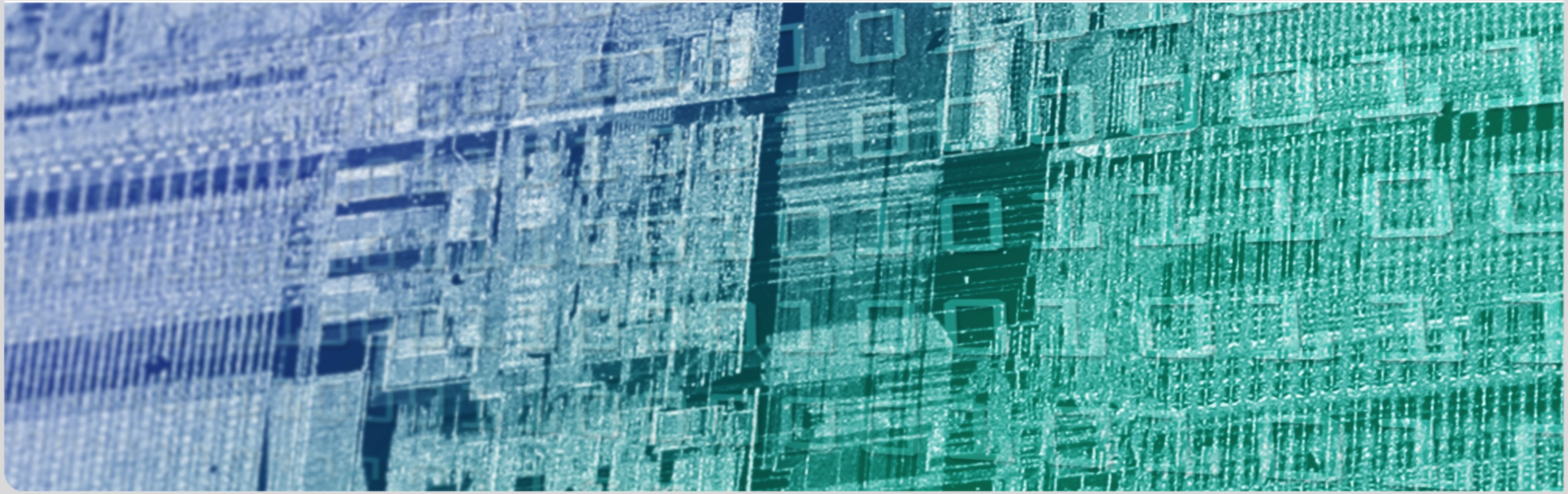
Engineering Parallel Applications with Tunable Architectures

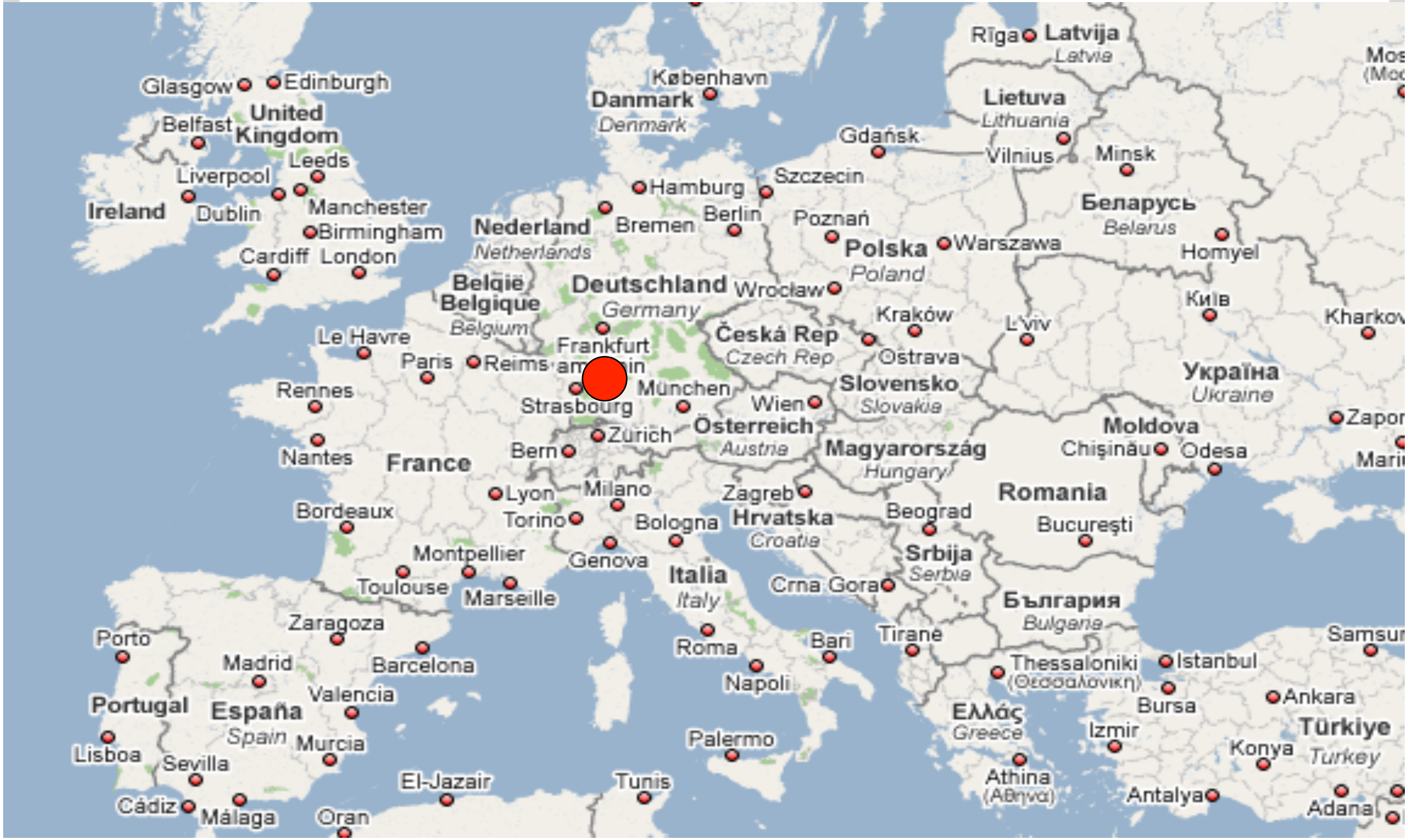
International Conference on Software Engineering, May 2010
Christoph A. Schaefer/Victor Pankratius/Walter Tichy

Institute for Program Structures and Data Organization



Where is Karlsruhe?

