Experiences with Pattern-based Software Development

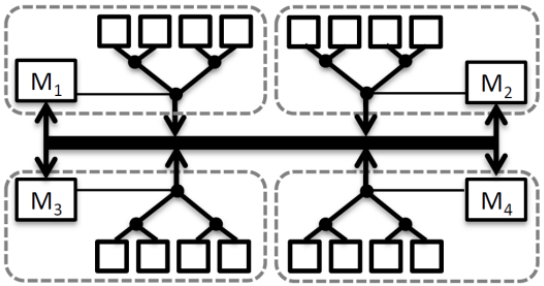
Ralf Jahr
University of Augsburg, Germany

parMERASA Dissemination Event,
Barcelona, 23/09/2014

Outline

- Motivation
- Parallelization with Parallel Design Patterns (PDPs)
  - Applied Concepts
  - parMERASA Pattern Catalogue with PDPs and Analyzable Synchronization Idioms
- Pattern-supported Parallelization Approach
  - Overview
  - Model-based Optimization
  - Timing-analyzable Algorithmic Skeletons (TAS)
  - Measures for Effectiveness of Parallelization
- Summary & Conclusion

Requirements for Static Timing Analysis of Parallel Software



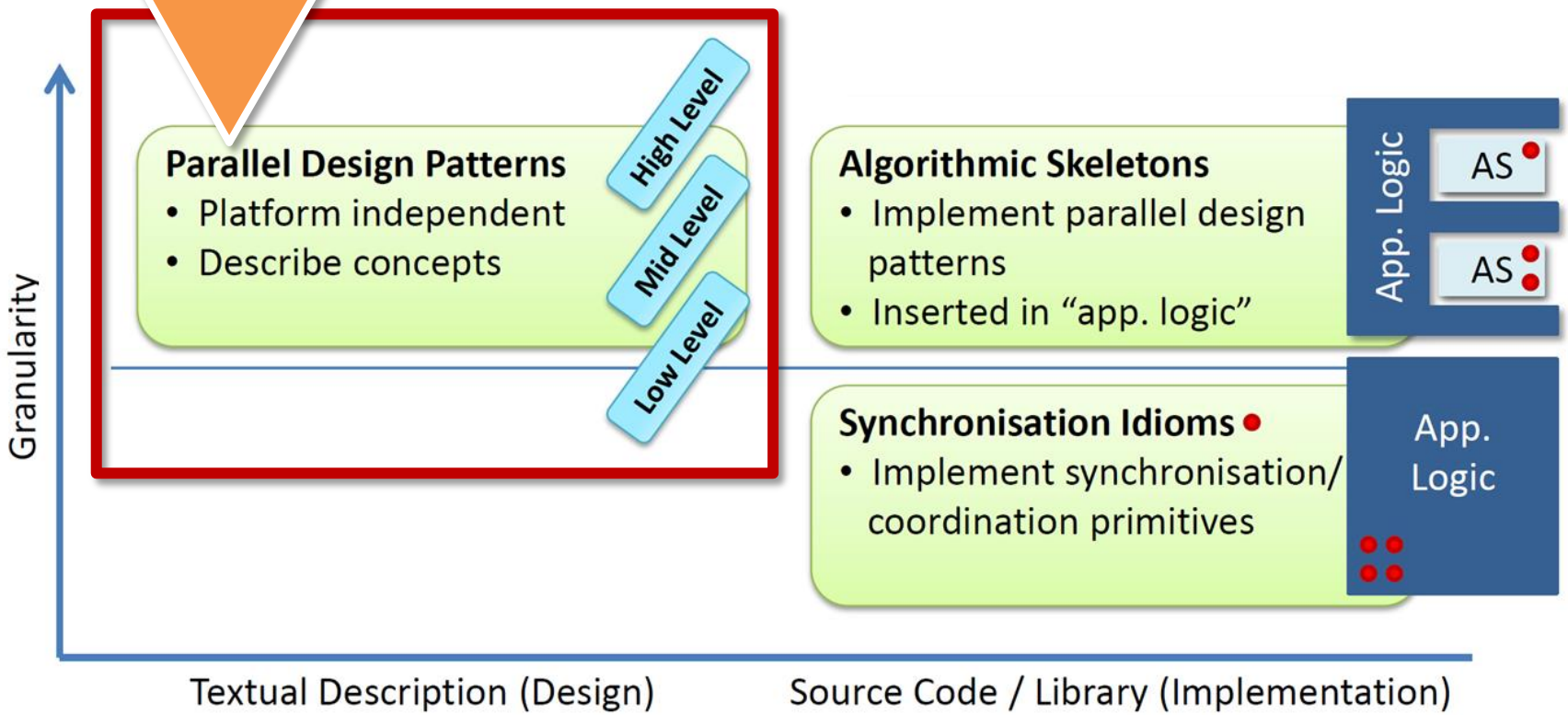
- Main challenge: interferences of threads
  - By hardware → memory, NoC, e.g.
  - By software for coordination and synchronization → locks, barriers, e.g.

