



A Pattern-supported Parallelization Approach

Ralf Jahr, Mike Gerdes, Theo Ungerer
University of Augsburg, Germany

The 2013 International Workshop on Programming Models and
Applications for Multicores and Manycores (PMAM 2013)



- Motivation
- Parallel Design Patterns
- Pattern-supported Parallelization Approach
 - Two phases
 - Activity and Pattern Diagram
 - Pattern Catalogue
- Case study: Unmanned Aerial Vehicle
- Summary

- Multicore and manycore CPUs in embedded systems
- Goals:
 - Faster execution of a workload
 - Concurrent execution of multiple tasks
 - Shorter reaction times
 - Energy savings because of lower clock frequency

→ Need for parallel applications

- But, especially for embedded systems:
 - Much legacy code
 - Limited development resources
 - Complicated testing and debugging

A starburst graphic with a jagged, orange border and a white center, containing the text "Parallelization Approach".

Parallelization
Approach



- Design Patterns
 - Idea initially in architecture
 - Recurring problems → best practice solutions
 - Transfer to software engineering
 - Mainly object oriented design, see “Gang of four”
 - Standardized description: Pattern Catalogue

- Parallel design patterns
 - Extended concept: design patterns providing parallelism
 - Tradeoff: flexibility in design vs. development effort

- **Starting point:**
 - Sequential program ("legacy code")
 - Pattern Catalogue with parallel design patterns
- **Phase 1: Targeting Maximum Parallelism**
 - Create model to reveal parallelism
 - Model consisting of sequential parts and parallel design patterns
 - Platform independent
- **Phase 2: Targeting Optimal Parallelism**
 - Agglomeration of nodes, definition of parameters
 - Creation of threads and mapping onto target architecture
 - Platform dependent



Agent and Repository

Name

Agent & Repository

Problem

Context

Forces

Solution

A Pattern Language for Parallel Programming ver2.0

Read me first! → Our Pattern Language

(notes) (Glossary)(Blogs) (Pattern Template) (Pattern Abstract) (Pattern Workshop) (Pattern in Education)

Applications

Structural Patterns	Computational Patterns
Agent and Repository	Backtrack Branch and Bound (notes)
Pipe-and-filter	Monte Carlo Methods (notes)
Process Control	N-Body Methods
Arbitrary Static Task Graph	Circuits (notes)
Iterative_refinement	Sparse Linear Algebra
Event-based, implicit invocation	Dense Linear Algebra (notes)
Layered systems	Dynamic Programming doc (notes)
Map reduce	finetestatemachine.pdf
Model-view controller	Graph Algorithms
	Graphical Models
	Structured Grids (notes)
	Unstructured Grids
	Sorting

Parallel Algorithm Strategy Patterns

Task Parallelism (notes)	Discrete Event	Geometric Decomposition (notes)	Non-work-efficient Parallelism
Recursive splitting (notes)	Pipeline doc (notes)	Data Parallelism	Speculation

Implementation Strategy Patterns

SMPD	MasterWorker	sharedqueue.pdf	Distributed Array
Strict-data-par	LoopParallelism	Shared Hash Table	shared data
ForkJoin	BSP		memory parallelism (notes)
Actors	Task Queue		
Graph Partitioning (notes)			

(Program Structure) (Data Structure)

Concurrent Execution Patterns

MIMD	Thread Pool	Message Passing	P2P Sync
Task Graph	Speculation	Collective Communication	Collective Synchronization
SIMD	Data Flow	Mutual Exclusion	Transactional Memory
DigitalCircuits (notes)			

(Advancing Program Counters) (Coordination)

(Pattern v1.0) (Pattern v2.0)

All contents on this website are by OPL Working Group, available under a Creative Commons Attribution 3.0 Unported License. Copyright © 2008-2010, OPL Working Group.