Karlsruhe Institute of Technology



Where is Karlsruhe?

Karlsruhe Institute of Technology

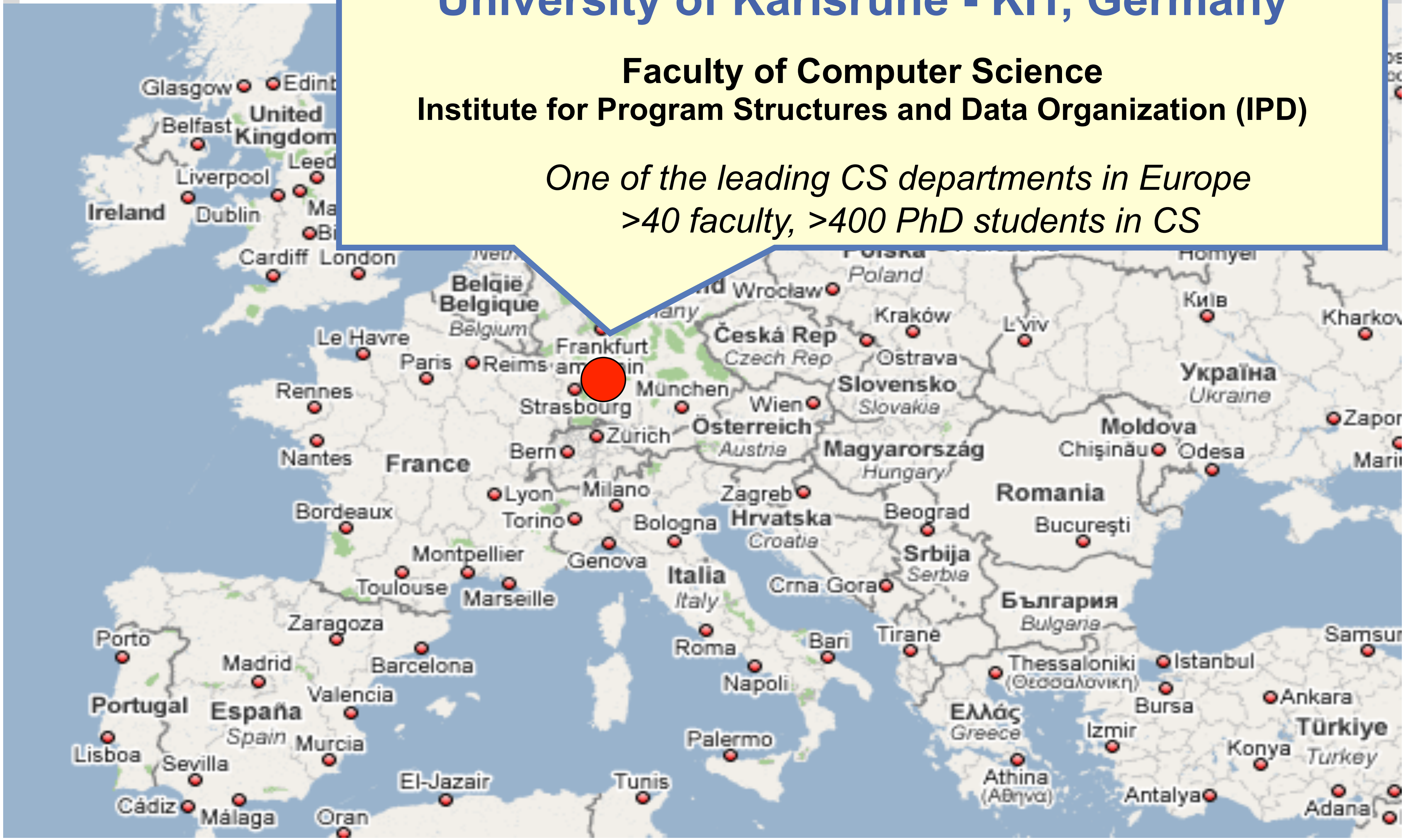
University of Karlsruhe - KIT, Germany

Faculty of Computer Science

Institute for Program Structures and Data Organization (IPD)

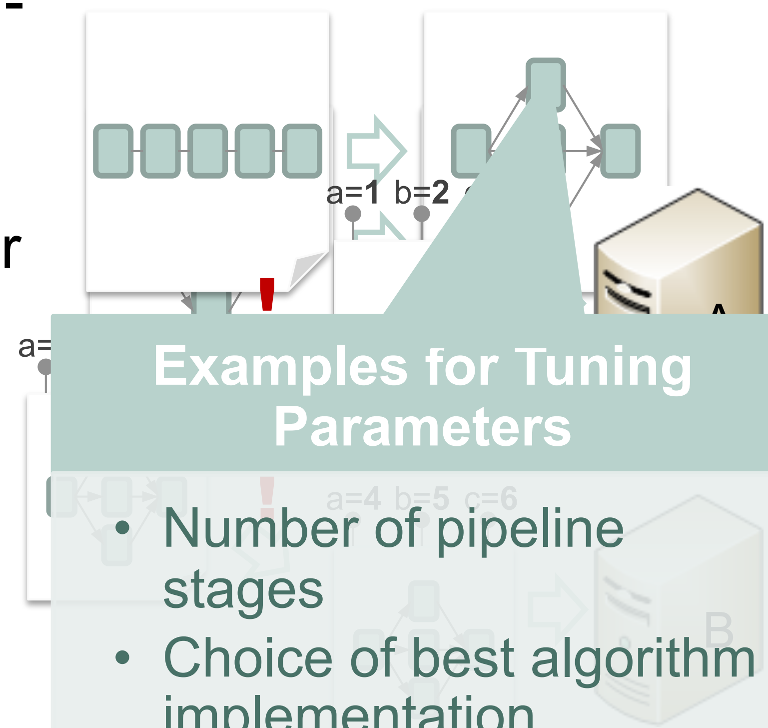
One of the leading CS departments in Europe

>40 faculty, >400 PhD students in CS



Challenges of Parallel Applications

- Parallelization is complex and error-prone
- Parallel programs contain a number of tuning parameters
- Manual optimization difficult and time-consuming
- Each target platform may require re-tuning
- **Auto-tuning**: Let the computer do the tuning!



