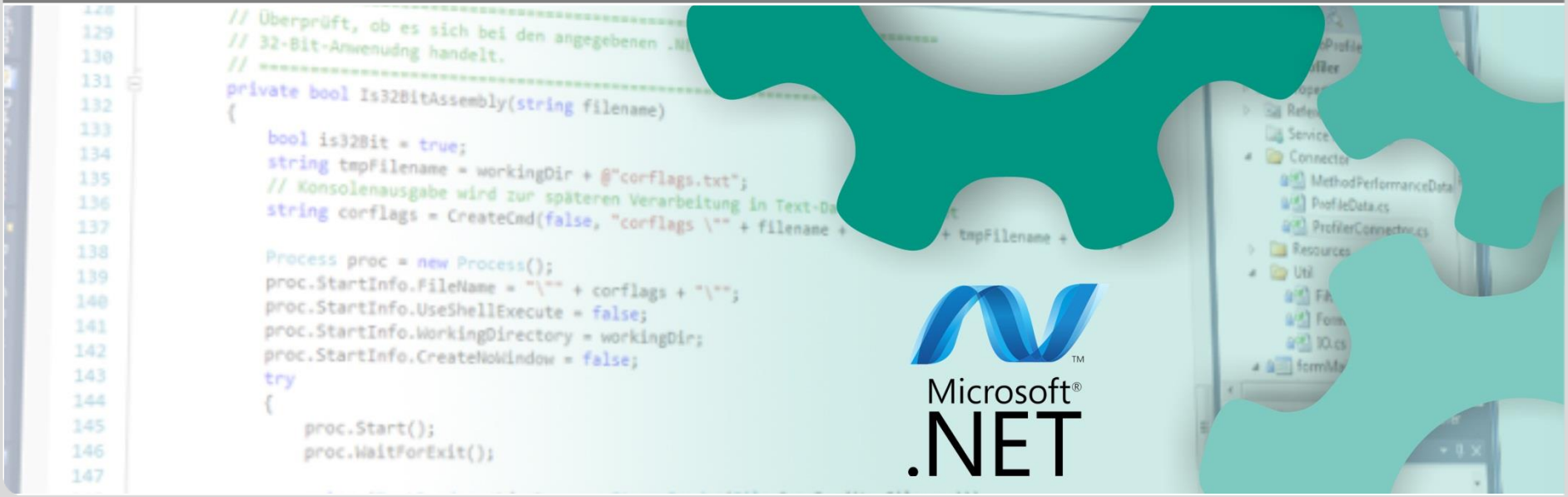
Karlsruhe Institute of Technology

SIEMENS
Corporate Technology

The *Fellow Research Group*: Pattern-based Parallelization (AParT)
A research cooperation between the KIT and Siemens Corporate Technology

Korbinian Molitorisz

IPD Prof. Walter F. Tichy – Chair for Programming Systems



Location and Current Research

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

- Fellow Research Group AParT
  - Parallelizing existing „legacy“ software
  - Support engineering parallel software
  - Active collaboration with Siemens Corporate Technology

