

Kick-Off
Proseminar Sommersemester 2014
Parallelitätsmuster

René Schöne
Max Leuthäuser

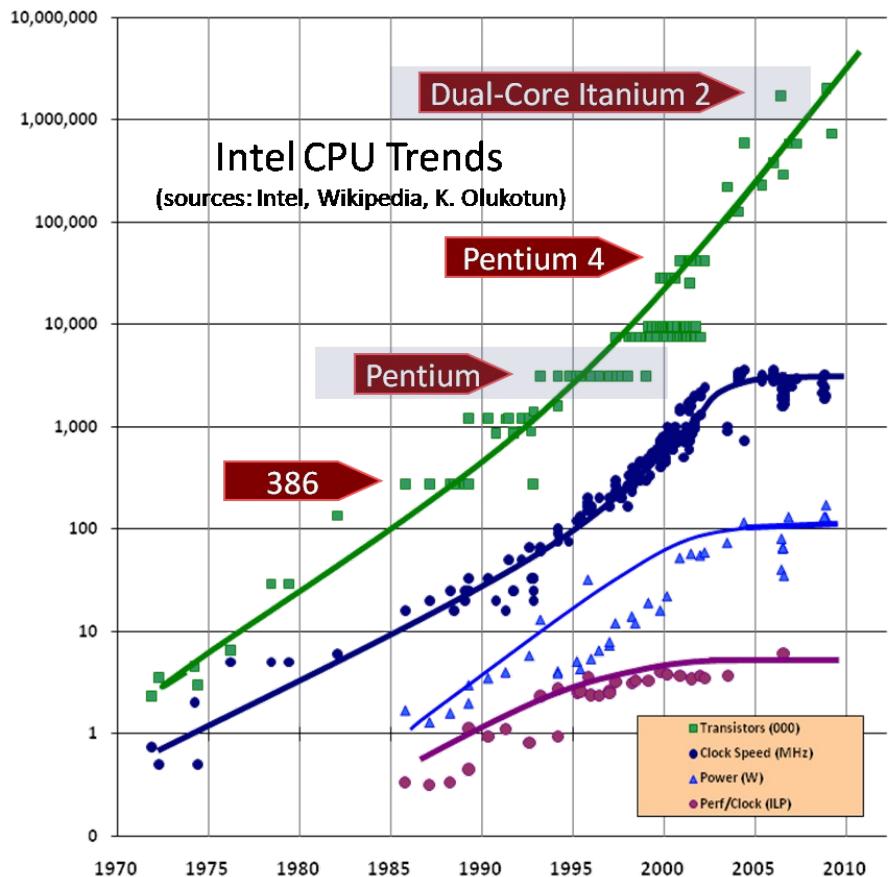


Abbildung 1: Intel CPU Introductions [2]

„The complexity for minimum component costs has increased at a rate of roughly a factor of two per year.“ – G. Moore [1]

- Physikalische Grenze der Prozessorgeschwindigkeit seit etwa 2005 erreicht
 - Taktfrequenz
 - Leistung (Energieverbrauch)
- Aktuell Mehrkern- und Co-Prozessorarchitekturen
 - Mehrkern CPU's
 - CPU und GPU in einem Chip

Moore's Law gilt weiterhin!

„Software is getting slower more rapidly than hardware becomes faster.“ - N. Wirth [3]

- Programme profitieren
 - von schnelleren Prozessoren
 - von Optimierungen des Compilers für bestimmte Prozessortypen
- Programme profitieren **nicht**
 - von mehr Kernen im Computer
 - von Spezial-Prozessoren (z.B. GPUs)
... wenn nicht direkt für Zielarchitektur gebaut
- Nicht jedes Programm ist parallelisierbar [4]
- Parallelisierung unentscheidbar für Compiler

**Programmiersprachen müssen Konzepte bereitstellen,
um paralleles Programmieren zu ermöglichen!**



Lassen sich Muster finden, welche die Entwicklung von parallelen Anwendungen vereinfachen?

- Wann und wo?
- Welche Architektur / Welches Design der Software?

Wie muss parallele Software gebaut werden?

