **Best-case execution time**
- **Unsafe:** execution time measurement
- **Exact worst-case execution time**
- **Safe worst-case execution time estimate**
Embedded Control Software

- Tends to be large and complex
  - Lots of functionality
  - Code-generating tools
  - 3rd party software
    - RTOS
    - communication libraries
The Ever-Growing Gap

\[ x = a + b; \]

```
LOAD r2, _a
LOAD r1, _b
ADD r3, r2, r1
```

68K (1990)

Execution time (clock cycles)

<table>
<thead>
<tr>
<th></th>
<th>Best case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

MPC 5xx (2000)

Execution time depending on flash memory

<table>
<thead>
<tr>
<th>Wait cycles</th>
<th>Time (clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>External</td>
<td>30</td>
</tr>
</tbody>
</table>

PPC 755 (2001)

Execution time (clock cycles)

<table>
<thead>
<tr>
<th></th>
<th>Best case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>320</td>
</tr>
</tbody>
</table>
a³ aiT WCET Analyzer

Combines

- **global static program analysis** by Abstract Interpretation:
  - microarchitecture analysis (caches, pipelines, ...) + value analysis
- integer linear programming for path analysis

in a single intuitive GUI.

![Diagram of aiT WCET Analyzer process]

**Application Code**

- Compiler
- Linker
- Executable (*.elf / *.out)

**Specifications (*.ais)**

- clock 10200 kHz
- loop "_codebook" + 1 loop exactly 16 end;
- recursion "_fac" max 6;
- SNIPPET "printf" is NOT ANALYZED AND TAKES MAX 333 CYCLES;
- flow "U_MOD" + 0xAC bytes / "U_MOD" + 0xC4 bytes is max 4;
- area from 0x20 to 0x497 is read-only;

**EntryPoint**

- aiT

- Worst Case Execution Time: 572 cycles
- Visualization, Documentation
Kelvin D. Nilsen, Bernt Rygg, Worst-Case Execution Time Analysis on Modern Processors

“Furthermore, given the ever increasing sizes of multiple-level cache hierarchies, and the high complexity of static cache-behavior analysis, it seems unlikely that, even in the best of circumstances, the cache analyzer can predict more than 50% of the actual cache hits for realistic workloads.”
PAG Program Analyzer Generator

Specifying
- Domains
- Transfer functions
- Join functions
- Interface

PAG
Program Analyzer Generator

C Code

Compiler

Front End

Control Flow Analyzer

Optimizer

Optimizer

Optimizer

Back End

Libraries
Example: Direct Mapped I-Cache

CPU

Program Counter: 1028

Instruction: mul 1024

I-Cache

1032: ble 1024
1028: mul ...

Main memory

1024: add ...
1028: mul ...
1032: ble 1024

Cache Hit: ~ 1 Cycle

Cache Miss: ~ +1 to +100 Cycles
Set Associative Cache

CPU

Address: Address

Set number

Byte in line

1

2

... A

Adr. prefix Tag Rep Data block Adr. prefix Tag Rep Data block ...

Set: Fully associative subcache of A elements with LRU, FIFO, rand. replacement strategy

... ...

... ...

... ...

... ...

Compare address prefix

If not equal, fetch block from memory

Byte select & align

Main Memory

Data Out
Example:
Fully Associative Cache (2 Elements)
Abstract Semantics: Transfer

concrete

abstract

"young"

"old"

Age

{s}
Abstract Semantics: Join

**Join (must)**

```
{ a }
{ }   
{ c, f }
{ d }
```

```
{ c }   
{ e }   
{ a }   
{ d }   
```

“intersection + maximal age”

```
{ }      
{ }      
{ a, c }  
{ d }      
```

Interpretation: memory block a is definitively in the (concrete) cache => always hit

Question: How many references will a memory block surely survive in the cache?
Structure of the aiT WCET Analyzer

Executable program

CFG builder

Loop trafo

CRL file

AIS file

Static analyses

Loop analyzer

Value analyzer

Cache/pipeline analyzer

Path analysis

ILP generator

LP solver

Evaluation

WCET, visualization
Pipeline Analysis

- Goal: calculate all possible pipeline states at a program point
- Method: perform a cycle-wise evolution of the pipeline, determining all possible successor pipeline states
- Implementation: from a formal model of the pipeline, its stages and communication between them
- Generation: from a PAG specification
- Result: WCET for basic blocks
Pipelines

Ideal case: 1 instruction per cycle
Pipeline of the PPC755

[Diagram of the PPC755 pipeline with various units and connections]
Pipeline Model
Visualization of Pipeline Analysis Results
Path Analysis
by Integer Linear Programming (ILP)