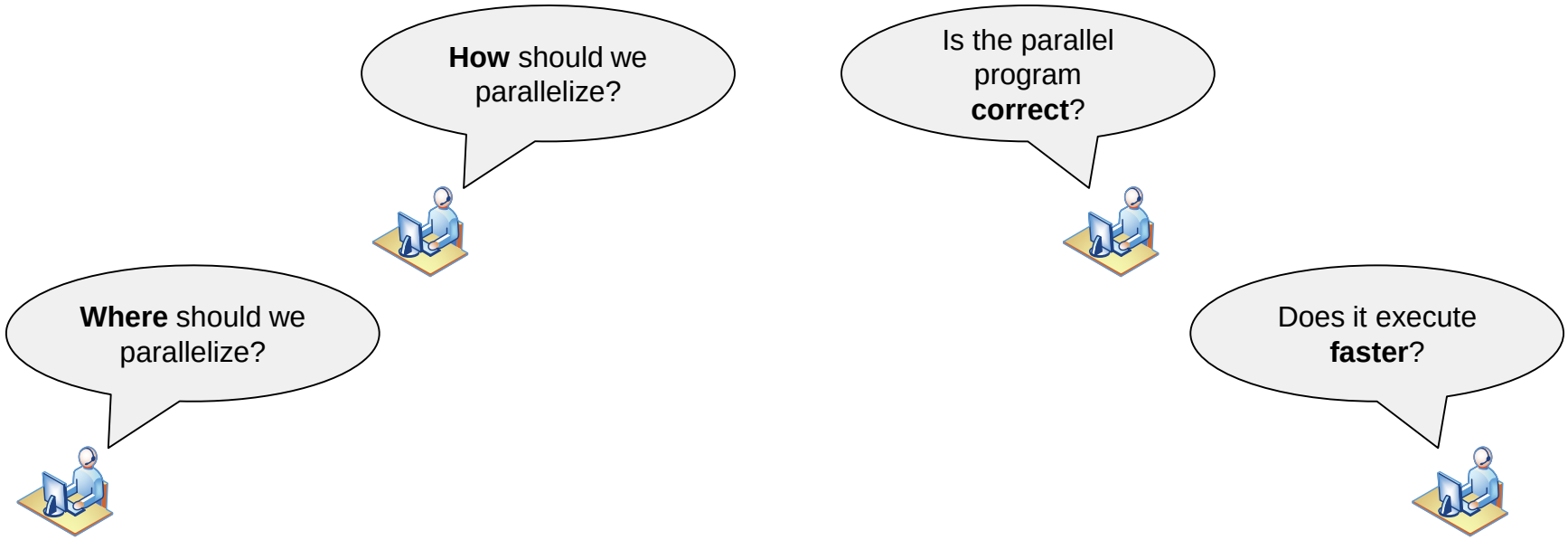
About myself

- Employee at the chair of Prof. Walter F. Tichy
  - 2003 – 2014: Activity in the ESCde
    - Support institution in cooperation with Microsoft
    - 2003 – 2006: Support engineer, Head of QA
    - 2006 – 2014: General Manager
  - Since 2009: Ph.D. in *Multicore*
    - Design pattern-based parallelization of existing software systems
    - Member in the *Fellow Research Group* (FRG) http://frg.ipd.kit.edu
  - Since 2014: Head of FRG
    - Research group with 4 members and 8 students (BA/MA/grad. assistant)
    - Integrated support concept for students‘ theses
- Contact information
  - Building 50.34, room 370,
    Tel. 0721 / 608 - 4 7155

Korbinian Molitorisz
Molitorisz@kit.edu

Why do we deal with parallelization at all?

- „*The free lunch is over*“ – Multicore processors need parallel software
- But: Engineering parallel software is hard



What makes engineering parallel software hard?