- **University of Berkley:** OPL (Our Pattern Language), [6].
 - Konzentration auf höhere Abstraktionsebenen:
 - Structural Patterns
 - Computational Patterns
- **University of Illinois:** Parallel Programming Patterns, [7].
 - Konzentration auf niedriges Abstraktionsniveau:
 - Concurrent Execution Patterns
- **Mattson, Sanders, Massingill:** PLPP (Pattern Language for Parallel Programming), [8]
- **Illinois:** PLoP (Pattern Languages of Programs), [9]
 - Workshop

<u>Structural Patterns</u>		<u>Computational Pattern</u>	
Pipe-and-Filter	Moder-view-controller	Graph Algorithmus	Graphical models
Agent Repository	Iterative Refinement	Dynamic Programming	Finite state machines
Process Control	Map Reduce	Dense Linear Algebra	Backtrack, Branch and Bound
Event-based, implicit invocation	Arbitrary static task graph	Sparse Linear Algebra	N-Body methods
Puppeteer	Layered Systems	Unstructured Grids	Circuits
		Structured Grids	Spectral Methods
			Monte Carlo

Algorithm-strategy Patterns

Task Parallelism	Data Parallelism	Discrete Event	Speculation
Recursive splitting	Pipeline	Geometric Decompression	

Patterns aus [6]

Implementation strategy Patterns

		<u>Program structure</u>	<u>Data structure</u>	
SPMP	ForkJoin	Loop-Par.	Shared Queue	Distributed Array
Strict data parallelism	Actors	BSP	Shared Hash Table	Shared Data
Graph partitioning	Master/ Worker	Task Queue		

Parallel execution Patterns

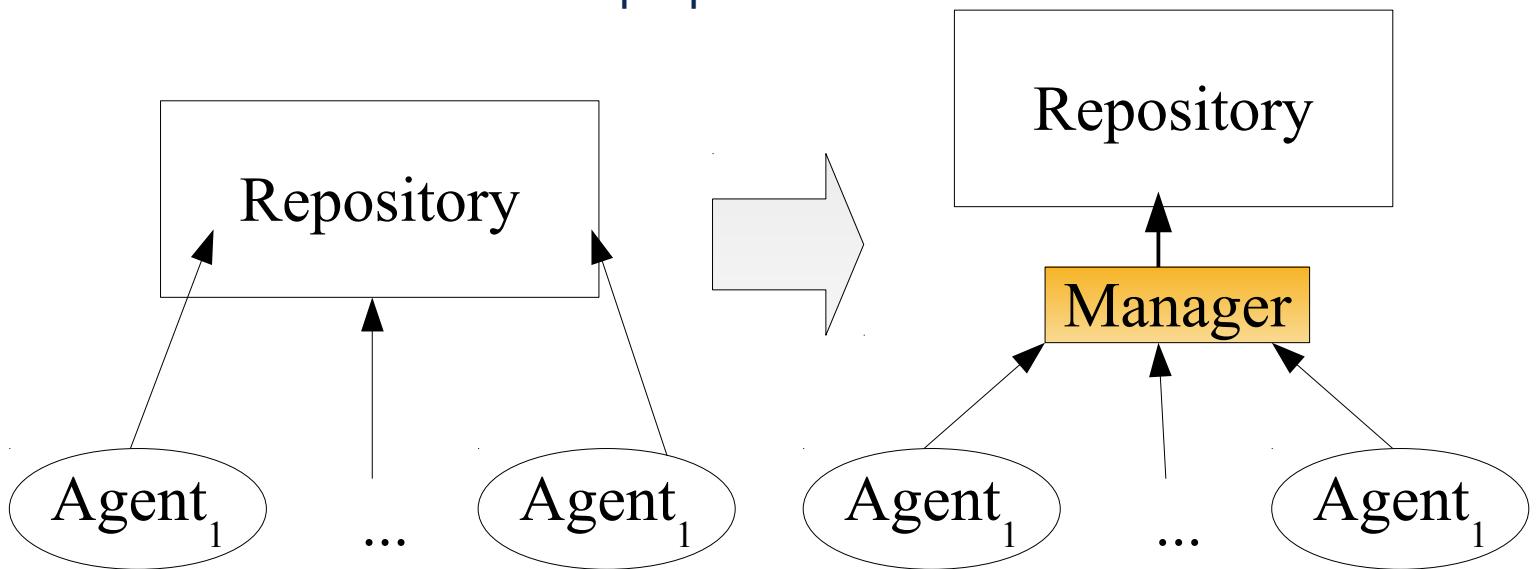
		<u>Advancing Program Counters</u>	<u>Coordination</u>	
MIMD	Thread Pool	Task Graph	Message Pass	Point-to-point synchronisation
SIMD	Speculation	Data flow	Collective communication	Collective synchronisation
		Digital circuits	Mutual exclusion	Transactional Memory

