Tool-Supported Mini-App Extraction to Facilitate Program Analysis and Parallelization

TECHNISCHE UNIVERSITÄT DARMSTADT

International Conference on Parallel Processing (ICPP '21)

Aug. 10th 2021, Virtual Conference

SF 4.0
Software-Factory 4.0

Jan-Patrick Lehr @jplehr
jan-patrick.lehr@tu-darmstadt.de

Christian Bischof, Florian Dewald,
Heiko Mantel, Mohammad Norouzi, Felix Wolf

github.com/tudasc

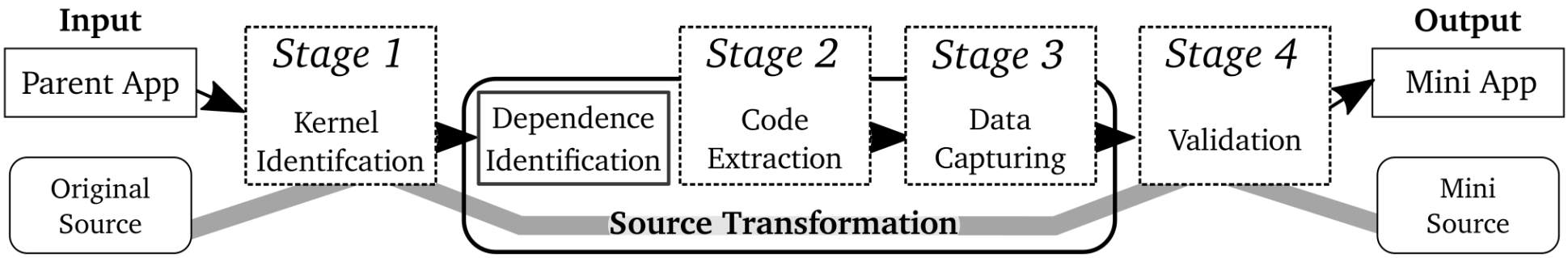
github.com/discopop-project

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- The vastness of today's software challenges manual and tool-supported analysis, for example, for correctness or parallelization
- Mini-apps are an intriguing vehicle to reduce complexity while maintaining key properties of the original application
- Creation of mini-apps challenging and time-consuming
  - Requires expertise in domain and computer science

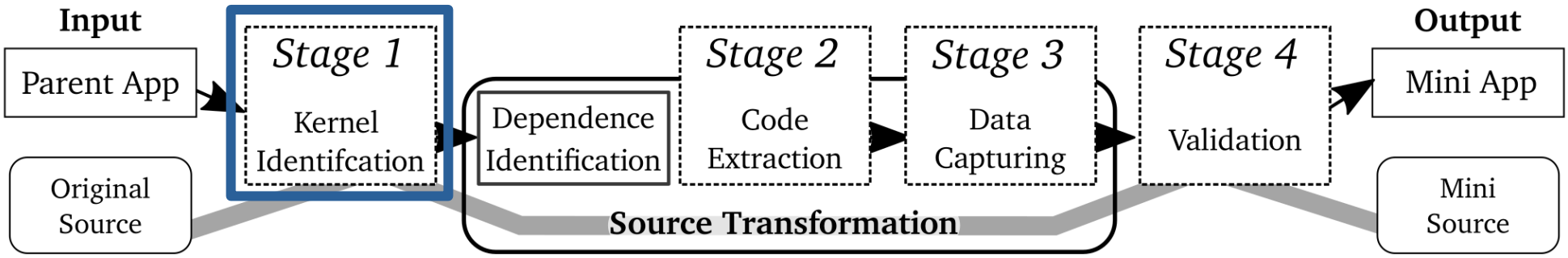
Approach Overview



Tool-support

1) **PIRA** [1] for automatic kernel detection
2) Novel **Clang**-based source-to-source translator
3) Type-safe checkpoint restart using **TyCart** [2] / **TypeART** [3]
4) Hierarchical clustering of execution metrics (**Caliper** [4] / **PAPI** [5], **PIN** [6])

