Towards Multicore WCET Analysis

Simon Wegener, AbsInt Angewandte Informatik GmbH
Outline

- **Introduction**
  Why are we interested in (multicore) WCET analysis?

- **Resource Conflicts**
  What are the challenges?

- **Analysis Techniques**
  How can we perform a multicore WCET analysis?

- **Reducing Resource Conflicts**
  Some strategies to minimize resource conflicts.

- **Multicore Architectures**
  Which COTS multicore architecture should I use?

- **Conclusion**
Introduction
Safety-Critical Hard Real-Time Software

- Controllers in planes, cars, plants, ... are expected to finish their tasks within **reliable time bounds**.
- **Timing analysis** must be performed.
### Table 1 — Topics to be covered by modelling and coding guidelines

<table>
<thead>
<tr>
<th>Topics</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enforcement of low complexity</td>
<td>A++</td>
</tr>
<tr>
<td>Use of language subsets</td>
<td>A++</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>A++</td>
</tr>
<tr>
<td>B++</td>
</tr>
<tr>
<td>C++</td>
</tr>
<tr>
<td>D++</td>
</tr>
</tbody>
</table>

**Criticality levels:** A (lowest) to D (highest)

---

**7.4.17** An upper estimation of required resources for the embedded software shall be made, including:

a) the execution time;

b) the storage space; and

---

Excerpt from:
Draft BS ISO 26262-6 Road vehicles - Functional safety – Part 6: Product development: software level
Execution Time Variablility (Singlecore)

1990: 68020

Execution Time (Clock Cycles)

Best Case

Worst Case

2001: MPC755

Execution Time (Clock Cycles)

Best Case

Worst Case

Up to a factor of 100 between best-case and worst-case!
The Timing Problem

![Graph showing the relationship between probability, execution time, and safety. The graph includes points labeled as BCET, Measurement, WCET, and Upper Bound, illustrating the variability in timing across different execution scenarios.](image-url)
Two Levels of Timing Analysis

• **Code level**
  - **Single** process, task, ISR
  - Focus on
    • Control flow
    • Processor architecture with pipelines and caches
    • WCET

• **System level**
  - **Multiple** functions or tasks
  - Focus on
    • Integration and scheduling
    • End-to-end timing
    • Worst-Case Response Time (WCRT)

\[
R_i = C_i + \sum_{j \in \text{ht}(i)} \frac{R_j}{T_j} \leq D_i = T_i
\]

Response time

Core execution time = WCET

Fixed-point problem

\# of preemptions

Interference
Problem Solved?

Reinhard Wilhelm et al.:  
The Worst-case Execution Time Problem  
— Overview of Methods and Survey of Tools

CONCLUSIONS

„The problem of determining upper bounds on execution times for single tasks and for quite complex processor architectures has been solved. Several commercial WCET tools are available and have experienced very positive feedback from extensive industrial use.“

Problem Solved? No!

- The statement only addressed singlecore architectures.
- At least 17 publications concerning multicore timing analysis have been presented at the past five instances of the WCET Workshop.
- Several past and current research projects investigated multicore timing analysis
  - ARAMiS
  - ARAMiS II
  - ARGO
  - ASSUME
  - CERTAINTY
  - CONIRAS
  - PREDATOR
  - T-CREST
  - ...
Resource Conflicts
Singlecore
Singlecore
Multicore

- Task 1
- Task 2
- Shared Memory
Multicore

Task 1

Task 2

Shared Memory
Multicore with Resource Conflicts

Task 1

Task 2

Shared Memory
Multicore with Resource Conflicts

![Diagram showing resource conflicts between Task 1 and Task 2 accessing Shared Memory.](image-url)
Any sound multicore WCET analysis must take interference delays into account!
Freescale P4080 Access Latencies

