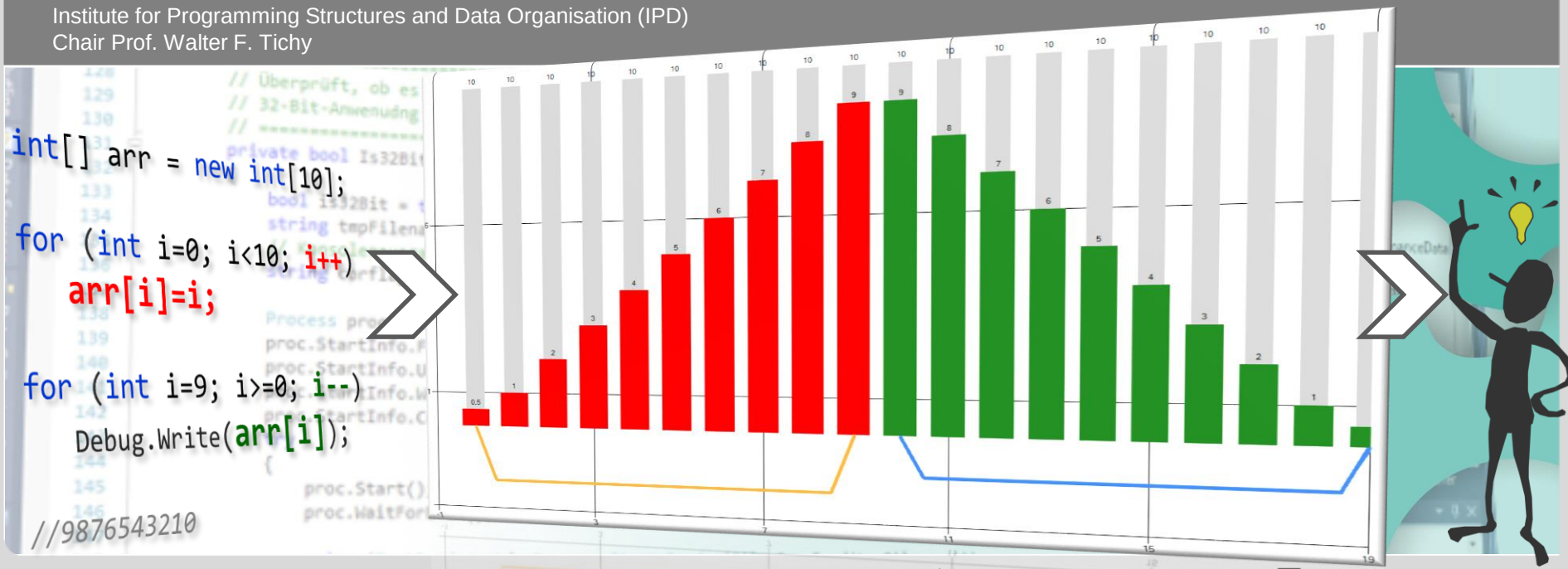
Karlsruhe Institute of Technology

SIEMENS
Corporate Technology

Locating Parallelization Potential in Object-Oriented Data Structures

Korbinian Molitorisz

Institute for Programming Structures and Data Organisation (IPD)
Chair Prof. Walter F. Tichy



Forschungszentrum Karlsruhe
in der Helmholtz-Gemeinschaft

Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Location and Current Research

- Karlsruhe Institute of Technology (KIT)
  - Faculty of Computer Science
  - Chair Prof. Dr. Walter F. Tichy
  - Research Group AParT

- Research Group AParT
  - Parallelizing existing „legacy“ software
  - Support engineering parallel software
  - Active collaboration with Siemens Corporate Technology
  - 4 PhDs, several Bachelor-/Master-students

What makes parallelization so hard?