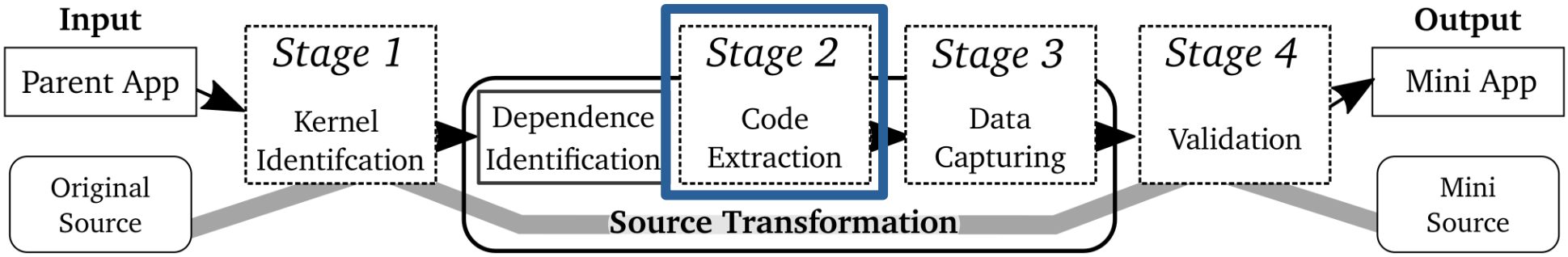
Approach Overview



Tool-support

1) **PIRA** [1] for automatic kernel detection
2) Novel **Clang**-based source-to-source translator
3) Type-safe checkpoint restart using **TyCart** [2] / **TypeART** [3]
4) Hierarchical clustering of execution metrics (**Caliper** [4] / **PAPI** [5], **PIN** [6])

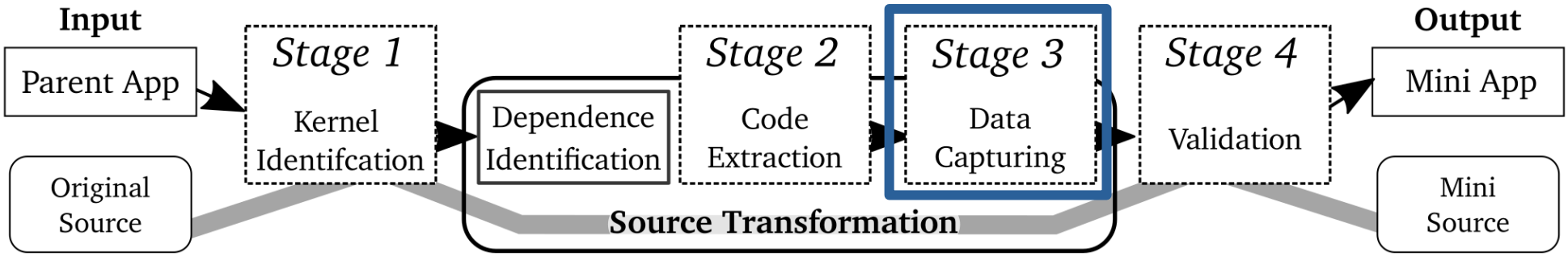
Approach Overview



Tool-support

1) **PIRA** [1] for automatic kernel detection
2) Novel **Clang**-based source-to-source translator
3) Type-safe checkpoint restart using **TyCart** [2] / **TypeART** [3]
4) Hierarchical clustering of execution metrics (**Caliper** [4] / **PAPI** [5], **PIN** [6])