patterns/patterns.tot - Last modified: 2010/07/15 01:23 by vivat

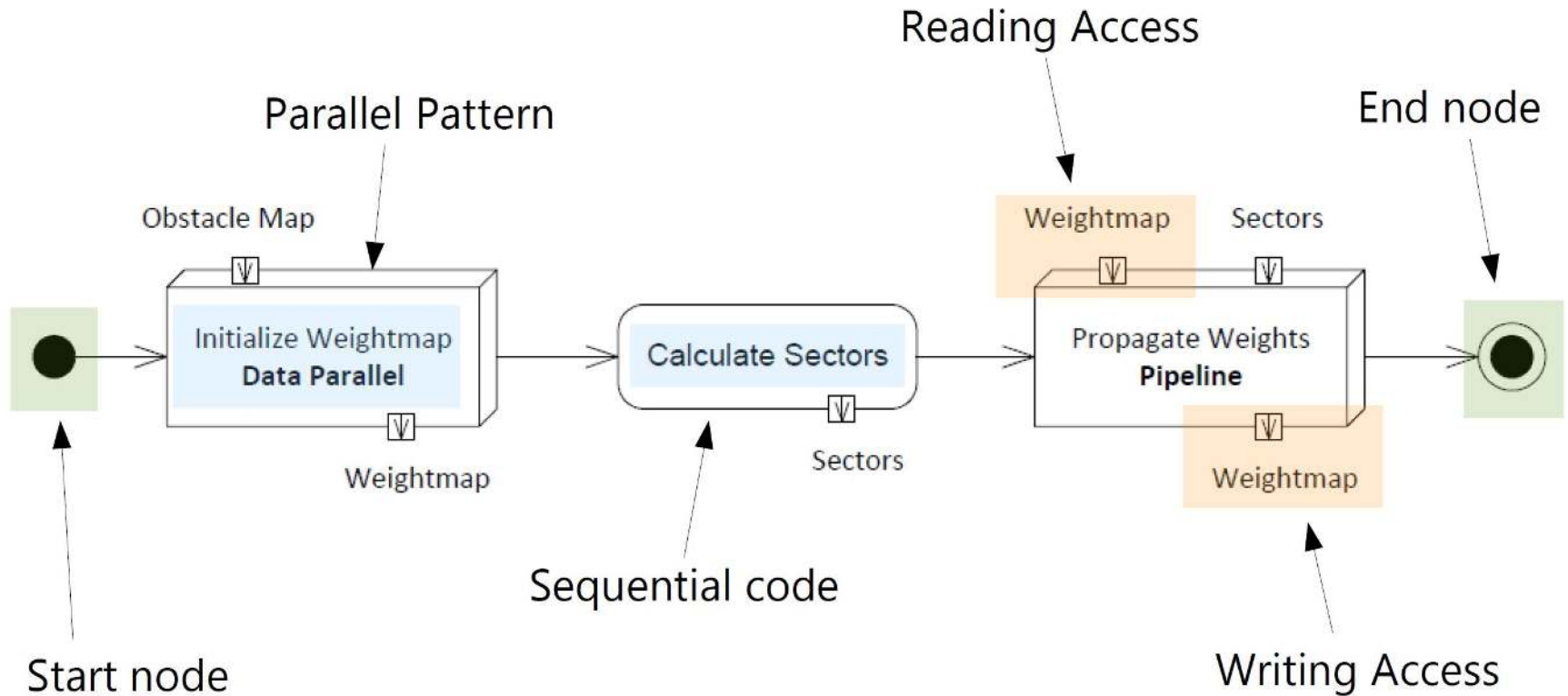
Show pagesource Old revisions Media Manager Login Sitemap Back to top