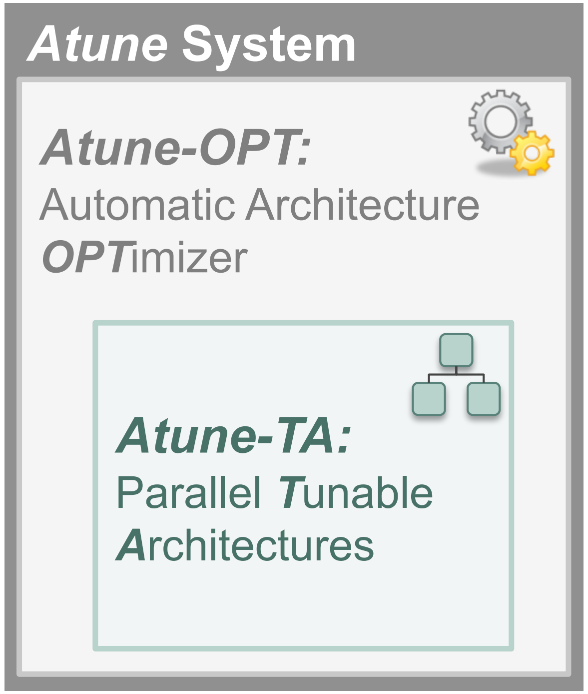
Examples for Tuning Parameters

- Number of pipeline stages
- Choice of best algorithm implementation
- Order of execution
- Size of data partitions
- Number of workers
- Load balancing strategy

Approach to Auto-Tuning

- *Atune-TA*: Approach for description of certain parallel, tunable architectures
  - Automatic implementation of architectures
  - Portability regarding performance

- *Atune-OPT*: Automatic search-based performance tuning on multi-core platforms (*auto-tuner*)
  - Not limited to specific application domain or numeric programs
  - Extension of search-based optimization to handle large parallel applications



Example: Parallel Desktop Search (Indexing)
1. Definition of Tasks

Karlsruhe Institute of Technology

- Abstraction from threads and fine-grained parallelization
- → Concept of tasks: definition of essential processing steps

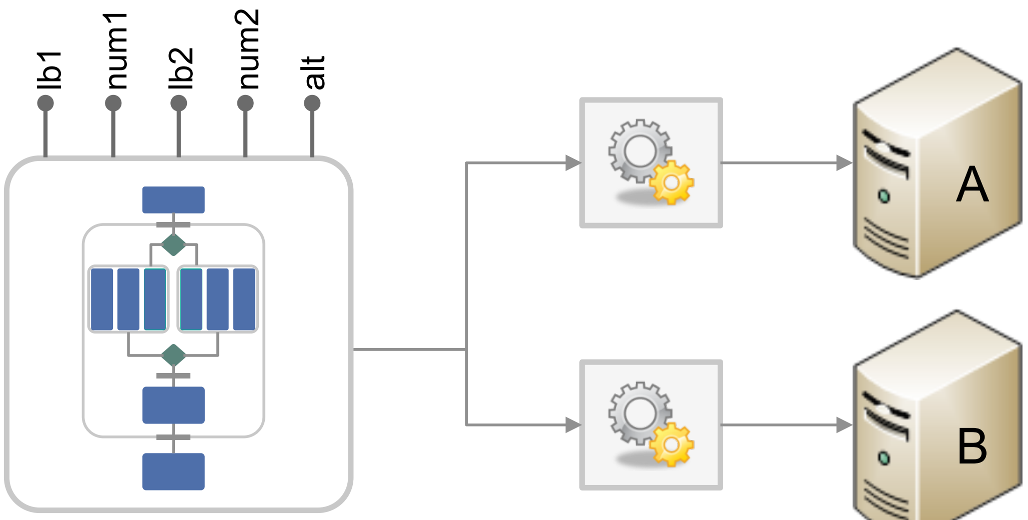
Crawl	ParseAlgo 1	ParseAlgo 2	Update Index	Create Indexfile
Save index datastructure on disk				

Methods to implement

```
• Crawl():List<string>
• ParseAlgo1(string s):ParseResult
• ParseAlgo2(string s):ParseResult
• UpdateIndex(ParseResult p):Index
• CreateIndexFile(Index i):void
```