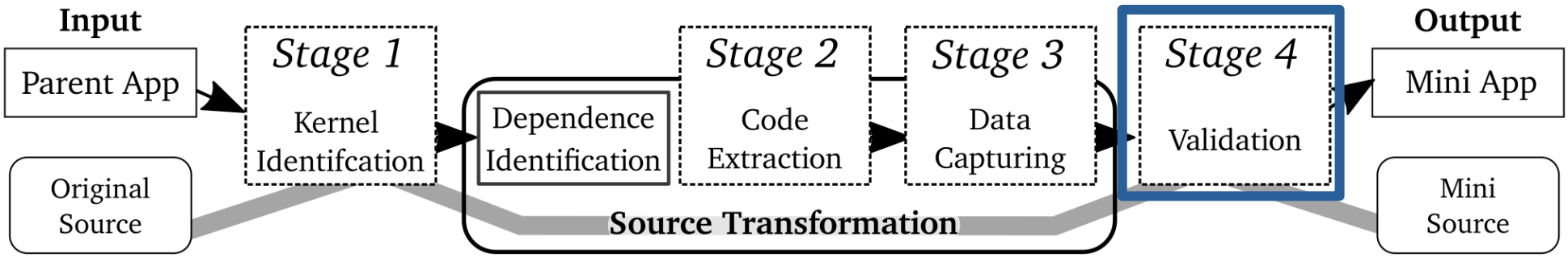
Approach Overview



Tool-support

1) **PIRA** [1] for automatic kernel detection
2) Novel **Clang**-based source-to-source translator
3) Type-safe checkpoint restart using **TyCart** [2] / **TypeART** [3]
4) Hierarchical clustering of execution metrics (**Caliper** [4] / **PAPI** [5], **PIN** [6])

Approach Overview



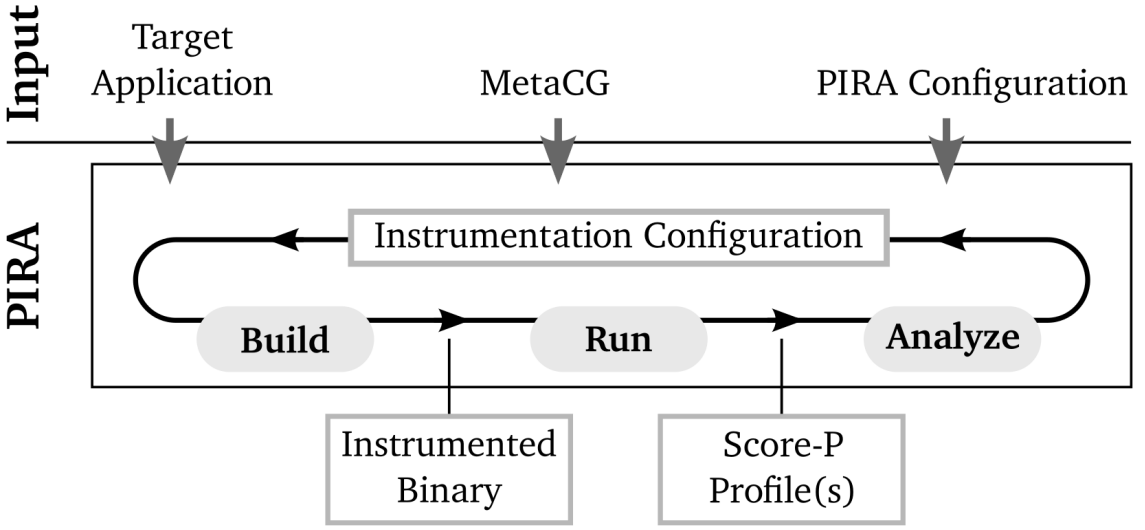
Tool-support

1) **PIRA** [1] for automatic kernel detection
2) Novel **Clang**-based source-to-source translator
3) Type-safe checkpoint restart using **TyCart** [2] / **TypeART** [3]
4) Hierarchical clustering of execution metrics (**Caliper** [4] / **PAPI** [5], **PIN** [6])

I. Kernel Identification

TECHNISCHE UNIVERSITÄT DARMSTADT

- **PIRA** automatically filters short-running functions from measurement
- **PIRA** uses **Extra-P** [7] to construct empirical performance models



- Performance models are used to extrapolate function runtimes for larger inputs
  - Determines points in the program that limit scaling
  - Shows parts that benefit from (additional) parallelism, due to increased workload

II. Source Extraction

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Clang**-based source extractor
  - Works in a mark-and-sweep fashion
- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

```
1  double some_const;// required global variable
2  float compute_X();// non-system function
3  void kernel(int i, double *pd, double *r) {
4    // performs heavy computation with pd
5    r = some_const * compute_X() * ...;
6  }
7  double* compute(int arg_i, double* arg_d) {
8    double *res = malloc(sizeof(double));
9    kernel(arg_i, arg_d, res);// kernel call-site
10   return res;
11 }
```

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

Scientific Computing

II. Source Extraction

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Clang**-based source extractor
  - Works in a mark-and-sweep fashion
- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

```
1  double some_const;// required global variable
2  float compute_X();// non-system function
3  void kernel(int i, double *pd, double *r) {
4    // performs heavy computation with pd
5    r = some_const * compute_X() * ...;
6  }
7  double* compute(int arg_i, double* arg_d) {
8    double *res = malloc(sizeof(double));
9    kernel(arg_i, arg_d, res) // kernel call-site
10   return res;
11 }
```