- Solution:
  - Predictable hardware
    - parMERASA processor architecture
  - Predictable parallel software:
    - Analyzable synchronization primitives: ticket lock, f&i-barrier
    - Structured parallelism by parallel design patterns
    - Coding guidelines for predictable sequential and parallel program parts

"The structured approach to parallelism proposes that commonly used patterns of computation and interaction should be abstracted as parameter-isable library [...]", Murray Cole

Parallel Design Patterns

- Defined structures for computation and interaction
- Describe best-practice solutions
- Enforce structured parallelization



parMERASA Pattern Catalogue

- Assessment of PDPs and SIs for determinism and timing-analyzability:
  - PDPs only with defined SIs, defined synchronization points
  - Static mapping of tasks/code to cores
  - Little or no usage of (shared) dynamic data structures

- Description of PDPs and SIs with extended meta-patterns in parMERASA Pattern Catalogue:
  - Synchronization Idioms
  - Real-time prerequisites
  - WCET Hints

[M. Gerdes, R. Jahr, and T. Ungerer. parMERASA pattern catalogue: Timing Predictable Parallel Design Patterns. Technical Report 2013-11, Department of Computer Science, University of Augsburg, Augsburg, Germany, 2013.]

PDPs in Applications

Application	Task Parallelism	Periodic Task Parallelism	Data Parallel	Parallel Pipeline
HON: 3DPP			X	X
HON: Stereo-Nav			X	X
BMA: Crawler Crane	X	X		
DNDE: Diesel EMS	X	X		

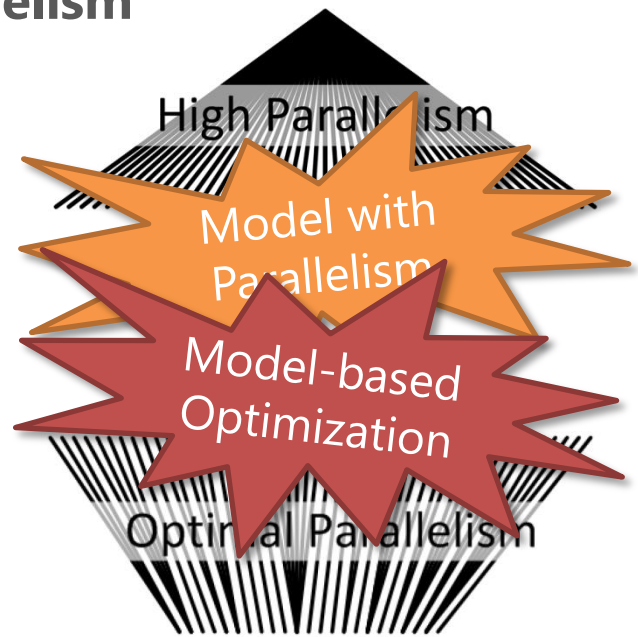
- Synchronization Idioms: ticket lock and f&i-barrier
  - Fairness
  - Analyzability

How to get Parallelism with PDPs?