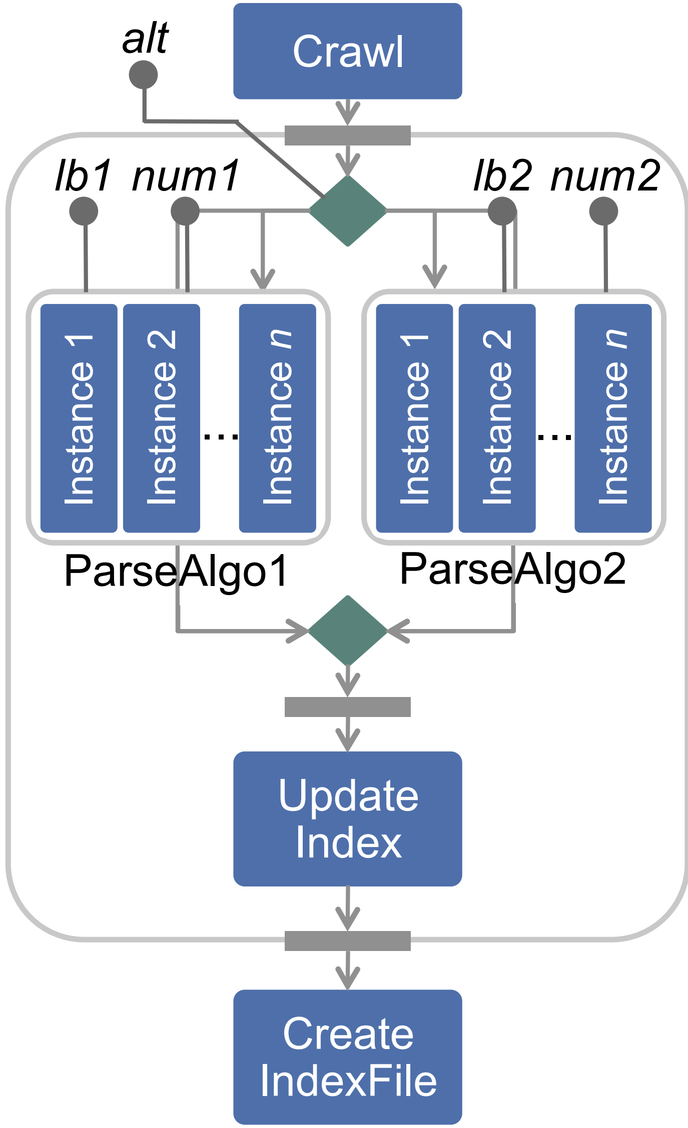
Example continued
2. Design, Implementation and Optimization

```
TunablePipeline MyDesktopSearch
[source:AC_Crawl;sink:AC_CreateIndexFile]
{
    TunableAlternative
    {
        AC_ParseAlgo2
        AC_ParseAlgo1
    },
    AC_UpdateIndex
}
```



Parameterized architecture implementation

Automatic platform-specific optimization



Atune-TA

Tunable Architecture Description Language (TADL)

- Description language for compact design of parallel tunable architectures
  - *Atomic components*
    - Represent essential sequential program tasks
    - Contain no internal parallelism, but allow replication
    - Implemented by program methods (AC methods)
  - *Connectors*
    - Connect atomic components and define processing and parallelization strategies
    - Support nesting
    - Implicitly expose predefined tuning parameters

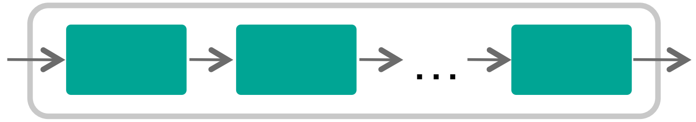
```
AC_MethodName[replicable]
```

```
TunablePipeline pipeline
{     [source:AC_InputMethod;
 sink:AC_OutputMethod]
      AC_Method1,
      AC_Method2,
      AC_Method3
}
```

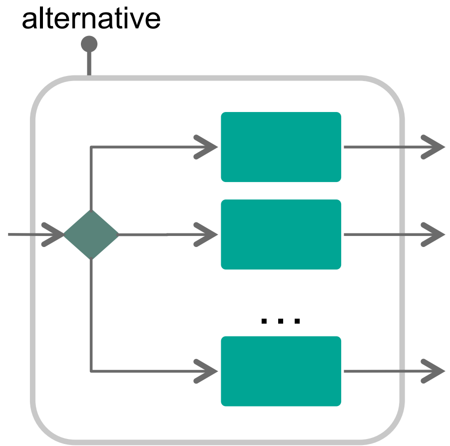

Forschungszentrum Karlsruhe in der Helmholtz-Gemeinschaft
Universität Karlsruhe (TH) Forschungsuniversität • gegründet 1825
Karlsruhe Institute of Technology

Atune-TA
TADL Connectors (1)

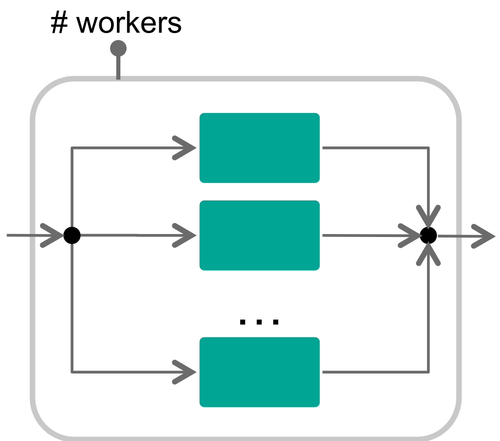
- *Sequential Composition*
  - General-purpose connector with sequential execution semantics



- *Tunable Alternative*
  - Describes exclusive choice
  - Auto-tuner tests alternatives during optimization process

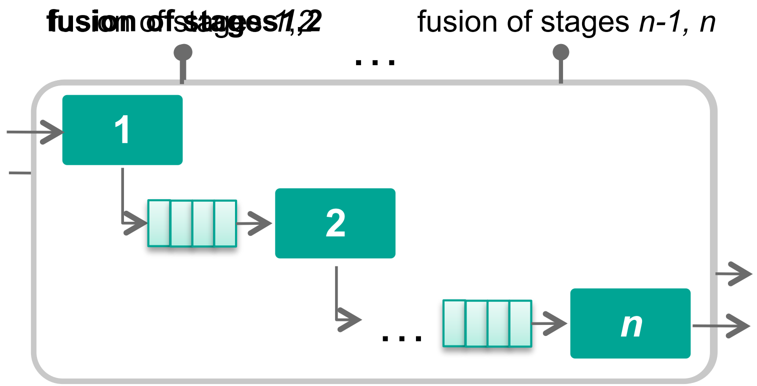


- *Tunable Fork/Join*
  - Introduces task parallelism

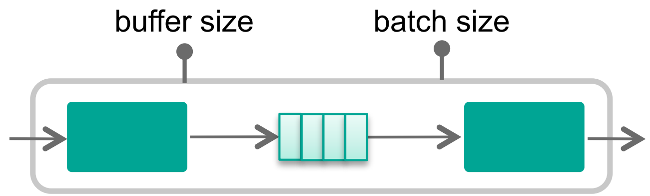


Atune-TA
TADL Connectors (2)