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

Scientific Computing

II. Source Extraction

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Clang**-based source extractor
  - Works in a mark-and-sweep fashion
- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

```
1  double some_const;// required global variable
2  float compute_X();// non-system function
3  void kernel(int i, double *pd, double *r) {
4    // performs heavy computation with pd
5    r = some_const * compute_X() * ...;
6  }
7  double* compute(int arg_i, double* arg_d) {
8    double *res = malloc(sizeof(double));
9    kernel(arg_i, arg_d, res);// kernel call-site
10   return res;
11 }
```

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

II. Source Extraction

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Clang**-based source extractor
  - Works in a mark-and-sweep fashion
- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

```
1  double some_const;// required global variable
2  float compute_X();// non-system function
3  void kernel(int i, double *pd, double *r) {
4    // performs heavy computation with pd
5    r = some_const * compute_X() * ...;
6  }
7  double* compute(int arg_i, double* arg_d) {
8    double *res = malloc(sizeof(double));
9    kernel(arg_i, arg_d, res);// kernel call-site
10   return res;
11 }
```

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

Scientific Computing

II. Source Extraction

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Clang**-based source extractor
  - Works in a mark-and-sweep fashion
- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

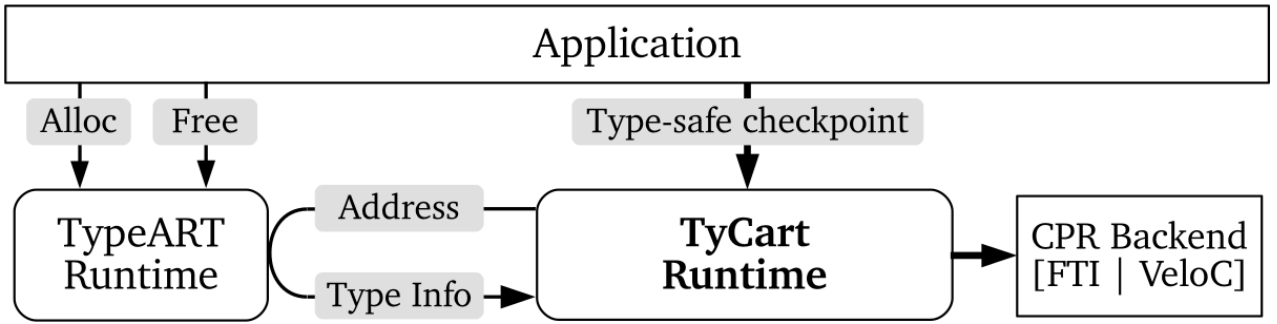
```
1  double some_const;// required global variable
2  float compute_X();// non-system function
3  void kernel(int i, double *pd, double *r) {
4    // performs heavy computation with pd
5    r = some_const * compute_X() * ...;
6  }
7  double* compute(int arg_i, double* arg_d) {
8    double *res = malloc(sizeof(double));
9    kernel(arg_i, arg_d, res);// kernel call-site
10   return res;
11 }
```

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

III. Data Capturing

TECHNISCHE UNIVERSITÄT DARMSTADT

- Use type-safe checkpoint-restart (CPR) interface provided by **TyCart**
  - Built on top of **TypeART** with one sequential as well as **VeloC** [8] and **FTI** [9] backends