- Derived through measurements

Jan Nowotsch et al. *Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement*. ECRTS 2014

| TABLE I. P4080 MEMORY ACCESS LATENCIES FOR INCREASING NUMBER OF CONCURRENT CORES. LATENCIES USED FOR EVALUATION ARE MARKED BOLD. |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| read | 41 | 75 | 171 | 269 | 296 | 439 | 460 | 604 |
| write | 39 | 164 | 245 | 463 | 517 | 737 | 784 | 1007 |

- Write latency increased by 2550 % if all cores try to write concurrently to main memory
Analysis Techniques
Analysis techniques

- Joint analysis (integrated code-level/system-level analysis) vs. separate WCET analysis for each core
- Static methods vs. measurement-based/hybrid methods
- (Probabilistic methods)
Joint Analysis

- Simultaneous analysis of concurrently running tasks
- But: pipeline state graph of one basic block may contain already several thousand states when computing the singlecore WCET
- Computationally not feasible due to huge state space
Separate Analysis

- Use singlecore WCET analysis, add interference costs in extra step
- Computationally easier
- But: we need to take care for non-timing-compositional architectures
- Idea for memory accesses:
  - Since memory accesses are orders of magnitude slower than normal instructions, one can argue that the pipeline will drain during the processing of memory accesses.
  - Thus, the interference delays imposed by resource conflicts do not cause timing anomalies and can be added later to the singlecore WCET bound.
Measurement-based Analysis

• By its nature a joint analysis – effects of other running cores are visible in the measurements

• Quality depends on the source of measured events:
  – Software instrumentation leads to the probe effect
  – Embedded trace units like Nexus 5001 or ARM CoreSight allow the fine-grained observation of a core's program flow
  – However, timing information is more coarse-grained (e.g. in Branch History Message traces)

• But: with multiple running cores, the bandwidth of the trace port may be too low to emit all trace messages
Measurement-based Analysis (cont‘d)

• We observe the timing effects of events like DRAM refreshes and resource conflicts.
• That’s exactly what we want, but...
  – Assume one in ten accesses will suffer a severe interference delay.
  – During the observation of the program’s execution we measure for most accesses both the case where no interference happens as well as the case where the delay occurs.
  – Due to the worst-case assessment, we will incorporate the delay for all these accesses.
  – Thus, we overestimate the real WCET because many more accesses on the critical path will incorporate the delay than the “one in ten” ratio suggests.
Reducing Resource Conflicts
Solution: Privatisation of Shared Resources

- TDMA-based resource scheduling (cf. Schranzhofer et al., „Timing predictability on multi-processor systems with shared resources“, RePP workshop 2009)

- Needs changes on existing code
Solution: Privatisation of Shared Resources

- Copy data in warm-up phase from shared memory to local memory, copy data in cool-down phase from local memory to shared memory

- Needs tool or operating system support
- May have severe performance impact
Solution:
Runtime Resource Capacity Enforcement

- Uses three main concepts to reduce the interference delays
  - Limitation
  - Monitoring
  - Suspension
- Especially useful for mixed-critically systems
- Also provides a safety net against SEUs (single event upsets)
Multicore Architectures
Design Guidelines for Predictable Multicores

1. **Fully timing compositional cores**: no timing anomalies, no domino effects
2. **Disjoint** data and instruction caches
   - Unified caches cause uncertainties on data accesses to interfere with the instruction cache analysis
3. **LRU** replacement policies for caches
   - pLRU and FIFO replacement policies are not well predictable
4. **Private** caches
   - Shared caches induce uncertainty on their contents
5. **Private** memories, lonely sharing
   - Access latency to shared resources depends on utilization
6. **Shared bus protocol with bounded access delay**

**PREDATOR**

PREDATOR was an ICT project in the 7th Framework Program of the EU
Freescale QorIQ P4080

Power Architecture™
e500mc Core

Backside L2 Cache
128 KB

D-Cache 32 KB
I-Cache 32 KB

Memory Controller
64 bit DDR2/DDR3

Frontside L3 Cache
1024 KB

CoreNet™ Coherency Fabric

Memory Controller
64 bit DDR2/DDR3

Frontside L3 Cache
1024 KB
Freescale QorIQ P4080

1. Fully timing compositional cores
   -
2. Disjoint data and instruction caches
   - ✔
3. LRU replacement policies for caches
   - ✔
4. Private caches
   - ✗
5. Private memories, lonely sharing
   - ✗
6. Shared bus protocol with bounded access delay
   - ✗
Infineon AURIX TC27x