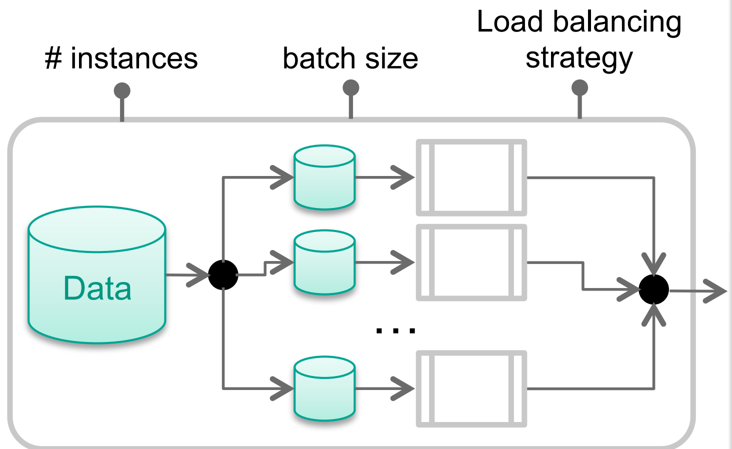
- *Tunable Pipeline*
  - Describes pipeline parallelism
  - Offers data stream semantics



- *Tunable Producer/Consumer*
  - Describes common synchronization pattern
  - Offers data stream semantics



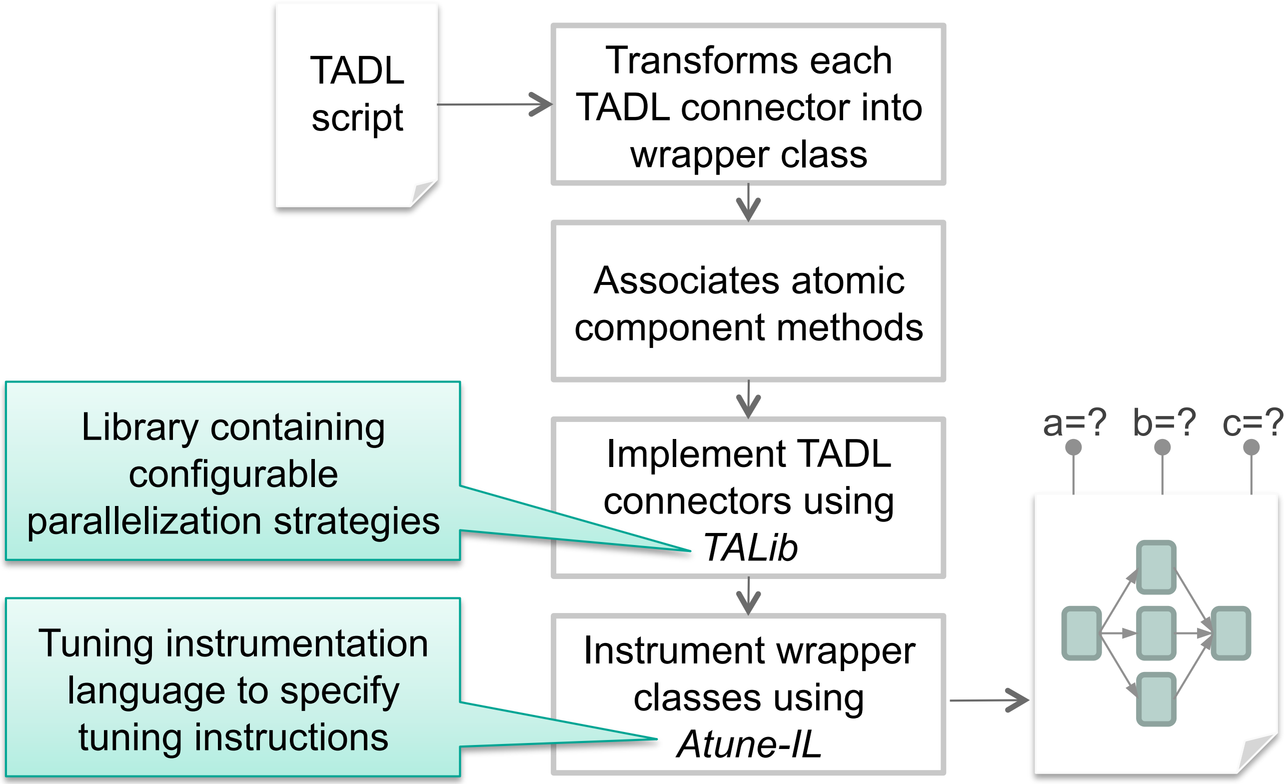
- *Tunable Replication*
  - Introduces data parallelism
  - Creates instances of atomic component