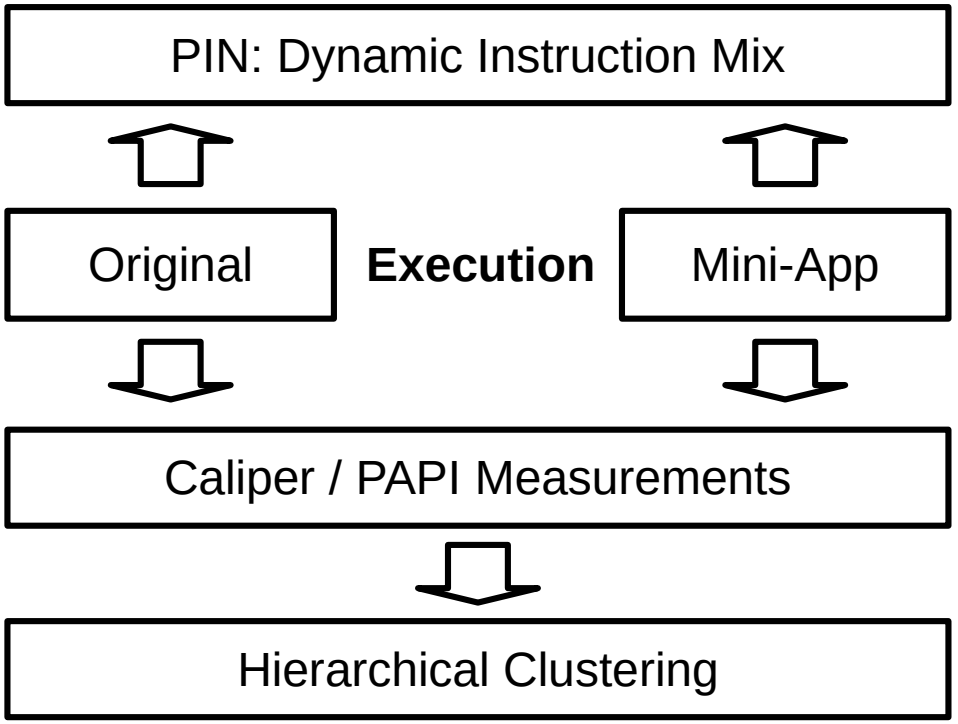


- Calls to CPR inserted into *kernel wrapper* to capture (original app) or restore (mini-app) application-level data
  - **TyCart** allows to query for type and length of an allocation

IV. Validation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Representativeness validation by performing hierarchical clustering of execution metrics
- Hardware performance counters for functions and kernel region
- Dynamic instruction mix for full application execution

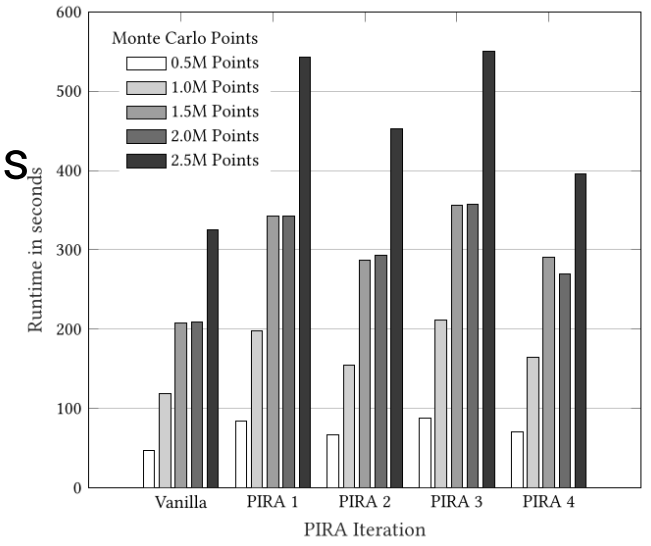


Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

Apply to astrophysics simulation of ~8.5m lines of code

- **PIRA** performs four iterations and identifies three kernels
- Flat profile shows clearly the dominating functions
- Performance models are – as expected – in the number fo Monte-Carlo points evaluated



Function	Runtime (s)		% Total
	Inclusive	Exclusive	
main	490.0	1.2	0.3 %
Int_NN_HF	488.9	1.2	0.3 %
f_NN_HF	488.1	359.9	73.5 %
get_qNN	128.2	26.3	5.4 %
SphericalY	101.9	101.9	20.8 %

Function	performance model
f_NN_HF	$-C_1 + 4.2^{-5} \times N$
get_qNN	$-C_2 + 2.3^{-5} \times N$
SphericalY	$-C_3 + 8.5^{-5} \times (N^{\frac{3}{4}}) \times log2(N)$