PMI
Scratchpad 32KB
I-Cache 16KB

FPU
TriCore™ 1.6 P

DMI
Scratchpad 120KB
D-Cache 8KB

LMU
Scratchpad 120KB
D-Cache 8KB

RAM
32KB

PMU
Data Flash
Key Flash
Boot ROM
Program Flash 2MB
Program Flash 2MB

SRI Cross Bar

Lockstep Core

PMI
Scratchpad 32KB
I-Cache 16KB

FPU
TriCore™ 1.6 P

DMI
Scratchpad 120KB
D-Cache 8KB

Lockstep Core

PMI
Scratchpad 24KB
I-Cache 8KB

FPU
TriCore™ 1.6 E

DMI
Scratchpad 112KB
Readbuffer 128B

Bridge

DMA

System Peripheral Bus
Infineon AURIX TC27x

1. Fully timing compositional cores
2. Disjoint data and instruction caches
3. LRU replacement policies for caches
4. Private caches
5. Private memories, lonely sharing
6. Shared bus protocol with bounded access delay
Infineon AURIX TC27x - Configuration

- Use one dedicated program flash memory for each of the performance cores to avoid conflicting accesses. Use the data flash for the efficiency core, if needed.

- Use the core-local data scratchpad whenever possible instead of the shared RAM to reduce conflicting accesses. If possible, preload data from the shared RAM and data flash to the local scratchpad memories to control when accesses to the shared memory happen.

- Place the stack in the core-local data scratchpad.

- Do not access the core-local scratchpad memories from other cores.

- I/O channels (CAN, FlexRay, . . .) should not be accessed by multiple cores. Assign each I/O channel in use to a specific core.
Conclusion
Conclusion

• Timing analysis of multicores is possible – but needs some work!
• Reduce resource conflicts with smart software architecture – use sharing only where really needed.
• Reduce resource conflicts by using smart configurations of COTS multicores.
• Predictable multicores: less complexity and more precise results.
Thank you for your attention!

Simon Wegener (wegener@absint.com)
AbsInt Angewandte Informatik GmbH
Science Park 1, D-66123 Saarbrücken, Germany
Solution:
Runtime Resource Capacity Enforcement

- Uses three main concepts to reduce the interference delays
  - Limitation
  - Monitoring
  - Suspension
- Especially useful for mixed-critically systems
- Also provides a safety net against SEUs (single event upsets)
Solution:
Runtime Resource Capacity Enforcement

- Limitation assigns each task a capacity (i.e. the number of allowed resource accesses)
Solution:
Runtime Resource Capacity Enforcement

• Runtime monitoring is used to observe the number of actual resource accesses
• Tasks exceeding their access capacity are suspended by the operating system
Solution:
Runtime Resource Capacity Enforcement

- The critical task does not exceed its deadline
Infineon AURIX TC27x - SRI XBar

- Up to 16 bus masters are connected to up to 15 slaves (+1 default slave).
- Resource conflict happens when two or more masters try to access the same slave device.
Infineon AURIX TC27x - SRI XBar

- Each slave has its own arbiter to handle the resource conflicts.
Infineon AURIX TC27x - SRI XBar

• Arbitration rules:
  – On the top level, the priorities of the master decide which request is handled first.
  – Priorities go from 0 to 7, with 0 the highest.
  – Only one master allowed per priority, except for priorities 2 and 5.
  – Priorities 2 and 5 form round-robin groups.
  – Within these groups, round-robin scheduling is performed.
  – Additionally, starvation is prevented with some kind of priority ceiling.

• The SRI XBar is well documented (about 70 pages in the TC27x user manual). It should thus be possible to derive the necessary formulas to predict the number of wait cycles depending on the number of conflicting accesses.