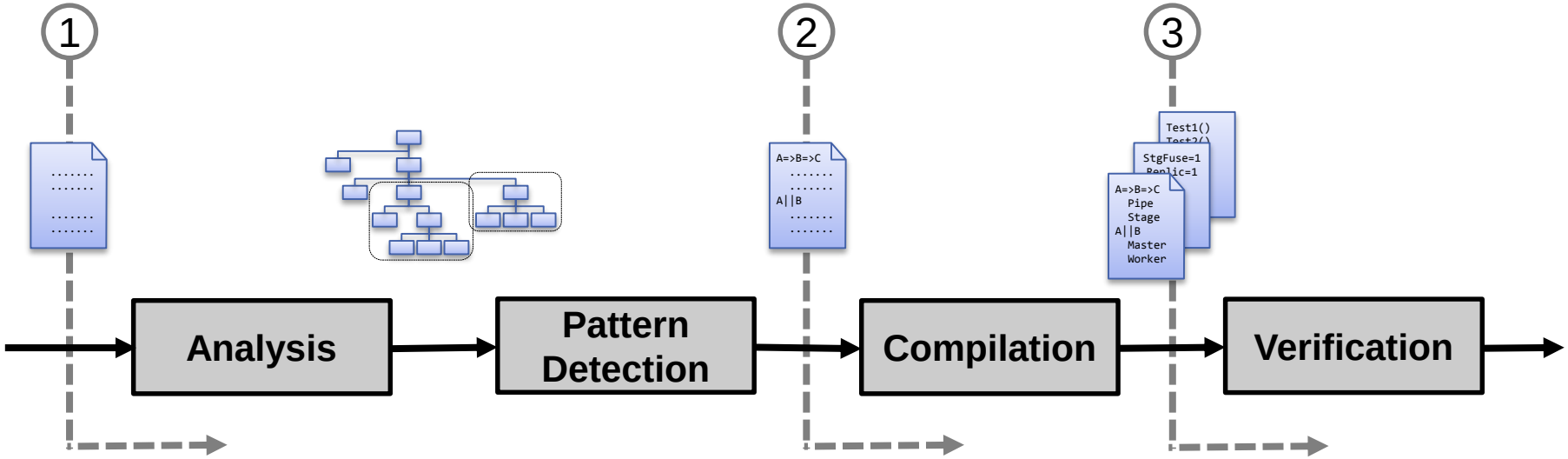
- Parallelization is time-consuming, tedious, and error-prone
  - Several weeks of work for implementing a pipeline (parallelization of a video processing software [OS+10])
  - Only the third parallelization strategy yielded a speedup (desktop search [MT10])
  - „*Multicore processors are ubiquitous by now. The skills to program them is not* “ [VM11]

⇨ Pattern-based parallelization process for legacy software is urgently needed [Mol13]

[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[VM11] – Hans Vandierendonck, Tom Mens. *Averting the Next Software Crisis*, 2011
[Mol13] – K. Molitorisz. *Pattern-based Refactoring Process of Sequential source Code*, CSMR 2013

Pattern-based parallelization process

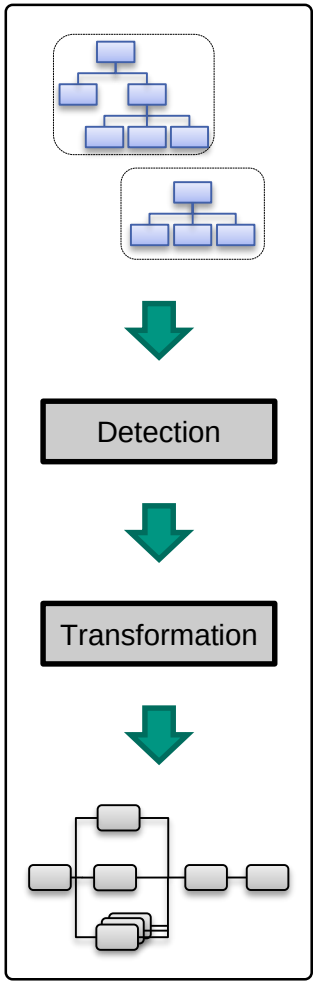
- Flexible parallelization
  - 1: Fully-automatic detection and transformation
  - 2: Semi-automatic transformation using architecture description
  - 3: Performance optimization of existing parallel architectures