Scientific Computing

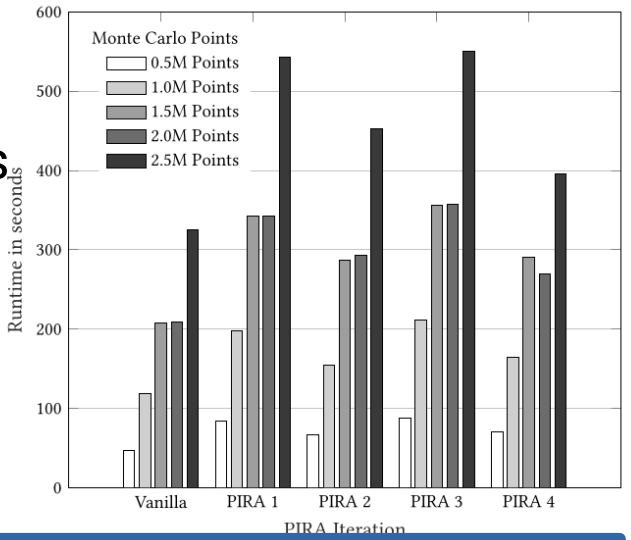
Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

Apply to astrophysics simulation of ~8.5m lines of code

- **PIRA** performs four iterations and identifies three kernels
- Flat profile shows clearly the dominating functions
- Performance models are – as expected – in the number fo Monte-Carlo points evaluated

Function	Runtime (s)		% Total
	Inclusive	Exclusive	
main	490.0	1.2	0.3 %
Int_NN_HF	488.9	1.2	0.3 %
f_NN_HF	488.1	359.9	73.5 %
get_qNN	128.2	26.3	5.4 %
SphericalY	101.9	101.9	20.8 %



Function	performance model
f_NN_HF	$-C_1 + 4.2^{-5} \times N$
get_qNN	$-C_2 + 2.3^{-5} \times N$
SphericalY	$-C_3 + 8.5^{-5} \times (N^{\frac{3}{4}}) \times log2(N)$

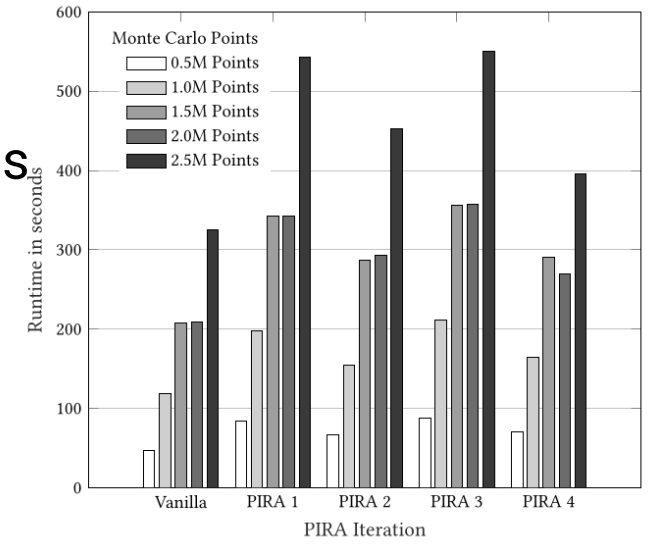
Scientific Computing

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

Apply to astrophysics simulation of ~8.5m lines of code

- **PIRA** performs four iterations and identifies three kernels
- Flat profile shows clearly the dominating functions
- Performance models are – as expected – in the number fo Monte-Carlo points evaluated



Function	Runtime (s)		% Total
	Inclusive	Exclusive	
main	490.0	1.2	0.3 %
Int_NN_HF	488.9	1.2	0.3 %
f_NN_HF	488.1	359.9	73.5 %
get_qNN	128.2	26.3	5.4 %
SphericalY	101.9	101.9	20.8 %

Function	performance model
f_NN_HF	$-C_1 + 4.2^{-5} \times N$
get_qNN	$-C_2 + 2.3^{-5} \times N$
SphericalY	$-C_3 + 8.5^{-5} \times (N^{\frac{3}{4}}) \times log2(N)$

Scientific Computing

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Clang**-based source-to-source translator created Mini-App
  - Manual addition of two includes (using the translator hints)

- Manual addition of Monte-Carlo library code results in MCS Mini-App
  - To expose more potential parallelism to **DiscoPoP**

- Reduction achieved (MCS Mini-App vs Original):
  - Lines of Code: **7674 x**
  - Functions: **380 x**
  - Variables: **3.4 x**
  - Types: **4.9 x**