- Parallelization of existing sequential single-core code:
  - *Pattern-supported parallelization approach*
  - Experience shows: simple PDPs are enough
- Starting from scratch:
  - Reveal parallelism in whole software development process
  - PDPs might be from higher levels
- Modelling by Activity and Pattern Diagram (APD): Extended UML2 Activity Diagram with PDP nodes

Parallelization Approach

- **Starting point:**
  - Single-core program ("legacy code")
  - Pattern Catalogue with predictable PDPs and Sis
- **Phase 1: Targeting High Degree of Parallelism**
  - Create model to reveal parallelism: Activity and Pattern Diagram (APD)
  - Model consisting of sequential parts and PDPs
  - Platform independent
- **Phase 2: Targeting Optimal Parallelism**
  - Agglomeration of nodes, definition of parameters
  - Platform dependent

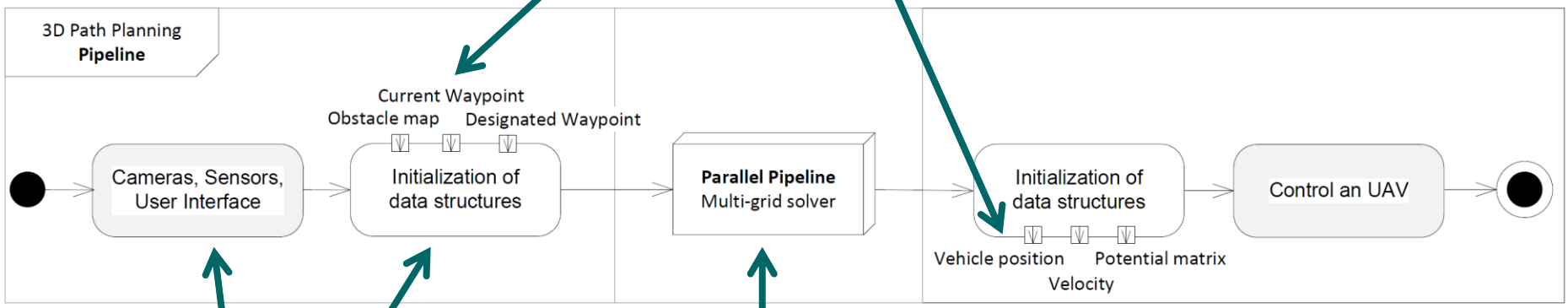


[R. Jahr, M. Gerdes, and T. Ungerer. A pattern-supported parallelization approach. In Proceedings of the 2013 International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM '13, Shenzhen, China, 2013. ACM.]

Activity and Pattern Diagram (APD)

Parallel Pipeline with three stages

Pins for reading/writing shared data



From HON 3DPP

Sequential activities

Parallel Design Pattern

- Derived from UML2 Activity Diagram:
  - PDP as additional node type
  - Fork/join operator not allowed -> structured parallelism only
- Activities/PDPs can encapsulate a single/multiple APD(s)

Impact on Timing Analysis

- Precondition: Analyzable sequential single-core code
- Only defined set assessed Parallel Design Patterns (PDPs) and synchronization idioms (SIs):
  - Static mapping of tasks to threads and cores
  - Only defined synchronization idioms: f&i-barrier, ticket locks
  - Known synchronization points
  - Limited usage of (shared) dynamic data structures
  - No indeterminism by race conditions
- Conclusion: Parallel software should also be analyzable