- Execution time of a program =
  \[ \sum_{Basic\_Block \ b} \text{Execution\_Time(b)} \times \text{Execution\_Count(b)} \]

- ILP solver maximizes this function to get the WCET
- Program structure described by linear constraints
  - automatically created from CFG structure
  - user provided loop/recursion bounds
  - arbitrary additional linear constraints to exclude infeasible paths
Path Analysis: Example

(simplified constraints)

if \( a \) then
    \( b \)
elseif \( c \) then
    \( d \)
else
    \( e \)
endif
\( f \)

max: \( 4x_a + 10x_b + 3x_c + \\
     2x_d + 6x_e + 5x_f \)

where \( x_a = x_b + x_c \)
\( x_c = x_d + x_e \)
\( x_f = x_b + x_d + x_e \)
\( x_a = 1 \)

Value of objective function: 19

\[
\begin{align*}
x_a & = 1 \\
x_b & = 1 \\
x_c & = 0 \\
x_d & = 0 \\
x_e & = 0 \\
x_f & = 1
\end{align*}
\]
Worst Case Execution Time: 572 cycles

main

countTime2

countTime2_reset

countTime2.L1 (loop)
countTime2_reset.L1 (loop)

modCount

modCount_reset

AbsInt
Angewandte Informatik
Domino Effect

- Timing anomaly
- Execution time increase is not bounded by hardware determined constants
- Certain instruction sequences e.g. in loop bodies can trigger this effect and increase latencies in further iterations
Pseudo-LRU Replacement (PPC755)

- Each setting of $B[0..2]$ points to a specific line:

```
  B0
  □  □
  /  \\nB1  B2
□  □
|  |
L0  L1  L2  L3
```

0  1
0  1
0  1

# 4-way PLRU Domino Effect

## Empty cache

<table>
<thead>
<tr>
<th></th>
<th>c:</th>
<th>d:</th>
<th>f:</th>
<th>c:</th>
<th>d:</th>
<th>h:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td></td>
<td></td>
<td></td>
<td>c:</td>
<td>d:</td>
<td></td>
</tr>
</tbody>
</table>

## Non-empty cache

<table>
<thead>
<tr>
<th></th>
<th>c:</th>
<th>d:</th>
<th>f:</th>
<th>c:</th>
<th>d:</th>
<th>h:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td></td>
<td></td>
<td></td>
<td>c:</td>
<td>d:</td>
<td></td>
</tr>
</tbody>
</table>

- This sequence is then repeated ad infinitum
- Only cache hits
- Two misses each time
Pipeline of the PPC755
Domino Effect on Instruction Sequence S1

A \ lwz \ r20, \ 0(r2) \\
B \ addi \ r21, \ r20, \ 4 \\
C \ mullw \ r19, \ r14, \ r29 \\
D \ lwz \ r23, \ 0(r20) \\
E \ addi \ r24, \ r23, \ 4 \\
F \ addi \ r25, \ r14, \ 4 \\
G \ lwz \ r26, \ 0(r19) \\
H \ mullw \ r27, \ r14, \ r29 \\
I \ lwz \ r28, \ 0(r26) \\
J \ addi \ r22, \ r28, \ 0 \\

- mullw can only be executed by integer unit IU1
- lwz can only be executed by the load/store unit LSU
- S1 must be repeated at least 3 times
Execution Units Overview

Distribution of instruction sequence S1 on the execution units IU1, IU2 and LSU.

- In cycle 1 instructions A and B are dispatched to LSU and IU2. So C can be dispatched to IU1 in cycle 1.
- \( 10 + 9(n-1) \) cycles are needed with \( n \) being the number of iterations.
Example: Domino Effect

Distribution of instruction sequence S1 on the execution units IU1, IU2 and LSU with an additional leading instruction X. **Domino effect**!

- With the insertion of instruction X, B is dispatched to IU1 in cycle 1.
- C can only be executed by IU1 and so has to wait for B to finish. B has to wait for the results of A.
- While J is executing B can be already dispatched to IU1 and the stream is again delayed
- **3 more cycles per iteration (33%)!!**
Effort to support new processors?