Metric	Original	Mini-App	MCS Mini-App
Lines of Code	8,571,815	422	1,117
Functions	31,196	46	82
– Reachable	445	33	52
Variables	1,206	183	356
Types	462	18	94

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Clang**-based source-to-source translator created Mini-App
  - Manual addition of two includes (using the translator hints)

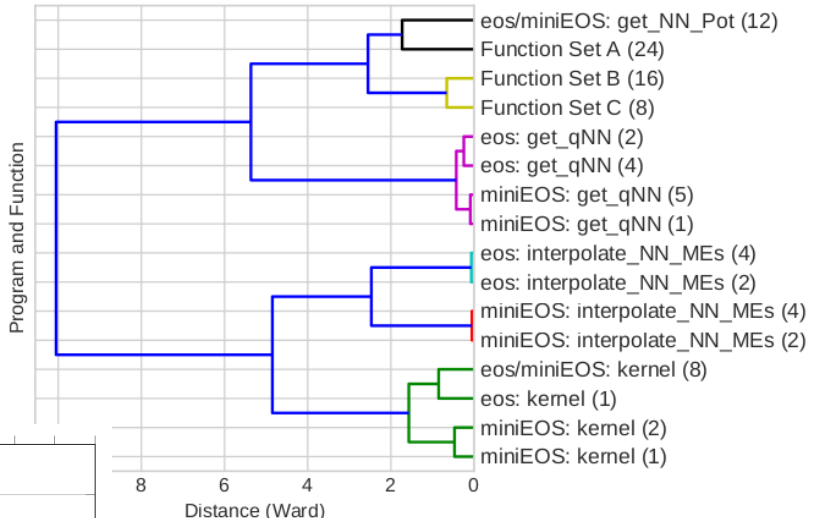
- Manual addition of Monte-Carlo library code results in MCS Mini-App
  - To expose more potential parallelism to **DiscoPoP**

- Reduction achieved (MCS Mini-App vs Original):
  - Lines of Code: **7674 x**
  - Functions: **380 x**
  - Variables: **3.4 x**
  - Types: **4.9 x**

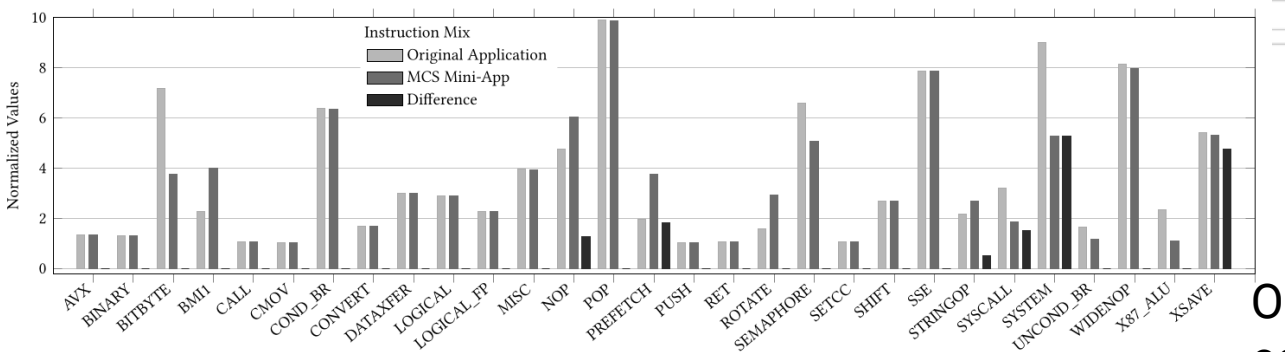
Metric	Original	Mini-App	MCS Mini-App
Lines of Code	8,571,815	422	1,117
Functions	31,196	46	82
– Reachable	445	33	52
Variables	1,206	183	356
Types	462	18	94

Evaluation

- Using **Caliper** to store **PAPI** in JSON format
- Kernel regions are similar in dendrogram
- Dynamic instruction mix is almost identical
  - Some differences in, e.g., system calls, due to more file I/O in original application



Smaller Ward distance is better for regions with the same name



Optimal: light and gray bar are equal in height.