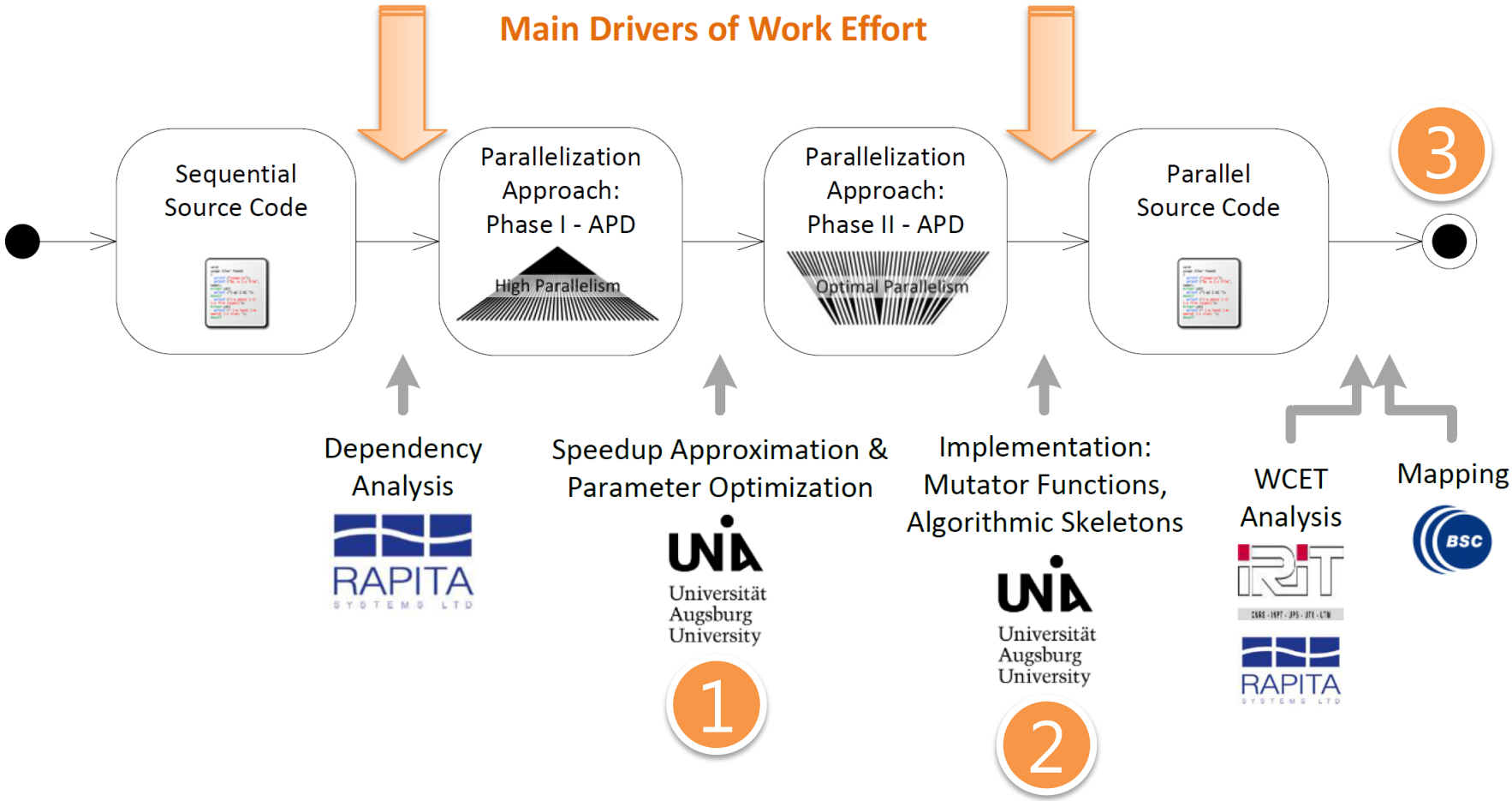
- But: Implementation of PDPs not fix, analyzability only defined for source → Timing Analyzable Algorithmic Skeletons (TAS)