Atune-TA

Tunable Architecture Implementation: TADL Compiler

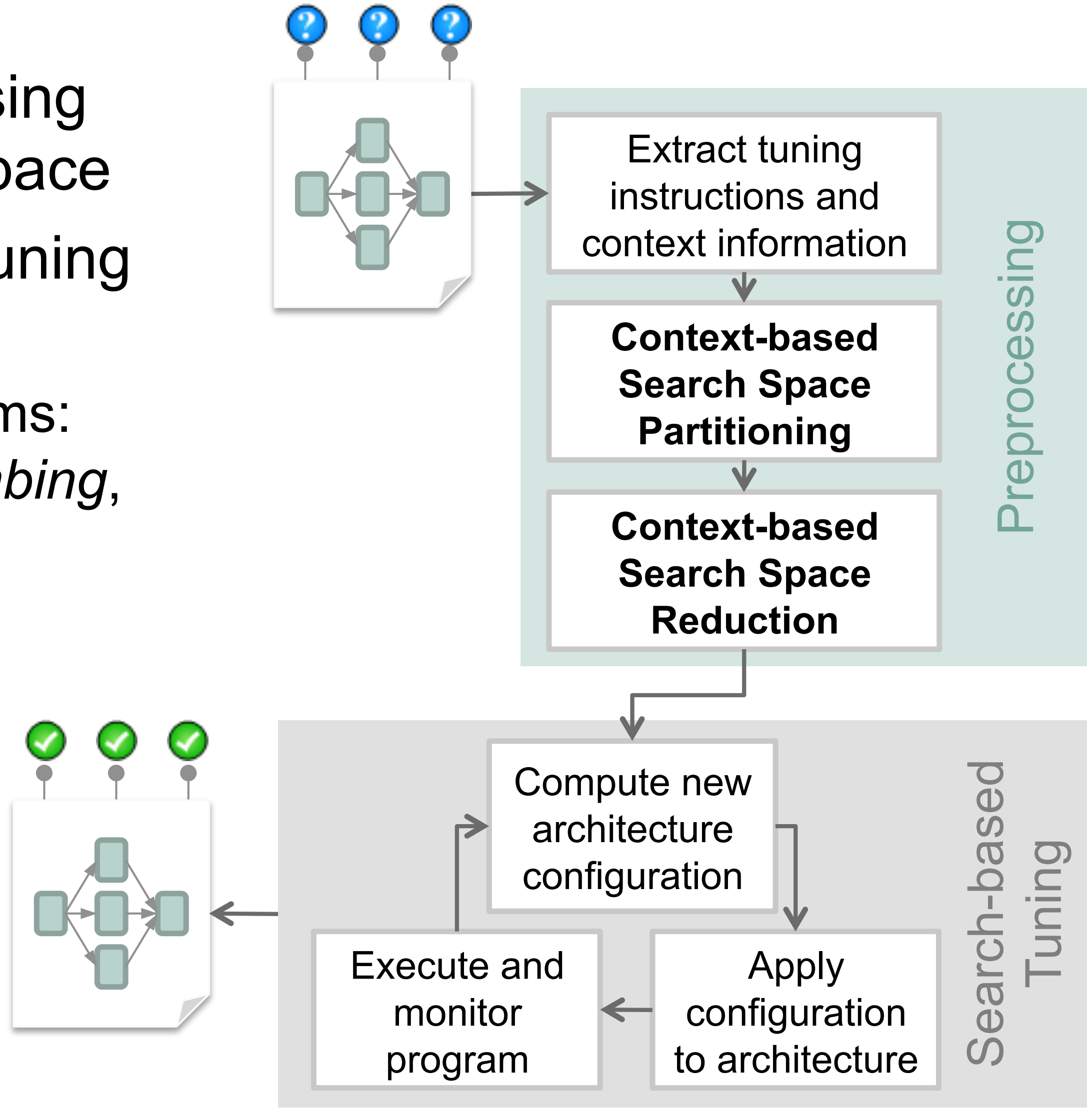
- *TADL compiler* transforms TADL script into instrumented, parallel executable code
- Result: portable intermediate representation of parallel program, ready for optimization on target platform



Atune-OPT

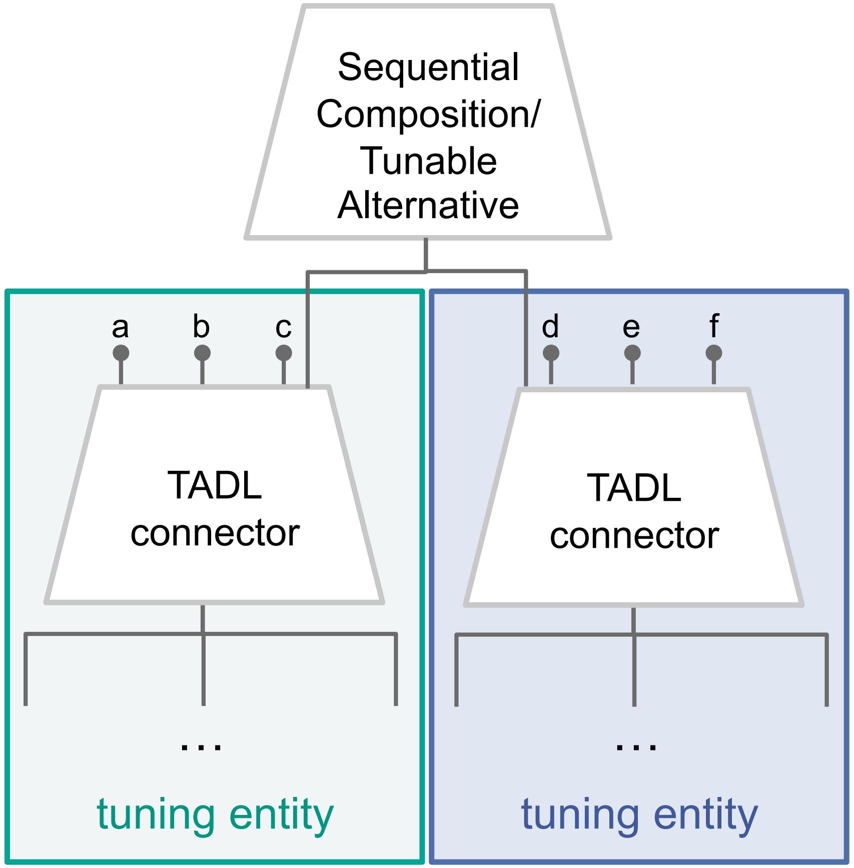
Overview and Process

- Context-based preprocessing steps to prepare search space
- Automatic search-based tuning of parallel architecture
  - Common search algorithms: *random sampling*, *hillclimbing*, *swarm optimization*