- 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

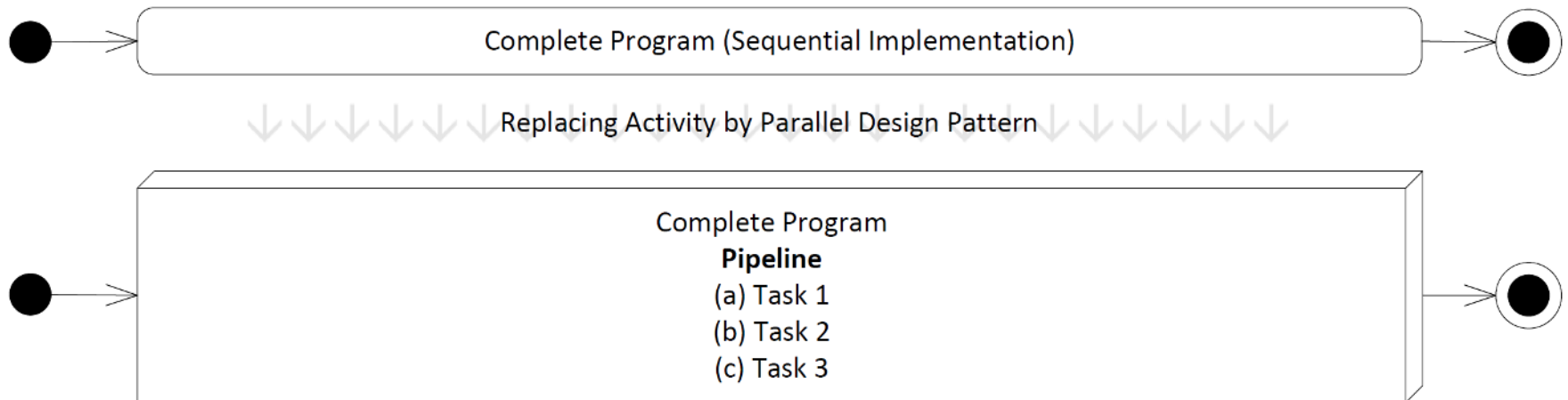
- Example: “Our Pattern Language”
 - <http://parlab.eecs.berkeley.edu/wiki/patterns/patterns>
 - Organized in multiple layers



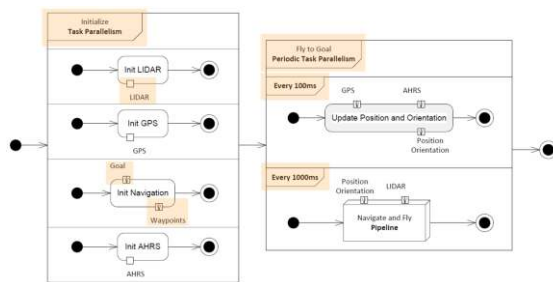
- Extension of UML2 Activity Diagram:
 - *Parallel design pattern* is new node type similar to activity
 - Activities: either sequential or encapsulate APD
 - Parallel design patterns: Multiple activities in parallel
- Patterns are only way to introduce parallelism
- Advantages over inventing a new notation:
 - Well known, easy to understand, tools exist
 - Support for dependencies, branches, and nesting



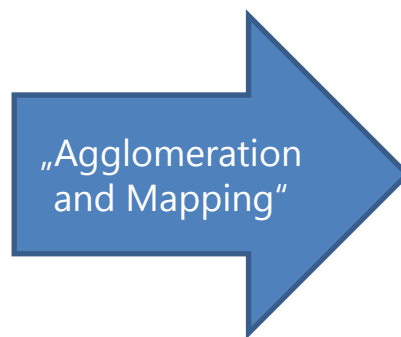
- Goal: Reveal sufficient parallelism for any platform as Activity and Pattern Diagram (APD)
- Start with single activity, repeatedly apply two operations:
 - Replacement:** apply parallel design pattern
 - Splitting:** decompose into multiple activities



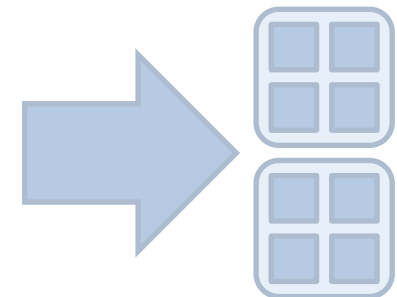
- Transition from maximum to optimal parallelism by agglomeration
- Similar to optimization problem:
 - Global Objective: reduce execution time, energy consumption, ...
 - Execution time influenced by e.g. communication/computation ratio, cost for synchronization, etc.
 - Side conditions: number of available cores/threads; dependencies (control, data, timing), etc.