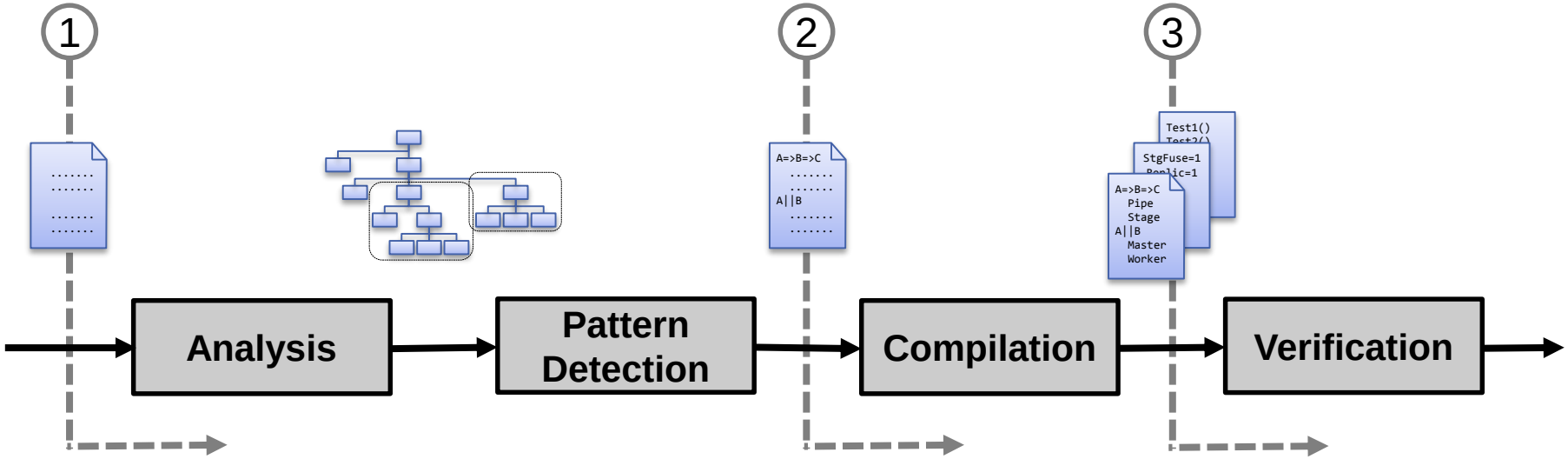
- Parallelization is time-consuming and error-prone
- Case Studies
  - Implementating a video processing pipeline: Several weeks [OS+10]
  - Implementing a desktop search engine: The third parallelization approach achieved acceptable performance [MT10]
- „Meanwhile, multicore processors have become *mainstream, but not the knowledge how to program them*“ [VM11]

⇨ Pattern-based parallelization process für legacy software is urgently needed

[OS+10] – Frank Otto, Christoph Schafer et al. *A language-based tuning mechanism for task and pipeline parallelism*, Euro-Par 2010
[MT10] – David Meder, Walter F. Tichy. Parallelizing an Index Generator for Desktop Search, ISCA 2010
[SM+13] – Jochen Schimmel, Korbinian Molitorisz, Ali Jannesari, Walter F. Tichy. *Automatic Generation of Parallel Unit Tests*, AST 2013
[VM11] – Hans Vandierendonck, Tom Mens. *Averting the Next Software Crisis*, 2011

Pattern-based Parallelization Process

- Identifies source patterns and runtime-relevant *tuning parameters*
- Transforms them to parallel patterns like *Pipeline* or *Master/Worker*



[MC+14] – K. Molitorisz, L. M. Carril. *Pattern-based Parallelization*, parallel 2014
[SM+13a] – J. Schimmel, K. Molitorisz, A. Jannesari, W. F. Tichy. *Automatic Generation of Parallel Unit Tests*, AST 2013
[SM+13b] – J. Schimmel, K. Molitorisz, W. F. Tichy. *An Evaluation of Data Race Detectors Using Bug Repositories*, AST 2013
[Mol13] – K. Molitorisz. *Pattern-based Refactoring Process of Sequential source Code*, CSMR 2013
[MS+12] – K. Molitorisz, J. Schimmel, F. Otto. *Automatic Parallelization using AutoFutures*, MSEPT 2012

Motivation – Let‘s try something different!