Atune-OPT
Context-based Search Space Partitioning

- Goal: Identification of program parts to tune independently → tuning entities
- Exploit semantics of
  - *Sequential Composition*
  - *Tunable Alternative*
- Sub trees can be tuned separately, as they never run concurrently
- Partitioning into tuning entities
- Separate optimization of the tuning entities → reduction of parameter configurations



$$S = V_a \times V_b \times V_c \times V_d \times V_e \times V_f$$

$$R = V_a \times V_b \times V_c \cup V_d \times V_e \times V_f$$

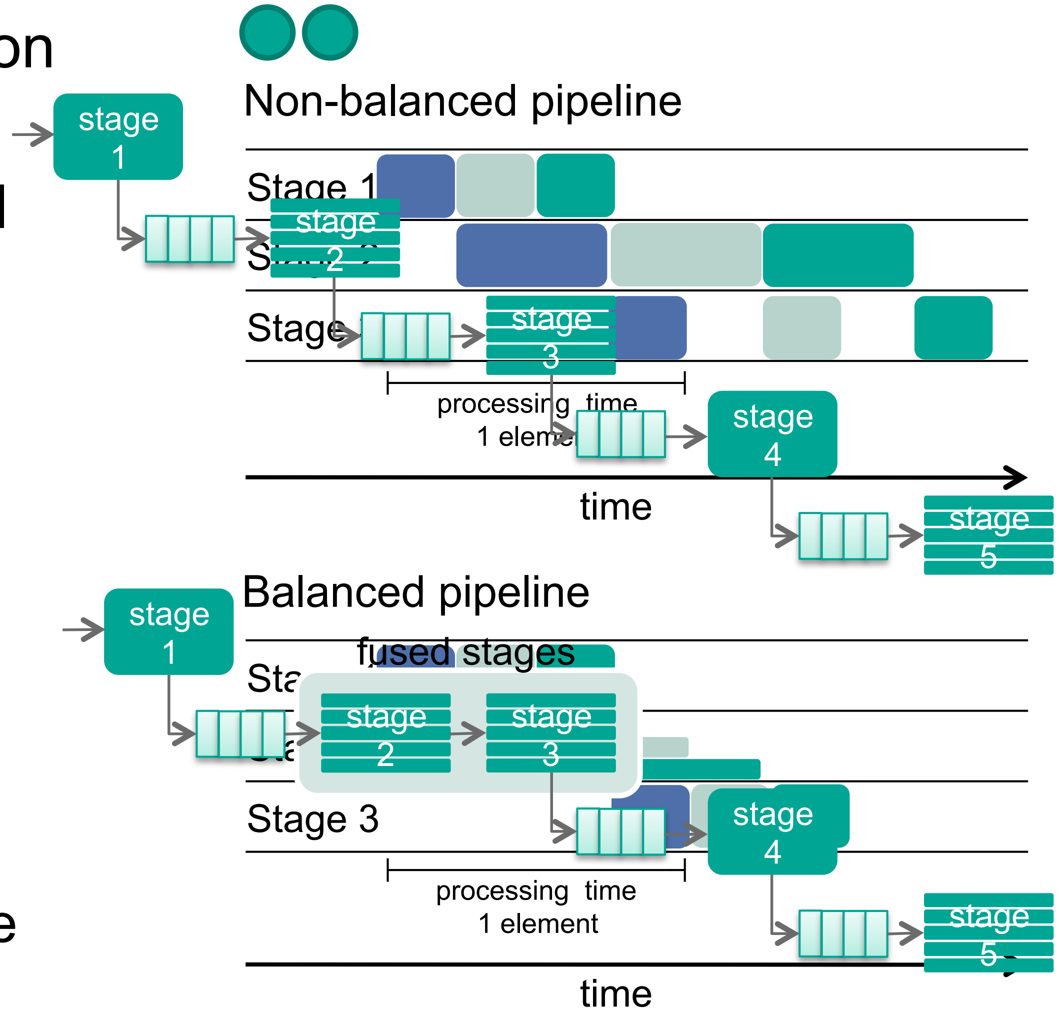
$$R \subset S$$

Atune-OPT
Context-based Search Space Reduction

- Goal: Search space reduction *using guided search*
- Exploit semantics of parallel TADL connectors

Example

- *Tunable Pipeline* with data-parallel stages
- Instead of “blind tuning” *we apply heuristics*:
  - Balancing the pipeline
  - Fuse groups of consecutive data-parallel stages



Evaluation
Case Studies

Application	Purpose	Size (LOC)	Exec. time sequential	Parallelism Types[1)]	Input data / benchmark
MetaboliteID (MID)	Bio-chemical data analysis	~ 100,000	85 s	T / D	mass spectrograms (1 GB)
GrGen.NET	Graph rewriting	~ 80,000	45 s	T / D	simulation of biological gene expression (~ 9 mio. nodes)
Desktop Search (DS)	Indexing of documents	~ 5,500	14 h 35 m	P / D	10,700 text files (max. 613 KB)
Video	Video processing	~ 1,000	19 s	P / D	video (180 frames, 800x600 px.)

[1)] **P**: pipeline parallelism, **T**: task parallelism, **D**: data parallelism