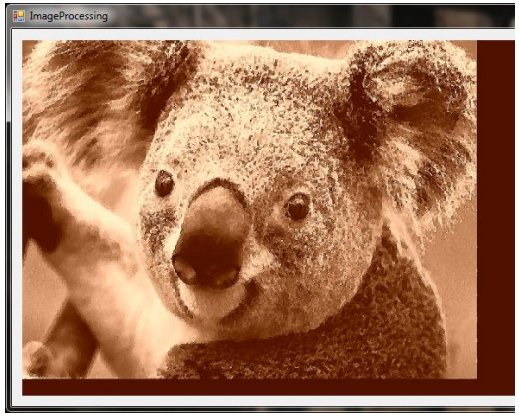
[MC+14] – K. Molitorisz, L. M. Carril. *Pattern-based Parallelization*, parallel 2014
[MK+14] – K. Molitorisz, T. Karcher, A. Bieleš, W. F. Tichy. *Locating Parallelization Potential in Object-Oriented Data Structures*, IPDPS 2014
[SM+13a] – J. Schimmel, K. Molitorisz, A. Jannesari, W. F. Tichy. *Automatic 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*, FedCSIS 2013
[MS+12] – K. Molitorisz, J. Schimmel, F. Otto. *Automatic Parallelization using AutoFutures*, MSEPT 2012

Components of pattern-based parallelization

- Pattern catalog
  - Source patterns
  - Target patterns
- Procedural method
  - Detection (Pattern instances, runtime parameters)
  - Transformation
- Specification of detected patterns
- Correctness validation
  - Creation of unit tests
  - Dynamic data race detector
- Runtime optimization
  - Auto tuning using explicit runtime parameters



Detection of Source Patterns




```
01  AviStream Process(AviStream aviIn)
02  {
03   AviStream aviOut = new AviStream();
04   foreach(Image i in aviIn.Images)
05   {
06    Image crop  = cropFilter.Apply(i);
07    Image histo = histogramFilter.Apply(i);
08    Image oil   = oilFilter.Apply(i);
09    Image res   = ConvTo32bpp.Apply(crop, histo, oil);
10    aviOut.Images.Add(res);
11   }
12   return aviOut;
13  }
```

- Code region with high runtime share
- Nesting of statements
  - Expression statement – statement block – `return`-statement
- Stepwise application of several image filters
- Program structures
  - Program loop: Iterative stream of statements
  - Statement sequence: Consecutive statements

Detection of Source Patterns

```
01  AviStream Process(AviStream aviIn)
02  {
03   AviStream aviOut = new AviStream();
04   foreach(Image i in aviIn.Images)                    1.440x
05   {
06    Image crop  = cropFilter.Apply(i);                    8%
07    Image histo = histogramFilter.Apply(i);               9%
08    Image oil   = oilFilter.Apply(i);                    68%
09    Image res   = ConvTo32bpp.Apply(crop, histo, oil);   12%
10    aviOut.Images.Add(res);                               3%
11   }
12   return aviOut;
13  }
```

- Program structure: Program iteration
  - One thread per iteration or iteration sequence
  - One thread per statement
- Program structure: Statement sequence
  - One thread per statement
  - One thread per stament sequence

Transformation to Target Patterns

```
01  AviStream Process(AviStream aviIn)
02  {
03   AviStream aviOut = new AviStream();
04   foreach(Image i in aviIn.Images)                    1.440x
05   {
06    Image crop  = cropFilter.Apply(i);                    8%
07    Image histo = histogramFilter.Apply(i);               9%
08    Image oil   = oilFilter.Apply(i);                    68%
09    Image res   = ConvTo32bpp.Apply(crop, histo, oil);   12%
10    aviOut.Images.Add(res);                               3%
11   }
12   return aviOut;
13  }
```

- Target patterns
  - Program iteration → Pipeline
    - Statements: Stages
    - Video stream: Data
  - Statement sequence → Master-/Worker
    - Control thread: Master
    - Statements: Worker
- Speedup (on an 8-core machine)
  - Manual: 6.47. Effort: 8 hours
  - Automatic: 7.18. Effort: 10 minutes