Effects of PDPs on WCET Analysis

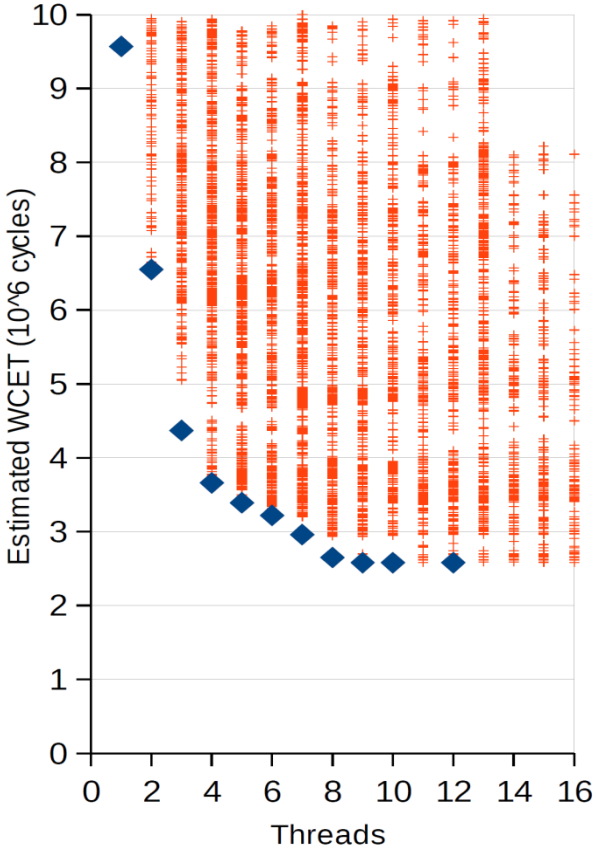
- Case-study: HON 3DPP
- PDPs out of parMERASA Pattern Catalogue enforce use of analyzable SIs only
- Comparison of existing parallel implementation with implementation based on PDPs
  - Smaller WCET preparation effort with PDPs
  - Comparable ACET performance of both implementations
  - Better WCET performance of implementation based on PDPs

[R. Jahr, M. Gerdes, T. Ungerer, H. Ozaktas, C. Rochange, and P. G. Zaykov. Effects of structured parallelism by parallel design patterns on embedded hard real-time systems. In 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Chongqing, China, 2014.]

Assistance for Parallelization



Model-based Optimization



1

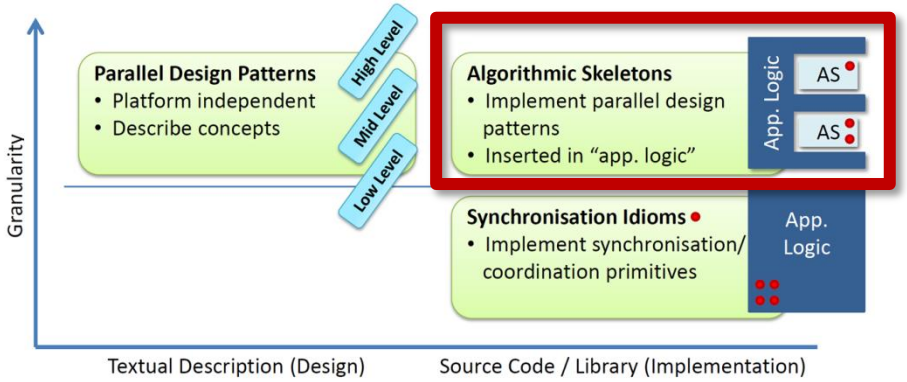
- Idea:
  - Find best assignment of threads to PDP instances
  - "Performance model" for parMERASA
  - Objectives: Minimize number of threads, Minimize WCET estimation, minimize accesses to/number of shared variables
- Implementation:
  - Optimization with Java and SMPSO
  - Input: XML-Format briefly describing APDs and list of accesses to variables
  - Case study: BMA crawler crane control code

[R. Jahr, M. Frieb, M. Gerdes, and T. Ungerer. Model-based Parallelization and Optimization of an Industrial Control Code. In Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme X, 2014]

Timing-analyzable Algorithmic Skeletons (TAS)