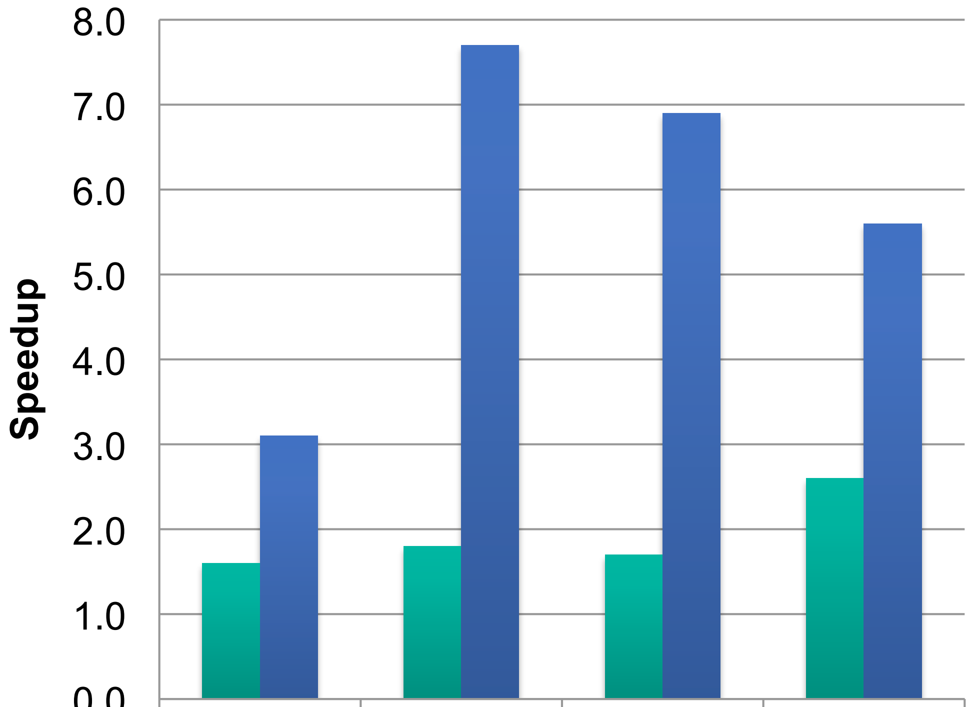
Evaluation
Experimental Results (1)

Karlsruhe Institute of Technology

- Performance evaluation: achieved speedup after optimizing parallel programs

Metrics

- *Worst* speedup
- *Best* speedup after tuning
- Tuning Performance Gain (TPG)



	MID	GrGen	DS	Video
Worst	1.6	1.8	1.7	2.6
Best	3.1	7.7	6.9	5.6
TPG	194%	428%	406%	215%

Experiments performed on 8-core-machine (2x Intel Xeon QC @ 1,86 GHz/Core).
Worst speedup results from testing most inappropriate parameter configuration.

Evaluation
Experimental Results (2)

- Evaluation: Reduction of implementaiton effort using *Atune-TA*
- Comparison of manual and Atune-TA-based implementation

Metrics

- LOC
- # explicit synchronization primitives
- # explicit tuning instrumentation statements

		MID	GrGen	DS	Video
LOC	Manual	290	120	465	300
	Atune-TA	3	3	3	3
	Reduction	**287** (99%)	**117** (98%)	**462** (99%)	**297** (99%)
Synchronization primitives[1)]	Manual	18	8	27	18
	Atune-TA	2	0	1	0
	Reduction	**16** (89%)	**8** (100%)	**26** (96%)	**18** (100%)
Tuning instrumentation statements	Manual	39	16	30	16
	Atune-TA	0	2	0	0
	Reduction	**39** (100%)	**14** (87%)	**30** (100%)	**16** (100%)

[1)] Includes all synchronization primitives, such as *lock, notify, wait, join,* etc.

Related Work

- *ATLAS/AEOS* (Whaley et al., 2000)
  - Auto-tuning system for algebraic operations and algorithms
  - Domain specific approach
  - No support for parallel programs
- *Active Harmony* (Tapus et al., 2002)
  - Search-based auto-tuning system for library optimization
  - Comprehensive analysis of search algorithms
  - Not applicable for parallel programs
- *MATE* (Morajko et al., 2007)
  - Model-based tuning system for distributed PVM programs
  - Provides good performance predictions
  - Limited to special program structures
- *Parallel Pattern Language* (Mattson et al., 2004)
  - Structured collection of parallel patterns
  - Provides guideline for parallel programming
  - Optimization is not considered

Future Work

- More tunable patterns!
- Language integration of patterns (XJava)
- Online tuning (instead of offline)
- Parameter prediction
  - Set good starting values for search, or elimiante search
    - Set replication depending on idle threads
    - Prefer tasks that have the most input waiting
    - Observe work stealing behavior for cutoff-value
  - First results: we achieve 90% of best configuration without search

Conclusion

- Multi-core systems force developers to exploit parallelism in programs
- Auto-tuning of parallel programs is indispensable to achieve good performance

- *Atune* provides automated approach to design, implement and optimize parallel tunable architectures
  - Combination of parallelization *and* optimization
  - High-level parallelization process of applications
  - Extension of search-based auto-tuning to handle entire architectures

- *Atune-TA*: Using tunable architectures results in reduction of implementation effort
- *Atune-OPT*: Novel tuning techniques provide efficient optimization and significant performance gain

THANK YOU!
QUESTIONS?

For details see:
Christoph A. Schaefer, Victor Pankratius, Walter F. Tichy:
Engineering Parallel Applications with Tunable Architectures.
In Proceedings of 32nd International Conference on Software Engineering (ICSE),
to appear May 2010

BACKUP SLIDES

Atune-IL: Tuning Instrumentation Language (1)

- Declaration of Tuning Blocks

```
#pragma atune STARTBLOCK myBlock type PIPELINE
    <source code statements>
    <other Atune-IL statements>
    …
#pragma atune ENDBLOCK
```

  - Define scopes of tuning parameters
  - Tuning Blocks support
    - Nesting (lexically or logically) to represent application structure
    - Types to specify context

- Declaration of Tuning Parameters

```
#pragma atune SETVAR myParameter type int
        values 10-100 step 10, weight 3, inside myBlock
```

Atune-IL: Tuning Instrumentation Language (2)

- Further constructs to
  - declare measuring points (incl. metric)
  - declare permutation regions (to re-order statements in host language)
- Atune-IL's design goals
  - Separation of program code and tuning instructions
  - Compact representation of performance-relevant variants of parallel architectures
  - Syntax suitable for automatic generation

Evaluation
Assumptions

- Estimation of manual implementation effort to implement functionality of TADL connectors

TADL connector	LOC	# Syncs [1)]
Tunable Alternative	15	0
Tunable Fork/Join	170	10
Tunable Pipeline	180	10
Tunable Producer/Consumer	150	9
Tunable Replication	120	8

[1)] Total number of synchronization-related statements in source code, such as *lock, notify, wait, join,* etc.