Executable program

Call- & CFG Graph Builder

Loop Transformation

CRL2 File

Static Analyses

Loop-Bound Analyzer

Value Analyzer

Cache/Pipeline Analyzer

Path Analysis

ILP-Generator

LP-Solver

Evaluation

AIS File Loop Bounds

AIS File

CRL2 File

AIS File
Pipeline Analyzer Generation

- Semi-automatic process
- Based on VHDL specification
- Generates C-Code that
  - performs abstract simulation of system behavior,
  - fits into the aiT framework and
  - incorporates the usual abstractions
- Theoretical background done in research project AVACS
  - National research program for basic research
  - Saarland University Prof. Wilhelm
  - without industrial participation
Semi-Automatic Derivation of Timing Models
Deriving the Timing Model

- Processor specification too large to be used in aiT framework
  Infineon PCP2 (~40,000 loc), Leon2 (~80,000 loc), Infineon TriCore 1.3 (~250,000 loc)
- Specification needs to be compressed
SCADE / aiT automated Flow
Analysis Reports

- Customizable HTML reports
- Global and detailed reports
- Diff feature
Integration with Modelling Tools
Example: ETAS ASCET MD
ALL-TIMES

- aiT (AbsInt)
- SymTA/S (Symtavision)
- T1 (Gliwa)
- SATiRe (TU Vienna)
- RapiTime (Rapita Systems)
- SWEET (MDH)

A. Combination of analysis and measurement
B. Measurements
C. Executable reader
D. Code execution time
E. Mutually exclusive execution paths
F. Measurements
G. Flow info
H. Activation events
I. Code execution time
J. Code execution time
K. Instrumentation
L. Evidence from measurement
M. Code execution time
N. Activation events
O. Automated annotation generation
P. Automated annotation generation
Q. Provision of frontend
R. Sharing of analysis results
S. Flow facts
T. Flow facts
XTC (Extensible Timing Cookies) Interface

- SymTA/S System model
- Code execution times Request
- Code execution times Response

Refinement

XTC 2.0

aiT / TimingExplorer

SymTA/S Scheduling Analysis
Timing in the V-Model

- hardware selection
- dimensioning
- mapping and configuration

- system verification
- availability & safety
- extensibility

- model refinements
- adjusting to new requirements

- debugging
- software optimization
- software integration

- re-use of models of 1st Generation

Timing Debugging
TimingExplorer: Early Estimation Problem

```
void Task (void)
{
  variable++;
  function();
  while (next)
  {
    do this;
    next--;
  }
  terminate();
}

Source code or models
```
TimingExplorer
ECU-level Exploration During Early Design Phases

Core/Config 1

Core/Config 2

Core/Config 3

Source files

T1
void Task (void)
{
    variable++;  // line 43
    function();
    while (next) {
        do this;
        next--;
    }
    terminate();
}

T2

WCET

<table>
<thead>
<tr>
<th>T1</th>
<th>388</th>
<th>500</th>
<th>253</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>760</td>
<td>896</td>
<td>543</td>
</tr>
</tbody>
</table>
Configuration Example
AIS Example

```plaintext
# Cache Architecture

# cache instruction
  set-count = 128, assoc = 4, line-size = 32,
  policy = LRU, may = empty

# and data
  set-count = 128, assoc = 4, line-size = 32,
  policy = LRU, may = empty;

# Properties of memory area

# everything completes in one cycle

# area 0x0..0x7fffffff

  access
    code cached, transfer-time = [1,1,1,1,1],
    data cached, write-through,
    data read cyclic-burst, transfer-time = [1,1,1,1,1],
    data write transfer-time = [1,1];
```
PREDATOR: Design for predictability and efficiency
Qualification Support Kits

- **Report Package**: template html files
  - **Operational Requirements Report**: lists all functional requirements
  - **Verification Test Plan**: describes one or more test cases to check each functional requirement.

- **Test Package**:
  - All test cases listed in the verification test plan report
  - **Scripts** to execute all test cases including an evaluation of the results
WCET Challenge 2006

- Organized by the University of Mälardalen
  http://www.idt.mdh.se/personal/jgn/challenge/
- Aim: Compare different approaches in analyzing the Worst-Case Execution Time
- Excerpt from the final report: "aiT is able to handle every kind of benchmark and every test program that was tested in the Challenge. aiT is able to support WCET analysis even for complex processors. [...] aiT demonstrates its leading position through all its features [...]"
- Full report: http://dc.informatik.uni-essen.de/Tan/all/
Recent Advances

- Over-estimation cache-miss penalty
- 20-30% over-estimation in 1995 (Lim et al.)
- 15% over-estimation in 2002 (Thesing et al.)
- 30-50% over-estimation in 2005 (Souyris et al.)
Safety-Critical Hard Real-Time Developments

- aiT enables development of complex hard-real time systems on state-of-the-art hardware
- Increases safety
- Saves development time and costs
- Usability proven in industrial practice
Concluding Remarks

- It took about 10 years to establish static code-level timing analysis in industry
- Ongoing research
- We had good support
  - Research cooperation
  - Starterzentrum
  - Customers