Parallelisation of Industrial Software for Hard Real-time Systems

Theo Ungerer
University of Augsburg, Germany
ungerer@informatik.uni-augsburg.de

SDRC 2013
Overview

- Hard real-time (HRT) demands and timing predictability
- EC projects MERASA and parMERASA achievements
- parMERASA parallelisation approach
- Beyond parMERASA: HRT, FT and TM
- Research challenges for TM from side of HRT
Increasing demand for functionality in current and future real-time embedded systems

Often demand for mixed criticality application execution

Increase of processor performance demanded
Hard Real-time Systems

- **Hard real-time:**
  - a deadline should never be missed
  - If missed it may cause harm to humans or equipment
Time is Relevant

is important in real life…

Just missed…

source: The HiPEAC roadmap slides
Safety-related **hard real-time** embedded systems require that a **deadline must never be missed**, 

→ need a **proof of timing requirements by WCET (worst case execution time) analysis**, or 

→ **at least**, demonstrate, depending on the criticality of the system, that the **implementation meets its timing requirements**.
Figure: Basic notions concerning timing analysis of systems

How to guarantee HRT demands?

- **Static WCET analysis**
  - *Modeling the processor and memory system*
  - Modeling all potential paths of the program
  - Compute WCET bound by ILP problem solver
  - E.g. OTAWA tool of UPS (Toulouse) or aiT of AbsInt (Saarbrücken)

- **Measurement-based/hybrid WCET analysis**
  - *Measurement of basic block execution lengths*
  - Modeling all potential paths of the program
  - Compute WCET value, not necessarily an upper bound
  - E.g. RapiTime tool of Rapita Systems Ltd. (York, UK)
  - Probabilistic timing analysis, new approach

- **Extensive Testing and adding a safety margin**
COTS (common of-the shelf) processors contain features that make a WCET analysis hard or even impossible, as e.g.

- Complex branch prediction, out-of-order execution, two level cache hierarchy, Simultaneous multithreading (SMT)

COTS multi-core processors bring in additional handicaps for hard real-time tasks

- Bus conflicts
- Shared secondary cache

COTS (multi-core) processors are designed for high average performance, not for timing predictability
Timing behaviour on a COTS multi-core is not analysable / hard to analyse and too pessimistic.

Our solutions:

- predictable embedded multi-core design
- in concert with WCET technology, verification tools, parallelisation support, and system architecture,
- and exemplary parallelisation of industrial applications.

MERASA project (2007-2010) was hardware-driven.
parMERASA project (2011-2014) is application-driven.
MERASA

Multi-Core Execution of Hard Real-Time Applications Supporting Analysability

EC FP-7 project 2007-2010
Project webpage: http://merasa.org
MERASA Project

MERASA multi-core architecture

- Timing-predictable multi-core architecture based on in-order SMT cores
- Capable of mixed execution of hard real-time and non real-time applications \(\rightarrow\) mixed criticality workloads
- Hard real-time support by full isolation of threads and bounding of interferences
Multi-Core Execution of parallelised Hard Real-Time Applications Supporting Analysability

EC FP-7 project 2011-2014
Project webpage: http://www.parmerasa.eu
parMERASA goes one step **beyond mixed criticality demands:**

We target future complex control algorithms by parallelising hard real-time programs to run on predictable multi-/many-core processors.

Currently, **timing behaviour of parallel applications is not analysable** with current programming paradigms and timing analysis techniques.
Select and parallelise industrial hard real-time applications.

Find ways to efficiently parallelise industrial applications for embedded real-time systems.

Provide hard real-time support in system software, WCET analysis and verification tools for multi-cores.

Develop techniques for time predictable multi-cores with 16 to 64 cores which are commercially feasible.

Contribute to Standards and Open Source Software.
### Industrial Use Cases for HRT Parallisation

- **Avionics (Honeywell International s.r.o.)**
  - 3D Path Planning for airborne collision avoidance
  - Stereo Navigation for aircraft localization when in loss of GNSS
  - Global Navigation Satellite System (GNSS)

- **Automotive (DENSO AUTOMOTIVE Deutschland GmbH)**
  - Engine control for diesel fuel injection

- **Construction Machinery (BAUER Maschinen GmbH)**
  - Control algorithm for dynamic compaction machine
parMERASA Tools

- **Static WCET analysis tool OTAWA (Univ. of Toulouse)**
  - specification of annotation format for source code annotations,
  - analysis of synchronisation primitives