Patterns aus [6]

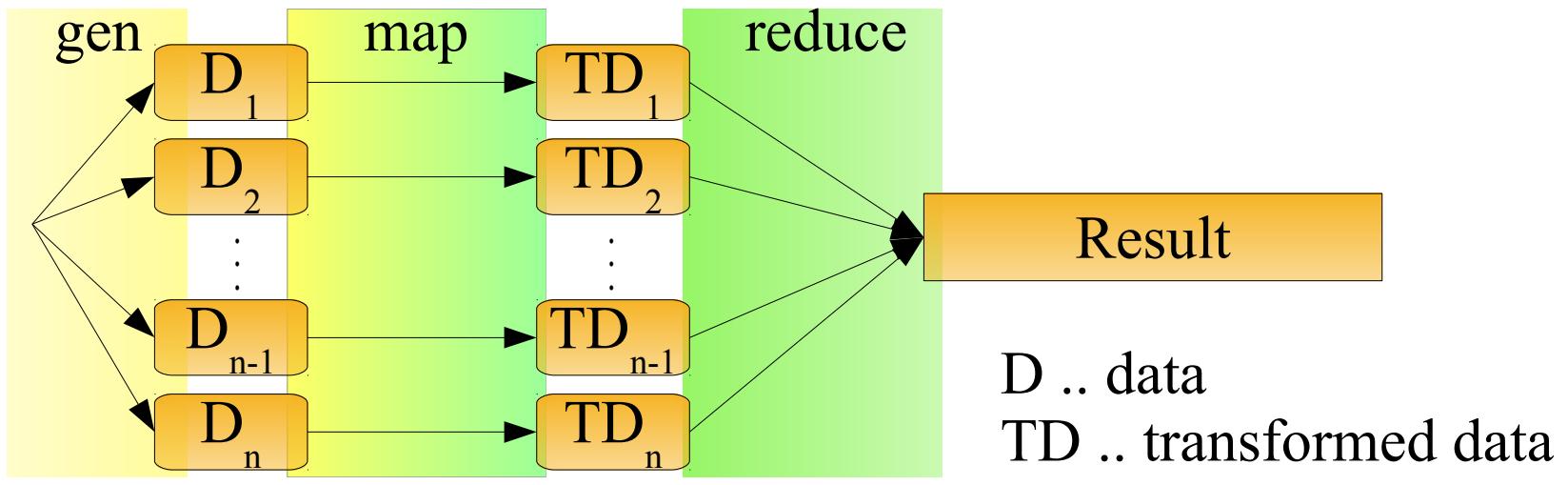
- **Pipe-and-Filter:** Divide set of transformation steps into filter (transformation) and pipes (connection)
- **Agent and Repository:** Consistent access to repository using dedicated manager
- **Iterative refinement:** High-level steps having multiple independent tasks
- **Process Control:** Dynamic control of physical environment using a feedback loop using Sensor, Actuator and Model
- **Map-Reduce:** Large number of independent computations and result collection
- **Event-based, implicit invocation:** Differentiation of intent to take action and implementation of action
- **Layered Systems:** Divide functionality into layers with different level of abstraction

Patterns aus [6]

- Problem: Ensure consistent and efficient accesss of agents to data (repository)
- Solution: Introduce manager for accessing (e.g. with locks, local locks, versioning)
- Relates to DB and ACID properties



- Problem: Large number of independent computations and collection of results
- Solution: 2 distinct phases (Map and Reduce)
 - Domain expert only has to write map and reduce function, not bother with parallelization
- Difficult: load balancing in both phases, data (i.e. result) placement (in a repository or locally in worker)



- Auswahl und Beschreibung eines Patterns
- Implementierung an einem ausgewählten Beispiel
- **Nächste Woche** (5 Minuten Vortrag):
 - Patternauswahl
 - Programmiersprache (Empfehlung: JVM-basiert)
 - Beispiel-Skizze
- **Präsentation** (30 Minuten Vortrag):
 - Patternbeschreibung
 - Programmiersprache: spezielle Features bzgl. Parallelität
 - Implementierung des Beispiels
- **Schriftliche Ausarbeitung:**
 - 4-5 Seiten (Empfehlung: Latex, Style: LNCS)

Nächster Termin 24.04.2014

- Patternauswahl
- Programmiersprache (Empfehlung: JVM-basiert)
- Beispiel

Zwischentreffen

- evtl. Zwischenfragen
- Status-Update
- Präsentations-Tutorial

Blockveranstaltung am Ende des Semesters

- Vortrag (30 Minuten)

Fusionforge

- SVN/Git
- Abgabedokumente etc.