Activity and Pattern Diagram

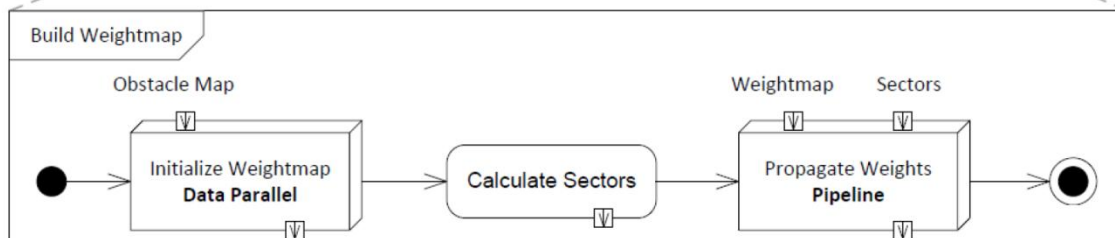
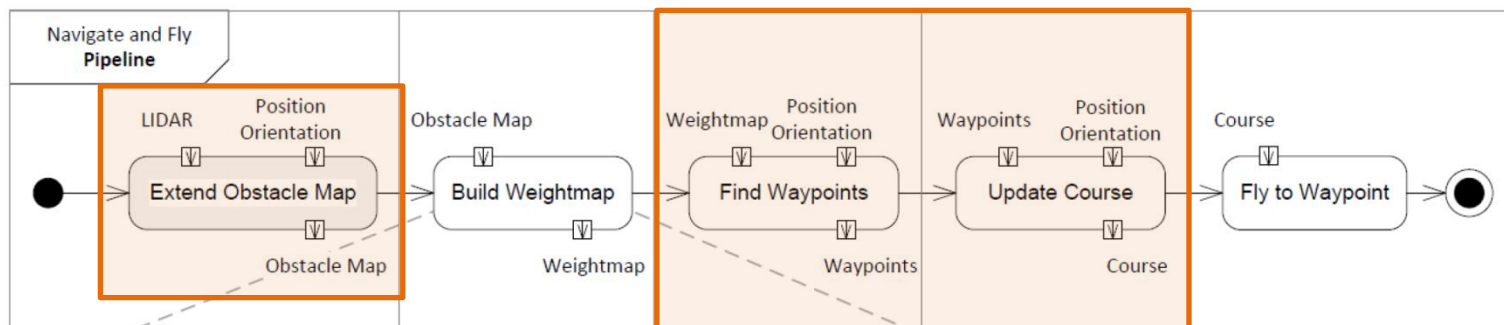


Threads/Tasks



Cores

- Agglomeration is...
 - **Replacing a parallel design pattern** by an activity, e.g., replacing pipeline by activity → Reduction of parallelism
 - **Joining elements** of parallel design pattern, e.g., multiple pipeline stages to single one → Reduction of overhead
 - **Defining parameters**, e.g., concurrent workers for data parallelism → Tailoring design patterns to target platform