- Today, detecting parallel potential always looks for hotspots (high runtime share, high number of executions)

- Observations
  - Object-oriented paradigm heavily used
  - Data structures (DS) comprise containers plus algorithms

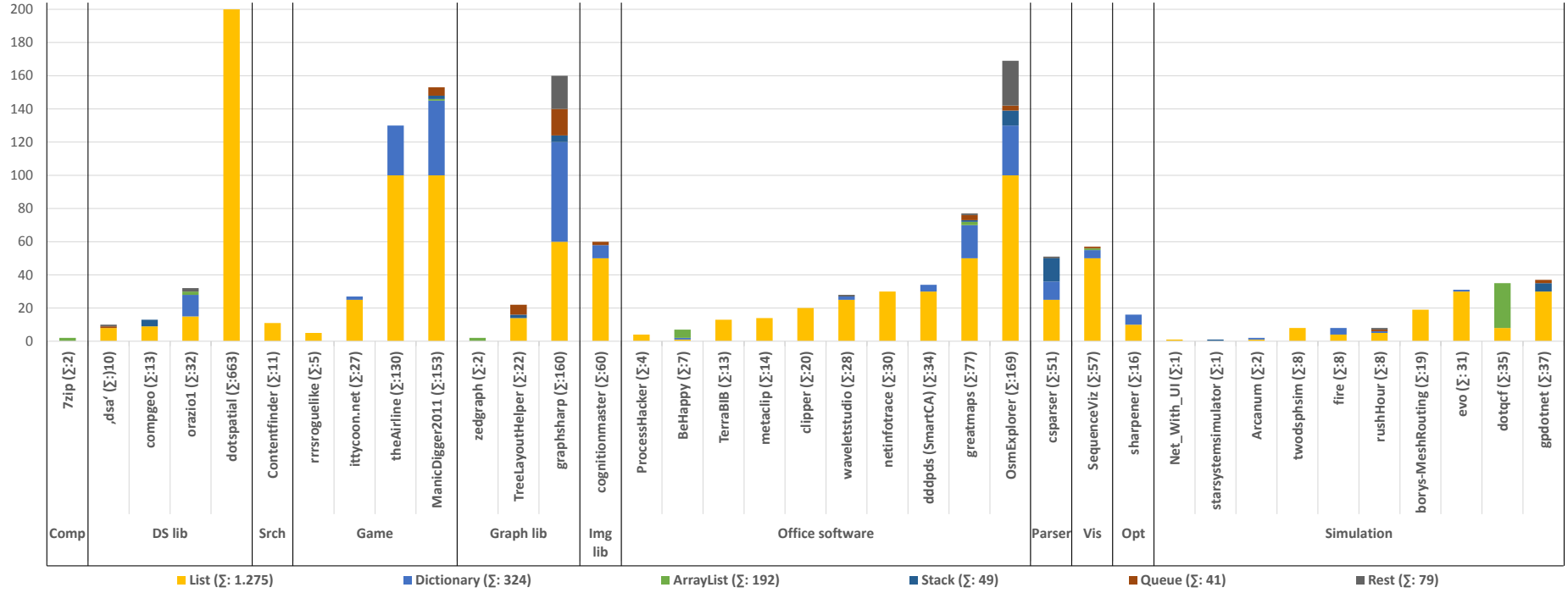
➡ Can we derive parallel potential from monitoring accesses to object-oriented data structures?

Empirical Study

- Research Questions
  - Q_1 – Frequency: What object-oriented data structures are used in reality? (Not those that can be found in text books!)
  - Q_2 – Access patterns: Can we find recurring regularities in the access profiles of these object-oriented data structures?
  - Q_3 – Parallelizabilty: Do they carry parallel potential?