[1] **Cramming more Components onto Integrated Circuits**

Gordon E. Moore

Electronics Magazine (1965)

[2] **The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software**

Herberd Sutter

Dr. Dobb's Journal (2005)

[3] **A Plea for Lean Software**

Niklaus Wirth

Computer 28.2 (1995)

[4] **Limits to Parallel Computation: P-completeness Theory**

Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo.

Oxford university press (1995)

[5] **The Role Object Pattern**

Dirk Bäumer, et al.

Washington University Dept. of Computer Science (1998)

[6] **Parlab Bibliothek**

OPL Working Group,

A pattern language for parallel programming ver2.0, Berkeley University of California June 2010

[7] **Parallel Programming Patterns**

<https://wiki.engr.illinois.edu/display/ppp>

Pattern Library

[8] **Patterns for Parallel Programming**

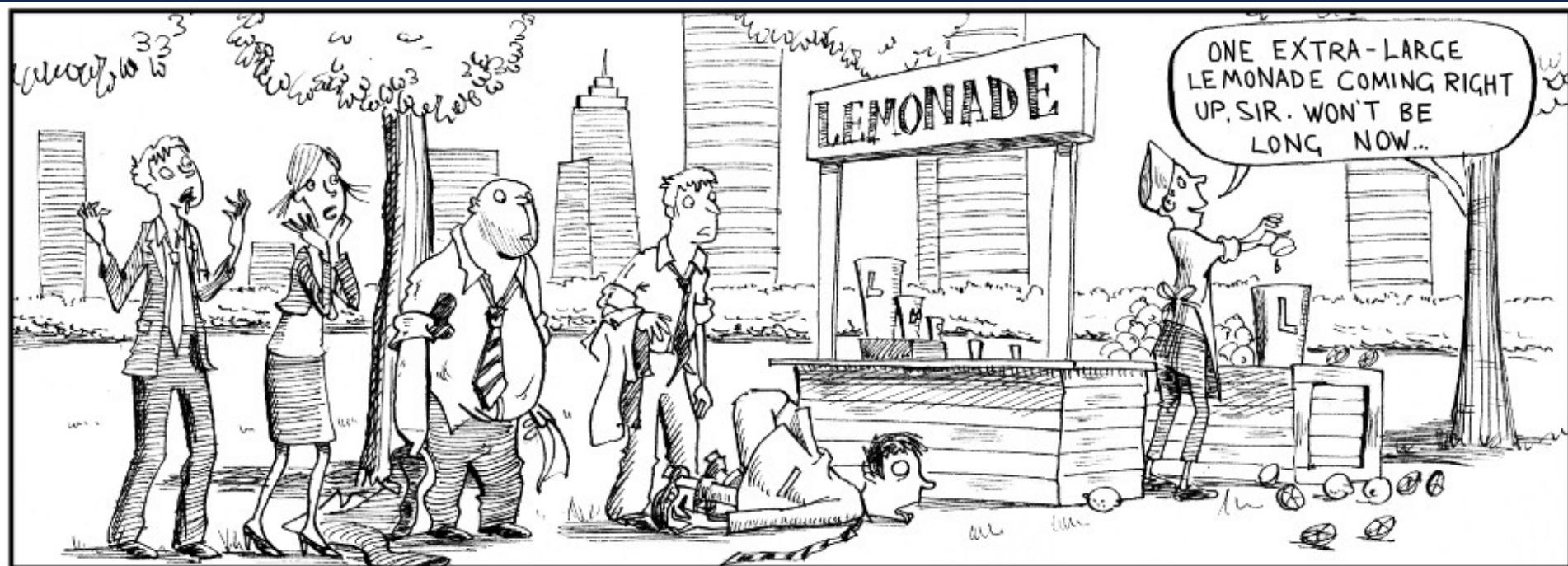
Timothy G. Mattson; Beverly A. Sanders; Berna L. Massingill

Pearson Education (Amsterdam)

[9] **Conference on Pattern Languages of Programs (PLoP)**

<http://www.hillside.net/plop/2014/>

Jährliche Konferenz



Ende