- **Five verification and parallelisation support tools (Rapita Systems Ltd., York, UK)**
  - WCET analysis tool RapiTime enhanced for parallel programs;
  - Parallelisation assistance tool;
  - Visualisation and profiling tool for parallel programs;
  - On-target code coverage tool for parallel programs;
  - Memory, cache and stack analysis tool for parallel programs.
Generic Multi-core Architecture
(BSC, TU Dortmund and University of Augsburg)
- Clustered multi-core architecture based on simple cores and predictable interconnect
- New predictable NoC structures; new coherency cache defined

System Architecture and System-level Software
- System architecture with common kernel library for all three application domains (University of Augsburg)
- TinyAUTOSAR, TinyIMA, Construction Machinery RTE
Overview

- Hard real-time (HRT) demands and timing predictability
- EC projects MERASA and parMERASA achievements
- parMERASA parallelisation approach
- Beyond parMERASA: HRT, FT and TM
- Research challenges for TM from side of HRT
Challenges for Parallelisation

- Main challenge: WCET analysability and WCET speedup:
- Target: high WCET speed-up, not average case speed-up

\[
\text{WCET speed-up} = \frac{\text{WCET of sequential program}}{\text{WCET of parallel program}}
\]

- To ease WCET analysis:
  - Predictable hardware with low interferences between threads
  - Software with well-formed structure and known building blocks

“ParMERASA Parallel Pattern-supported Parallelisation Approach Respecting the Requirements of WCET Analysis”
• Start: sequential HRT program / high-level model
  – Embedded domain: not a single sequential program to start but a bunch of time-triggered tasks

• Target: timing predictable parallel programs
  – Approach targets WCET analysable parallel design patterns

• Two commonly used parallelisation schemes from HPC
  – Foster (PCAM)
  – Mattson (Pattern System/Language)
Start from High-Performance Approaches

PCAM of Foster

- Partitioning
  Decompose from analysis of data and functional dependencies into tasks

- Communication
  Adding communication between partitioned tasks

- Agglomeration
  with respect to communication vs. computation ratio

- Mapping
  on a multi-/many-core processor.

+ Methodical approach

Pattern-based of Mattson

- Finding Concurrency
  Task decomposition, data decomposition, group tasks, order tasks...

- Algorithm Structure
  Task parallelism, divide and conquer, pipeline, event-based coordination...

- Supporting Structures
  Master/worker, fork/join, loop parallelism, shared data, shared queue...

- Implementation Mechanisms
  Threads, synchronisation, barriers, message passing...

+ Parallel design patterns
+ Pattern Catalogue

- No timing considerations
The ParMERASA Approach

- Starting point: sequential program (problem description)
- Final result: **predictable** parallel program

Step 1: Targeting Maximum Parallelism
- Create model to reveal parallelism
- Model consisting of sequential parts and parallel design patterns
- Platform independent

Step 2: Targeting Optimal Parallelism
- Agglomeration of its nodes
- Creation of threads
- Mapping onto target architecture
- Platform dependent

Joined work with Ralf Jahr of Univ. of Augsburg
Aim: Notation for fast modelling of parallelism

Extension of UML2 Activity Diagram:
- Parallel design pattern is new node type similar to activity
- Activities: either sequential or encapsulate APD
- Parallel design patterns: multiple activities in parallel

Patterns are only way to introduce parallelism

Advantages over inventing a new notation:
- Well known, easy to understand, tools exist
- Support for dependencies, branches, and nesting
Parallelisation of Industrial Software for Hard Real-time Systems
Goal: Reveal sufficient parallelism for any platform as Activity and Pattern Diagram (APD)

Start with single activity, repeatedly apply two operations:
- **Replacement**: apply parallel design pattern
- **Splitting**: decompose into multiple activities
Transition from maximum to optimal parallelism by agglomeration and mapping

Similar to optimization problem:
  – Global Objective: reduce WCET, energy consumption, ...

Activity and Pattern Diagram

"Agglomeration and Mapping"

Threads/Tasks

Cores
• The Pattern Catalogue:
  – Basis for parallelization
  – Contains all allowed parallel design patterns
  – Description according to meta-pattern
  – Description is textual, no reference implementations
  – Implementation examples are optional
  – Grows over time

• For hard real-time systems:
  – Patterns organized in two layers (*parallel design patterns* and *synchronization idioms*)
  – Extension of the meta-patterns with *real-time prerequisites*, *synchronization idioms*, and *WCET hints*
Layer 1: **Timing Analysable Parallel Design Patterns** based on cooperation with *industrial partners*

- Machinery domain: control loops
  1) Periodic Task Parallelism Pattern
- Automotive domain: engine control code
  2) Periodic Task Parallelism Pattern - Sporadic Task Extension
- Avionic Domain: 3D path planning, Stereo navigation
  3) Task Parallelism Pattern
  4) Producer/Consumer (Pipeline) Parallelism Pattern
  5) Data Parallelism (SPMD/Geometric Decomposition) Pattern
  6) Periodic Task Parallelism Pattern
Layer 2: **Synchronisation Idioms**

- Depending on processor, programming model, ISA, RTOS etc.
- Helping the WCET analysis tool to compute WCETs on known, timing analysable synchronisation mechanisms
- Helping the programmer to specify the needed information for the WCET analysis tool
- Allowing portability, when different platforms are used
- Categorised (e.g.):
  - **data-exchange**
    - Locks, lock-free (non-blocking) data structures, message passing etc.
  - **progress coordination**
    - Barriers, conditional variables, point-to-point synchronisation/messages etc.