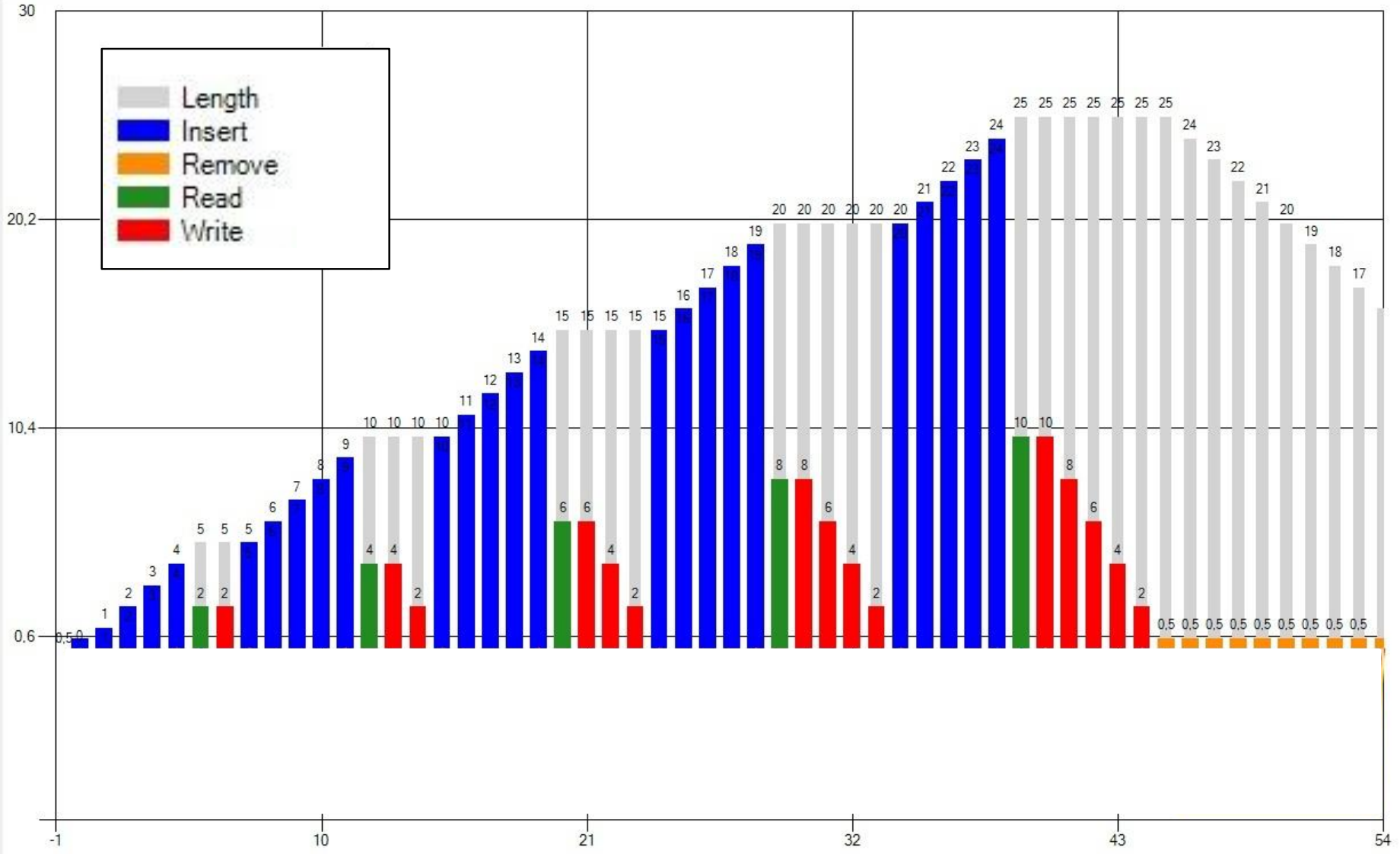
- Benchmark
  - 37 open source projects from 11 application domains
  - 2 – 718 DS instances, in total 1,960
  - 300 – 460,000 LOC, in total >936,000 LOC

Q_1: Data structure frequency?