Transformation to Target Patterns

- Code annotations
  - Defined operands and operators for architecture description and interconnecting architecture components
- Compiler with runtime library
  - Input: Architecture description and architecture components
  - Output: Instances from runtime library and tuning file

```
01  AviStream Process(AviStream aviIn)
02  {
03   AviStream aviOut = new AviStream();
04   #region TADL: (A || B || C+) => D => E
05   foreach(Image i in aviIn.Images)
06   {
07    #region A: Image c = cropFilter.Apply(i);       #endregion
08    #region B: Image h = histogramFilter.Apply(i); #endregion
09    #region C: Image o = oilFilter.Apply(i);        #endregion
10    #region D: Image r = Conv32bpp.Apply(c, h, o); #endregion
11    #region E: aviOut.Images.Add(r);                #endregion
12   }
13   #endregion
14   return aviOut;
15  }
```

```
01  AviStream Process(AviStream aviIn)
02  {
03   Item p1 = new Item (cropFilter.Apply());
04   Item p2 = new Item (histogramFilter.Apply());
04   Item p3 = new Item (oilFilter.Apply());
05   Item p4 = new Item (ConvTo32bpp.Apply());
06   Item p5 = new Item (aviOut.Images.Add());
07   MasterWorker mw = new MasterWorker (p1, p2, p3);
08   mw.Item(p3).replicable = true;
10   Pipeline p = new Pipeline (mw, p4, p5);
11   p.Input = aviIn.Images;
12   p.Run();
13   return p.Output;
14  }
```

Evaluation: Benchmark

- Assessment of the pattern-based parallelization process for *pipelines* and *master/worker* using 26,500 LOC

Project	#LOC	#Methods	Program structures	
			Statement sequence	Program iterations
MergeSort	94	6	14	5
RayTracer	522	37	58	5
DesktopSearch	315	22	40	7
CompGeo	1,058	73	149	20
ImageProcessing	110	12	14	2
PowerCollections	24,481	1,252	3,354	287

Evaluation:
Search Space Reduction and Recall

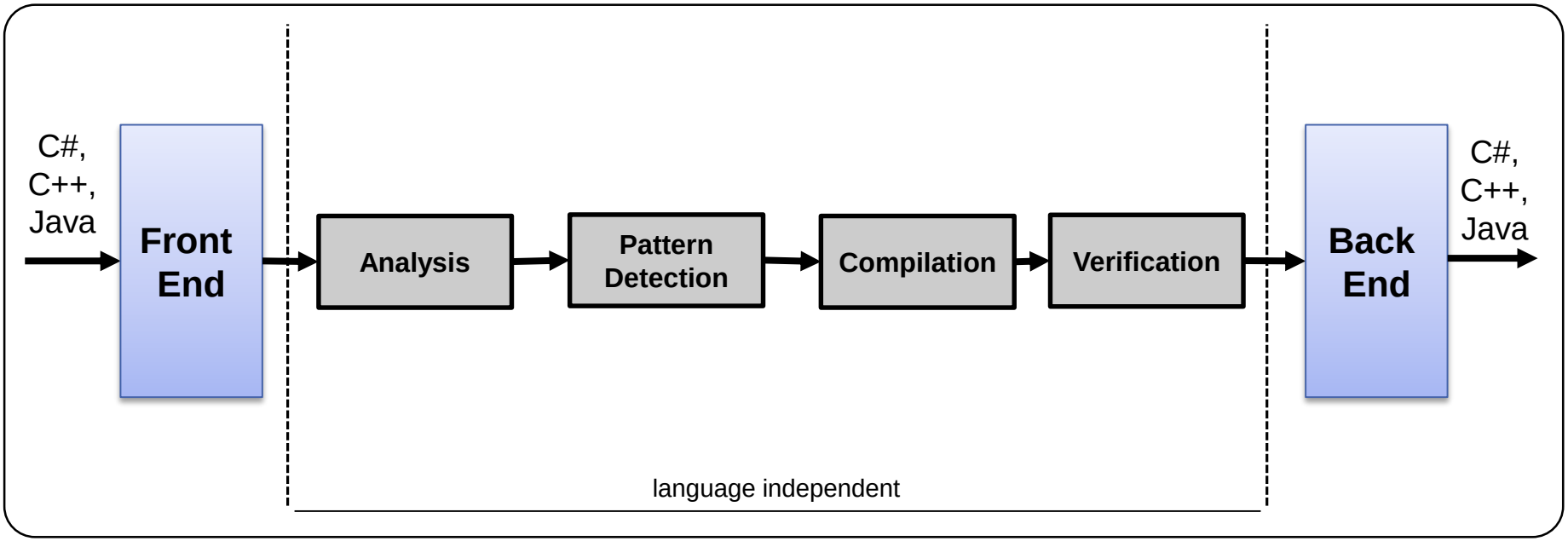
Project	#Program structures	Target patterns	Search space reduction
MergeSort	19	1	95 %
RayTracer	63	6	90 %
DesktopSearch	47	2	96 %
CompGeo	169	1	98 %
ImageProcessing	16	1	94 %
PowerCollections	3,641	172	97 %
			Ø = 95%