Joined work with Mike Gerdes of Univ. of Augsburg
Pattern Catalogue

- Status:
  - Prototype implementations of most of the patterns with POSIX threads to ease understanding of patterns
  - Based on interaction with company partners
  - Pattern Catalogue will be made available as Tech. Rep. Sept. 2013
WCET Analysis of Parallel Programs with Synchronisation

- Separation of execution time and waiting time
- Synchronisation functions must be timing predictable
- Waiting time for synchronisations must be bounded

\[
\text{global\_wcet} = \text{wcet}(M_1) + \max \left( \text{wcet}(M_2), \text{wcet}(W_1) \right) + \max \left( \text{wcet}(M_3), \text{wcet}(W_2) \right) + \text{wcet}(M_4)
\]

Joined work with Christine Rochange, Haluk Ozaktas of Univ. of Toulouse and Mike Gerdes of Univ. of Augsburg
WCET Analysis of Parallel Programs with Synchronisation

- Separation of execution time and waiting time
- Synchronisation functions (cs) must be timing predictable
- Waiting time for synchronisations must be bounded

\[ WCET = WCET_{comp} + WCWT + WCET(cs) \]
Overview

- Hard real-time (HRT) demands and timing predictability
- EC projects MERASA and parMERASA achievements
- parMERASA parallelisation approach
- Beyond parMERASA: HRT, FT and TM
- Research challenges for TM from side of HRT
Transactional memory (TM) well-known for its programmability advantages

Can we harness TM techniques for safety-critical embedded systems?

Two research approaches
  – TM and timing predictability
  – TM for fault tolerant execution

Research partly funded by an „Intel Germany Microprocessor Research Grant“

Joined work with Stefan Metzlaff and Sebastian Weis of Univ. of Augsburg
• “Synchronise” multiple hard real-time threads by TM instead of lock...unlock

• **Requirements for hard real-time (HRT) TM**
  – **Timing analysable TM primitives** (start, commit, abort, load, store, . . . )
  – **Commit guarantee for each transaction**
  – **Calculable number of transaction aborts**
    – HRT contention management

• Best be done by Hardware Transactional Memory

• Problem: WCET higher for TM as for pessimistic synchronisation primitives
Applications with tasks of different RT requirements, i.e. mixed-criticality

E.g.: Advanced Driver Assistance System
- Hard real-time (HRT): collision avoidance
- Soft real-time (SRT): night vision
- Best-effort (BE): traffic sign recognition

Data sharing among applications

Disallow interference of non-HRT tasks on HRT tasks
- Prioritised TM contention manager
1. Vulnerable code encapsulated in transactions
2. Replication of transactions in two or more threads
3. Comparison of CPU states on commit by HW or SW
4. Make commit visible
1. Vulnerable code encapsulated in transactions
2. Replication of transactions in two or more threads
3. Comparison of CPU states on commit by HW or SW
4. Make commit visible or retry in case of fault
Research challenges for TM from side of HRT

- Three Basics:
  - Timing analysable TM primitives (start, commit, abort, . . .)
  - Commit guarantee for each transaction
  - Calculable number of transaction aborts
- WCET of wait-free and lock-free algorithms
  - not concerning implementation overhead, but WCET
- WCET overhead for optimistic and pessimistic locking
  - Find tradeoff for application programs
- Parallel design patterns with TM synchronisation
  - Coding guidelines and WCET annotations
  - Pipelined transactions may be a step in this direction
- Transaction failure: combine FT and timing
- Sensor values change gradually: idea of a transaction commit that does not reexecute if values are only slightly different
- Timing predictability and multi-cores
- Achievements of MERASA and parMERASA projects
- parMERASA approach for predictable parallel software
  - Two steps
    - Reveal parallelism: architecture independent
    - Agglomerate and map: architecture dependent
  - Only parallel design patterns to introduce parallelism
  - Pattern catalogue
- Outlook on TM for Safety-Critical Applications
Our Mission

Make **timing predictable** techniques commercially feasible to **increase safety** in **avionics, automation and automotive** domains!