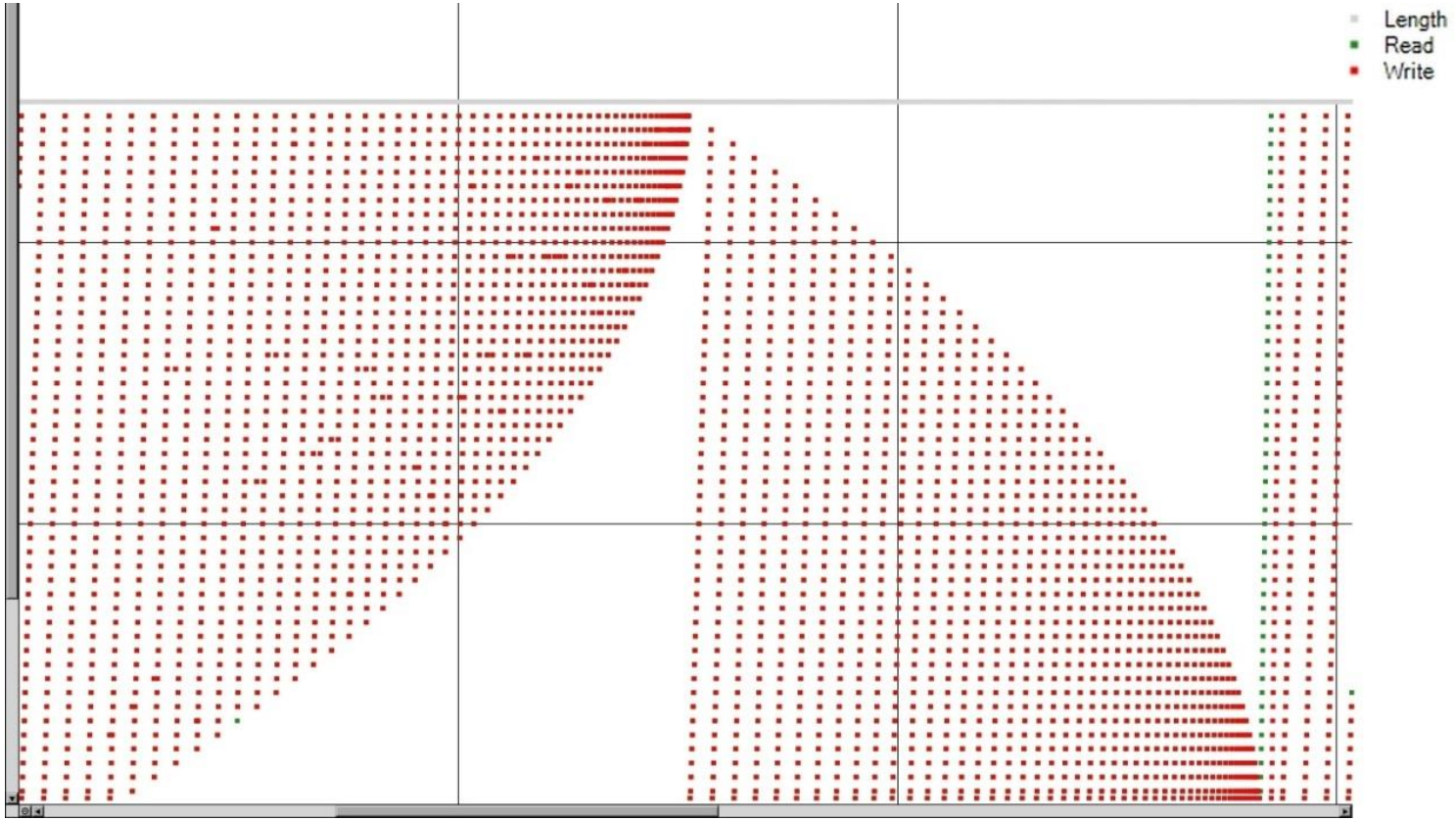


- Assumption: Arrays are heavily used static DS
- Research focus: dynamic DS
- Result: `List` and `Dictionary` together make up >81%

Q_2: Access Patterns?

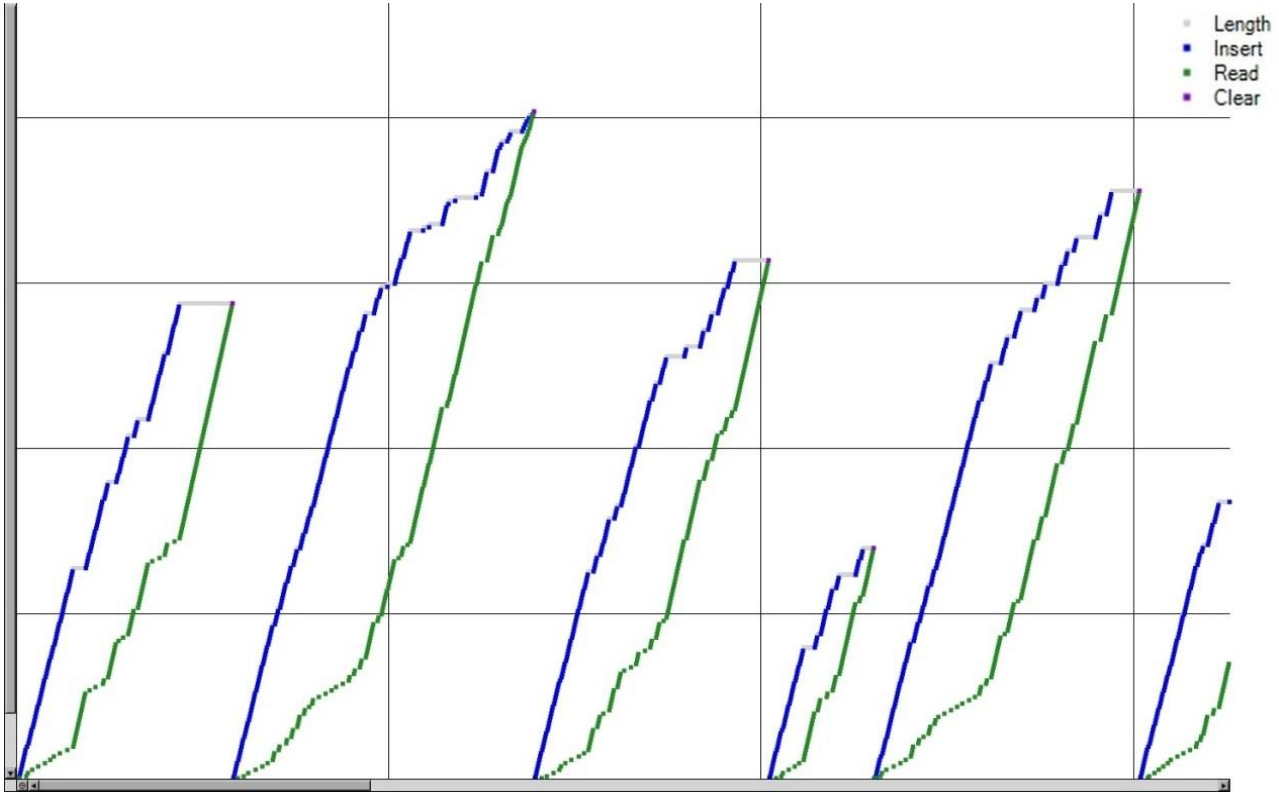


Q_2: Access Patterns? (Linpack benchmark)



- Subsequent write operations (red dots)
  - From front → end (offset one at the front)
  - From front → end (offset one at the end)

Q_2: Access Patterns? (Mesh Routing)



- Insert elements at the end of DS (blue lines)
- Overlapping read operations from front → end (green lines)
- When read reaches the last element: `clear()`

Q_2: Access Patterns!