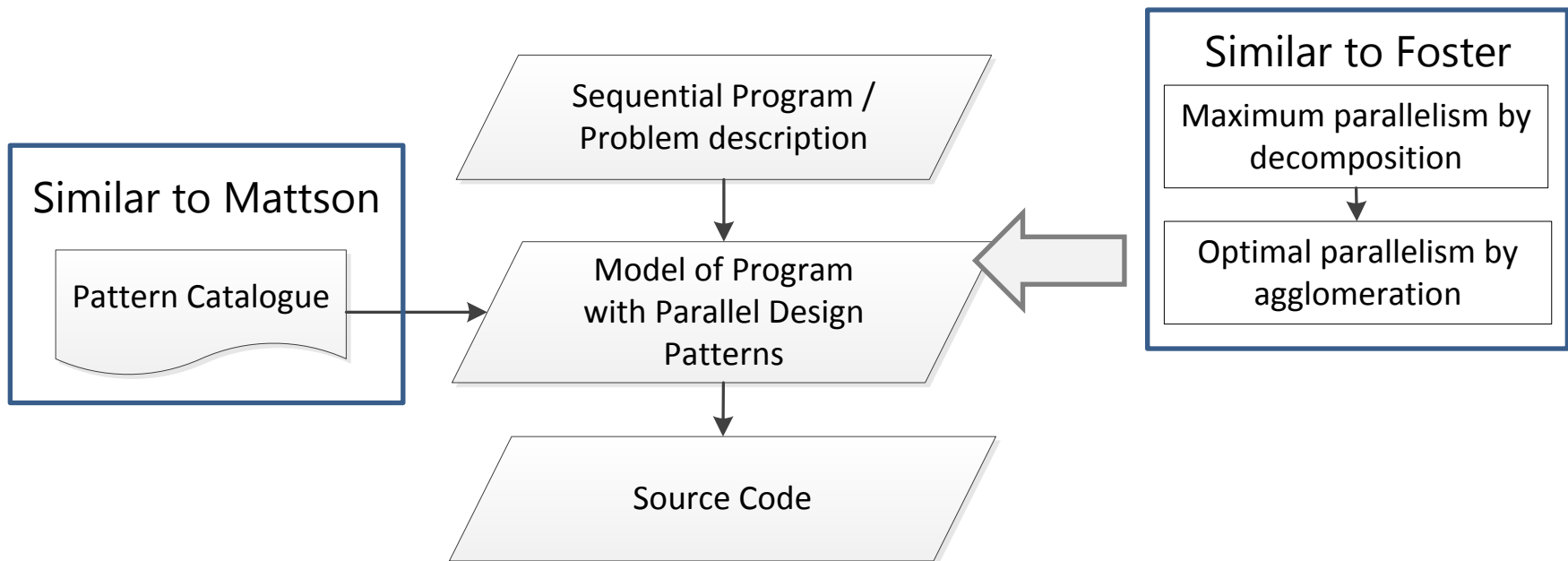


Pattern-supported Parallelization Approach: Phase 2

- Mapping
 - Find optimal mapping between code (APD) and threads/tasks and cores/clusters
 - Trade-off between optimal use of resources vs. parallelism
 - Not in focus of parallelization, different research area

- Objectives for parallelization process
 - Speedup/rough approximation of speedup
 - Resource usage
 - Energy consumption
 - Implementation effort (e.g. number of patterns)

- If necessary: iterative application of process!



- Manual process with clear methodology
- Fast modelling of parallelism with Activity and Pattern Diagram; derived from UML2
- Pattern Catalogue
 - Easier implementation of parallel program
 - Higher Documentation Quality
- Algorithmic skeletons for reduced implementation effort

Example & Work in Progress: Unmanned Aerial Vehicle (UAV)





- Autonomous flight over terrain
 - Obstacle detection
 - Automatic path planning (Laplace operator)

- Assumptions:
 - Sequential software exists

- Overview of the software:
 - Initialize system
 - Loop until goal is reached:
 - Determine position
 - Mark obstacles
 - Plan path
 - Set course

- Phase 1
 - Goal: Expose parallelism
 - Finished, see paper
 - Six instances of parallel design patterns

- Phase 2
 - Goal: Tailor parallelism to target platform
 - But: work in progress, no target platform yet defined
 - Approximated speedup based on profiling: 7.8
 - Enough parallelism for 2 to 6 cores
 - Further work necessary for 8+ cores

parMERASA

- Pattern-supported parallelization approach
 - Two phases:
 - Reveal parallelism: architecture independent
 - Agglomerate and map: architecture dependent
 - Only **parallel design patterns** to introduce parallelism
 - Parallel design patterns are described in **Pattern Catalogue**
 - Supporting structure: **Activity and Pattern Diagram**, similar to UML2 Activity Diagram
 - Limited effort for parallelization and implementation of parallel program
- Future work:
 - Tool support for parallelization, especially Phase 2
 - Extend parallelization process for hard real-time systems
 - More case studies, different platforms → gain knowledge