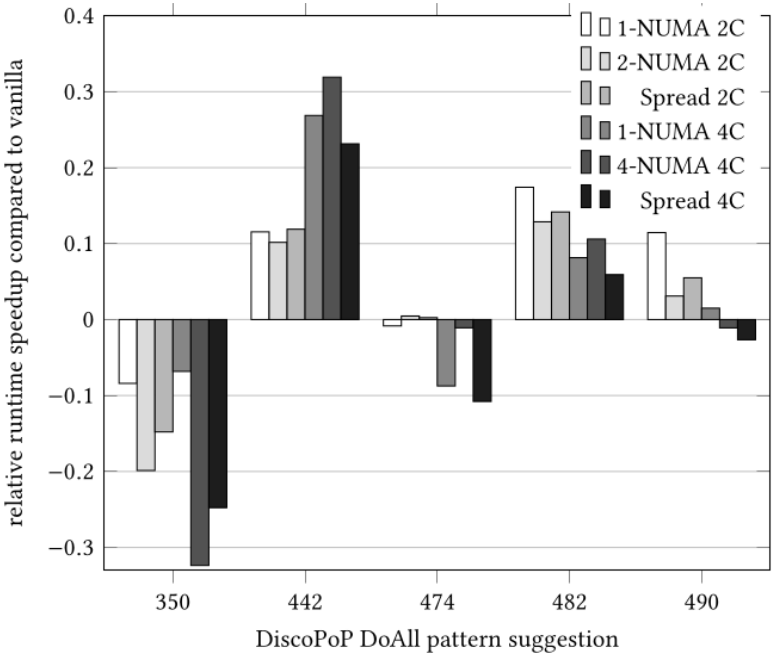
Parallelization

TECHNISCHE UNIVERSITÄT DARMSTADT

- Used the MCS Mini-App for tool-supported parallelization with **DiscoPoP** [10]
  - Identified 42 unique parallelization opportunities in the mini-app
  - Focus on “DoAll” opportunities

- Speedup achieved: 35% w/ 4 threads
- Correctness manually verified in Isabelle/HOL

Pattern	Containing Function	Kernel
350	`get_NN_Pot`	N
442	`get_qNN`	Y
474	`f_NN_HF`	Y
482	`f_NN_HF`	Y
490	`f_NN_HF`	Y



Scientific Computing

Summary / Future Work

TECHNISCHE UNIVERSITÄT DARMSTADT

- Tool-supported approach for mini-app extraction from large-scale applications
  - **PIRA**: automatic kernel identification
  - Source-to-source translator for code extraction
  - **TyCart**: type-safe checkpoint-restart
  - Hierarchical cluster representative-analysis
- Application to astrophysics simulation and subsequent tool-supported parallelization achieved speed-up of 35%

Software Available

github.com/tudasc

github.com/discopop-project

Future: Improve source-to-source translator for C++ w/ multi translation-unit support

References

- [1] Lehr, JP et al.: PIRA – Performance Instrumentation Refinement Automation, 2018.
  DOI: 10.1145/3281070.3281071
- [2] Lehr, JP et al.: Compiler-assisted type-safe checkpointing, 2020.
  DOI: 10.1007/978-3-030-59851-8_1
- [3] Hück, A. et al.: Compiler-aided type tracking for correctness checking of MPI applications, 2018.
  DOI: 10.1109/Correctness.2018.00011
- [4] Boehme, D. et al.: Caliper: Performance Introspection for HPC Software Stacks, 2016.
  DOI: 10.1109/SC.2016.46
- [5] Browne, S.: A Portable Programming Interface for Performance Evaluation on Modern Processors, 2000.
  DOI: 10.1177/109434200001400303
- [6] Luk, CK et al.: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation, 2005.
  DOI: 10.1145/1064978.1065034
- [7] Calotoiu, A. et al.: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes, 2013.
  DOI: 10.1145/2503210.2503277
- [8] Nicolae, B. et al.: VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale, 2019.
  DOI: 10.1109/IPDPS.2019.
- [9] Bautista-Gomez, L. et al.: {FTI}: {H}igh Performance Fault Tolerance Interface for Hybrid Systems, 2011.
  DOI: 10.1145/2063384.2063427
- [10] Li, Z et al.: Unveiling Parallelization Opportunities in Sequential Programs, 2016.
  DOI: 10.1016/j.jss.2016.03.045