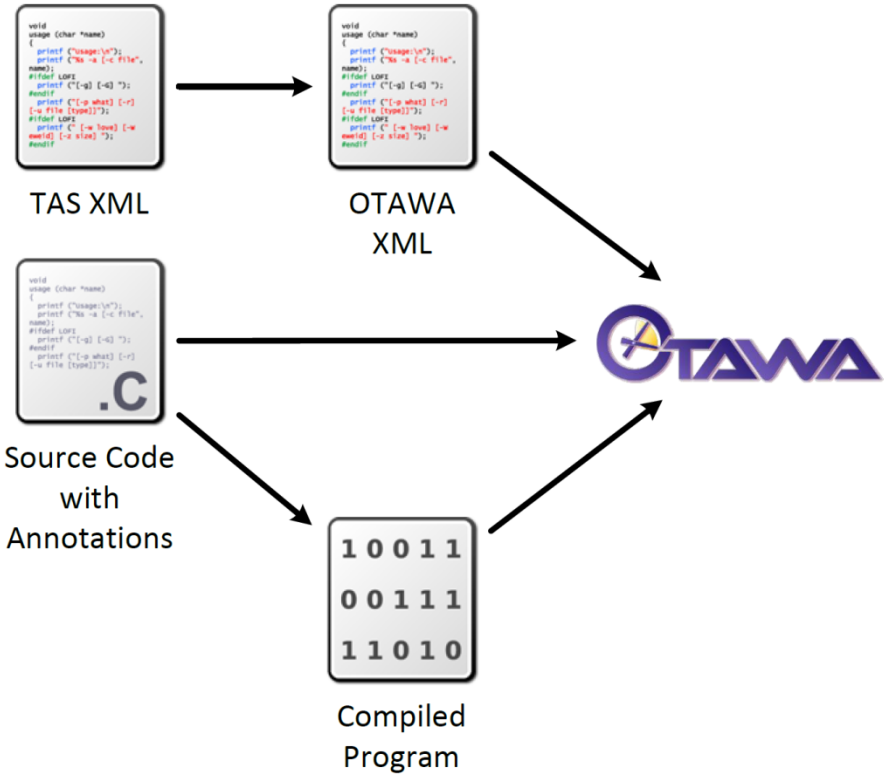
2

- Implementation of PDPs as library (pthreads and parMERASA): Data Parallel, Task Parallelism, Pipeline
- Parameterized implementation
- Effects:
  - Less effort for parallelization
  - Known implementation of PDPs improves WCET
  - Strong positive effects on software maintainability!



Support for Timing Analysis in TAS: Support for OTAWA

- Observations:
  - Parallel code completely hidden in skeletons
  - WCET analysis: IDs mainly necessary in TAS code, exception: function call for skeleton execution
- Simplification for OTAWA:
  - Compact TAS XML format describing TAS instances
  - Translator to OTAWA XML



Effectiveness of Parallelization

3

- Embedded systems must guarantee defined deadlines, even on multi-core → "ASAP" is not goal!
- Instead: do more or with better results, consume less power, have more regular update intervals

- Suggested measures:
  - Slack time: potential to compute more ← Speedup!
  - Jitter in Update intervals
  - Possible reduction of CPU clock rate