Evaluation: Precision and Speedup

- Test system: AMD FX 8120h, 8 processor cores, 3.1 GHz each

Project	Target pattern	Correctness		Precision	Speedup		Speedup
		C	¬C		S	¬S	
MergeSort	1	1	0	100 %	1	0	4.60
RayTracer	6	6	0	17 %	1	5	3.13
DesktopSearch	2	2	0	50 %	1	1	1.67
CompGeo	1	1	0	100 %	1	0	1.20
ImageProcessing	1	1	0	100 %	1	0	7.18
PowerCollections	15*	13	2	27 %	6	9	--
	∑ = 26	∑ = 24	∑ = 2	Ø = 66%	∑ = 11	∑ = 15	

Outlook



- Parallelization process
  - Generalization from C# to general object-oriented source code
  - Operate on sequential and parallel source code
  - Code coverage analysis for data race detection
  - Model-based data race detection

Related Work

		Assessment of parallel potential	Generation of parallel suggestion	Automatic tranformation	Focus on parallel patterns	Determination of data dependencies	Optimization of parallel program on target platform	Verification of parallel correctness	Adjustable pattern detection	Support of whole parallelization process	Supported platforms
Scientific publications	[GJ+11] - S. Garcia	+	-	-	-	dyn	-	-	-	-	C/C++
	[HS+09] - C. Hammacher	+	-	-	-	dyn	-	-	-	-	Java
	[K88] - M. Kumar	+	-	-	-	dyn	-	-	-	-	Fortran
	[KK+10] - M. Kim	-	+	-	MW, PL	dyn	-	-	-	-	OpenMP
	[MC+07] - T. Moseley	+	+	-	-	-	-	-	-	-	C++
	[MF+10] - J. Mak	+	+	+	FJ	dyn	-	-	-	-	Cilk
	[RV+10] - S. Rul	-	+	+	MW, PL	dyn	-	-	-	-	C++
	[TC+07] - W. Thies	-	-	-	PL	dyn	-	-	-	-	C
	[TF+10] - G. Tournavitis	+	+	+	PL	dyn	-	-	-	-	C/C++
	[TW+09] - G. Tournavitis	-	+	+	MW, PL	dyn	+	-	-	-	C/C++
	[W11] - A. Wilhelm	+	+	-	FJ, MW	stat	-	-	-	-	C/C++/Java
Works of the FRG	[MS+12] - K. Molitorisz	+	+	+	Futures	stat	-	-	+	+	Java
	[SM+13] - J. Schimmel	-	+	+	MW	dyn	-	+	+	+	.NET
	[Mol13] - K. Molitorisz	+	+	+	MW, PL, LP, RPL	stat/dyn	+	+	+	+	.NET
	[MK+14] – K. Molitorsz	-	+	-	DS	stat/dyn	-	-	+	+	.NET
Commercial tools	Intel Parallel Studio	+	-	-	FJ, MW	dyn	-	-	-	+	C/C++
	Critical Blue Prism	+	-	-	FJ, MW	dyn	-	+	-	+	Windows, Linux

Thank you!

molitorisz@kit.edu
http://frg.ipd.kit.edu