Number of DS… Program	…with detectable patterns	…with already identified patterns	…without patterns
astrogrep	0	2	1
borys-MeshRouting	4	3	7
clipper	3	9	1
compgeo	2	0	0
contentfinder	0	2	0
csparser	2	5	3
‚dsa‘	0	5	0
dotqcf	4	2	0
fire	1	1	1
ManicDigger2011	1	6	7
MidiSheetMusic	4	14	0
Net_With_UI	3	11	3
netinfotrace	4	13	4
rrrsroguelike	1	1	0
TerraBIB	2	1	1
TreeLayoutHelper	0	6	2
∑	**31**	**81**	**30**

Q_3: Parallel Potential?

Parallel Potential	Description / Threshold values	Exploit Potential Advice
`Long-Insert` (49x in 21 programs)	>100 insert patterns, >100 insert operations, >30% of all accesses	Parallelize the insert operation
`Frequent-Long-Read` (10x in 8 programs)	>10 consecutive reads, >50% read operations read >50% of the data structure	Check access origin. It might be a search operation for a specific element: Transform this operation to a parallel search operation
`Frequent-Search` (3x in 2 programs)	>1,000 search operations for a specific element, >2% of all access events are read operations	- Employ a parallel data structure for searches - Parallelize the search operation (e.g. data parallel search)
`Implement-Queue` (3x in 3 programs)	DS is used as queue, >100 read/write operations to one or both ends (FIFO, FILO), >60% of all accesses	Employ a parallel queue as data container
`Sort-After-Insert` (1x in 1 program)	DS is sorted after a Long-Insert	Parallelize both insert and search phases

Evaluation

Source Code			Dynamic Analysis			Access Patterns & Potential			Recommendation
Name	LOC	Domain	Runtime	Profiling	Slowdown	Data Structures	Potential	Search Space	Total Speedup
Algorithmia	2,800	Library	0.50	2.40	4.80	16	2 of 4	75.00%	1.83
Astrogrep	4,800	File Search	4.80	5.80	1.21	21	1 of 2	90.48%	2.90
Contentfinder	290	File Search	1.80	5.20	2.89	11	2 of 2	81.82%	1.56
CPU Benchmarks	400	Benchmark	0.01	0.55	55.00	7	4 of 5	28.57%	1.20
Gpdotnet	7,000	Simulation	0.36	78.00	216.67	37	2 of 5	86.49%	2.93
Mandelbrot	150	Solver	0.11	1.20	10.91	7	4 of 4	42.86%	3.00
WordWheelSolver	110	Solver	0.04	150	38.46	5	1 of 2	60.00%	1.50
Total	15,550				47.13	104	16 of 24	76.92%	2.13

- Assembled new benchmark, >15,000 LOC
- Test system: 8-core AMD FX 8120h, 3,1 Ghz, 8 GB RAM
- Rather moderate speedups, but for free

Conclusion

- Software engineers need a tool support for parallelization
- „Traditional“ hotspot analyses well-established
- Exploratory empirical study in >936 KLOC revealed `List` and `Dictionary` make up >81% of all dynamic DS
- Runtime profile of dynamic data structures in commodity software contains recurring access patterns
- They can be identified automatically and serve for parallelization with rather moderate speedups
- After IPDPS available on our website http://www.apart-project.de

Thank you for your attention!

Any questions?

molitorisz@kit.edu