parMERASA
WCET Analysis Tools

Haluk Ozaktas (Univ. of Toulouse)
Worst-Case Execution Time

probability

true WCET

measurement time

measurements

hybrid measurement-based

RapiTime

static analysis

22-Jan-13

parMERASA - WCET Analysis Tools - UPS
Static WCET analysis

source code

\[ N = \text{input data} \]

\[
even = 0; \\
odd = 0; \\
\text{for (i=0 ; i<N ; i++)} \\
\text{if (i%2 == 0)} \\
\text{even++;} \\
\text{else} \\
\text{oDD++;} \\
\]

flow analysis

low-level analysis

linear program

\[
\text{max } T = x_A \cdot c_A + x_B \cdot c_B + x_C \cdot c_C + x_D \cdot c_D + x_E \cdot c_E \\
x_A = 1 \\
x_B = x_{AB} + x_{EB} \\
x_C = x_{BC} \\
x_D = x_{BD} \\
x_E = x_{CE} + x_{DE} \\
x_{AB} \leq 255 \cdot x_{AB}; \\
x_{BC} \leq 0.5 \cdot x_B; \\
\]

IPET (Implicit Path Enumeration Technique)

solution

Tmax = \ldots \text{ (WCET)}

with:

\[
x_A = \ldots \quad x_B = \ldots \\
x_C = \ldots \quad x_D = \ldots \\
x_E = \ldots \\
\]

parMERASA - WCET Analysis Tools - UPS
WCET analysis on multicores

1st challenge: multicore architecture
conflicts to shared resources?
⇒ time-predictable architecture
(known upper bounds for latencies)

2nd challenge: parallel programming
inter-thread synchronisations generate
waiting times
⇒ need to bound them!
Timing analysis of synchronisations
Timing analysis of synchronisations

- **Computing WCWTs**

**Flow analysis**
- Loop bounds
- Infeasible paths

**Low-level analysis**
- Worst-case costs
  of basic blocks

**WCET analysis of a parallel application**

- Starting from the main function

- WCET of code fragment
  *without stall*

- Recursive

- WCWT related to each
  synchronisation operation
Implementation in OTAWA

description of synchronisations

hardware description (XML)

ISA support (plugins)

architecture abstraction

program representation
instructions, basic blocks, CFG

annotations
can be hooked to any object to express built-in or user-defined properties (e.g. WCET of a basic block, loop header, etc.)

program (.elf)

flow facts (e.g. loop bounds *)

WCET computation (IPET)

display (Eclipse plugin)

built-in or user-developed analyses

- CFG virtualization (to analyze functions in their call context)
- loop analysis (e.g. dominance)
- instruction and data caches (based on abstract interpretation techniques)
  - categorisation of instruction fetches and data accesses (always hit, always miss, loop persistent, unpredictable)
- pipeline behaviour analysis
  - execution times of basic blocks

recursive WCWTs analysis

* Our oRange tool determines loop bounds from source code analysis
Achievements and plans

- **Current state**
  - General procedure defined
  - Annotation language specified
  - Preliminary version implemented and used on toy examples

- **Future work**
  - Refined analysis of synchronisation primitives
  - Assistance to code annotation
  - Visualisation
parMERASA
WCET Analysis Tools

Haluk Ozaktas (Univ. of Toulouse)
Verification and Profiling Tools in parMERASA

Dr Ian Broster
Rapita Systems Ltd

parMERASA
Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analysability
Tool Challenges

(Context: Verification of Parallel, critical systems software for Multicore)

Software Design
- Understanding legacy software and dependencies
- Concurrency bugs, diagnostics
- Complexity of communication

Timing verification
- Hardware: cores interact: buses, shared memory, cache coherency, ...
- Software: parallelized applications are complex – synchronization and waiting times

Testing
- Visualization & Comprehension
- Coverage/on-target testing
- Repeatability/testability
Rapita Tool Support

Extensions of RVS technology

Design Support
- Dependencies in legacy software
- Communication profiling

In-depth tracing
- Visualization techniques
- Graphical view

Timing analysis
- RapiTime for Multicore
- Worst case execution times
- Multi-core profiling
**Tool requirements**

- **First stage**: derive a master set of tool requirements
Identified 5 tools to be developed to support the parMERASA project:

1. Extensions to RapiTime to support the analysis of parallel programs
2. Extensions to RapiCover to support the analysis of parallel programs
3. Memory, cache and stack analysis tool for parallel programs
4. Tool to assist with the parallelisation of existing sequential software
5. Visualization and profiling tool for parallel programs

5: Visualization and profiling tool for parallel programs
To parallelize old/legacy software – where do you start?
- Minimize the communication
- Identify independent parts

Use trace/profiling information to support parallelization
Automated timing measurements for multicore

- Blocking, waiting, synchronization times
- Analyse parallel activities simultaneously

Waiting time calculations involving:

- Critical sections, barrier synchronisation etc

WCET: worst case execution times

“High water marks” (maximum measurements), profiling etc.
Tool development: WCET for parallel applications
Conclusion

- Software verification for multicore is challenging
  - We focus on the timing and profiling parts
- In parMERASA, having some control over the application and the hardware makes this easier
- General multi-core timing analysis is hard, but achievable

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 287519, parMERASA.