Open Source Software

https://github.com/parmerasa-uau/

LGPL Free Software
Free as in Freedom

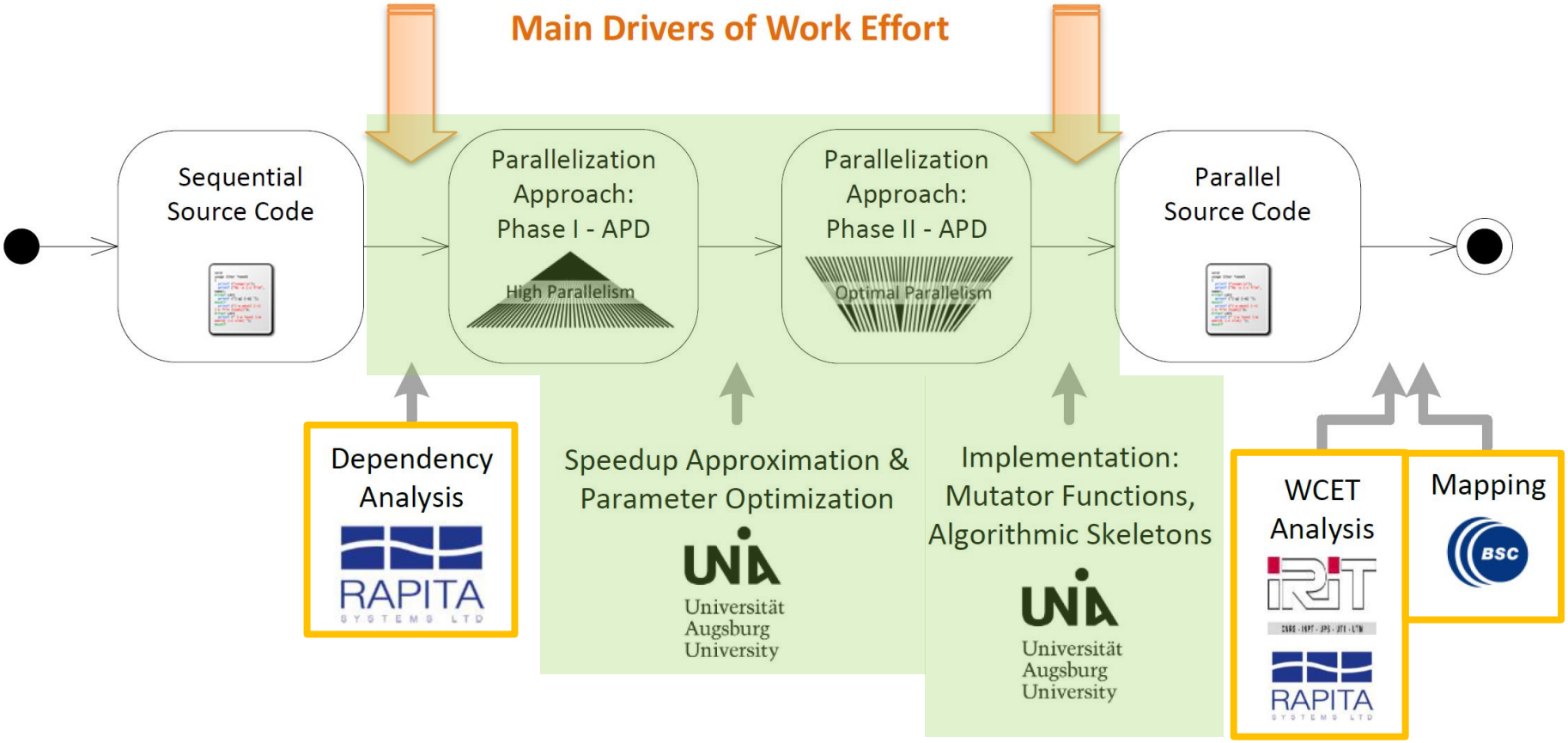
- Timing Analyzable Algorithmic Skeletons (TAS)
  - Predictable PDPs implemented in C with pThreads
  - Support for timing analysis with OTAWA
- Model-based Optimization
  - Find best assignment of threads to PDP/TAS instances
  - Multi-objective optimization with SMPSO
- Further information:

[R. Jahr, A. Stegmeier, R. Kiefhaber, M. Frieb, and T. Ungerer. User Manual for the Optimization and WCET Analysis of Software with Timing Analyzable Algorithmic Skeletons. Technical Report 2014-05, Department of Computer Science, University of Augsburg, Augsburg, Germany, 2014.]

Summary

- Structured Parallelism by **Parallel Design Patterns (PDPs)** and analyzable **Synchronization Idioms (SIs)**
- Predictable PDPs and SIs in **Pattern Catalogue**
- **Parallelization approach** with UML modelling and PDPs
- **Support of Parallelization** by skeletons and tools
- Such structured parallelism leads to **timing-analyzable parallel code**, **less effort for WCET analysis** and **better WCET estimates**

Conclusion



- Path from sequential to parallel software covered well
- Parallelization is feasible also with support of WCET analysis
- Tool-support is important (analysis, optimization, implementation)