Open Source HPC Research Tools at the Institute for Scientific Computing

TECHNISCHE UNIVERSITÄT DARMSTADT

FOSDEM'21 HPC, Big Data, and Data Science Devroom

Sunday, 7th February 2021
Virtual Conference

https://github.com/tudasc

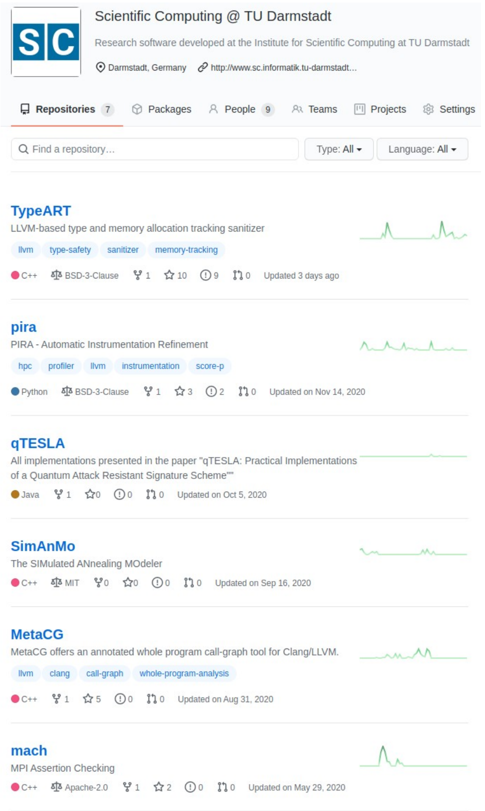
www.sc.informatik.tu-darmstadt.de

Jan-Patrick Lehr jan-patrick.lehr@tu-darmstadt.de
Alexander Hück alexander.hueck@tu-darmstadt.de
Michael Burger michael.burger@tu-darmstadt.de
Tim Jammer tim.jammer@tu-darmstadt.de

SC Scientific Computing

This talk

Outlines open source HPC tools developed at ***Scientific Computing @ TU Darmstadt***



(1)	**PIRA**	(JP Lehr)
(2)	**SimAnMo**	(Michael Burger)
(3)	**MACH**	(Tim Jammer)
(4)	**TypeART**	(Alexander Hück)

PIRA

Performance Instrumentation Refinement Automation

Automatically selects and filters functions for instrumentation-based profiling using static and dynamic information to iteratively reduce measurement overhead [1,2]

PIRA
https://github.com/tudasc/pira

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

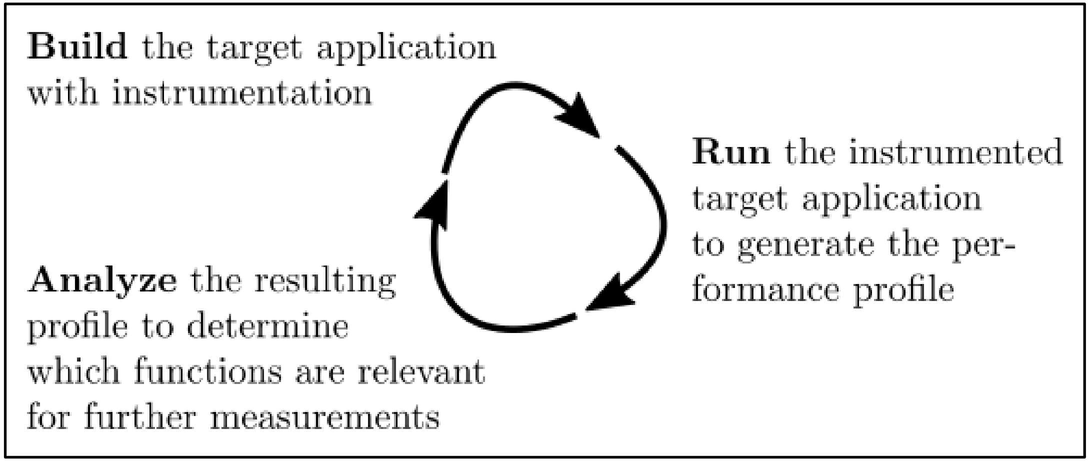
Overview

- **Automatically** reduce overhead by removing irrelevant instrumentation in target application
  ➔**Focus measurement to relevant parts and reduce runtime overhead**

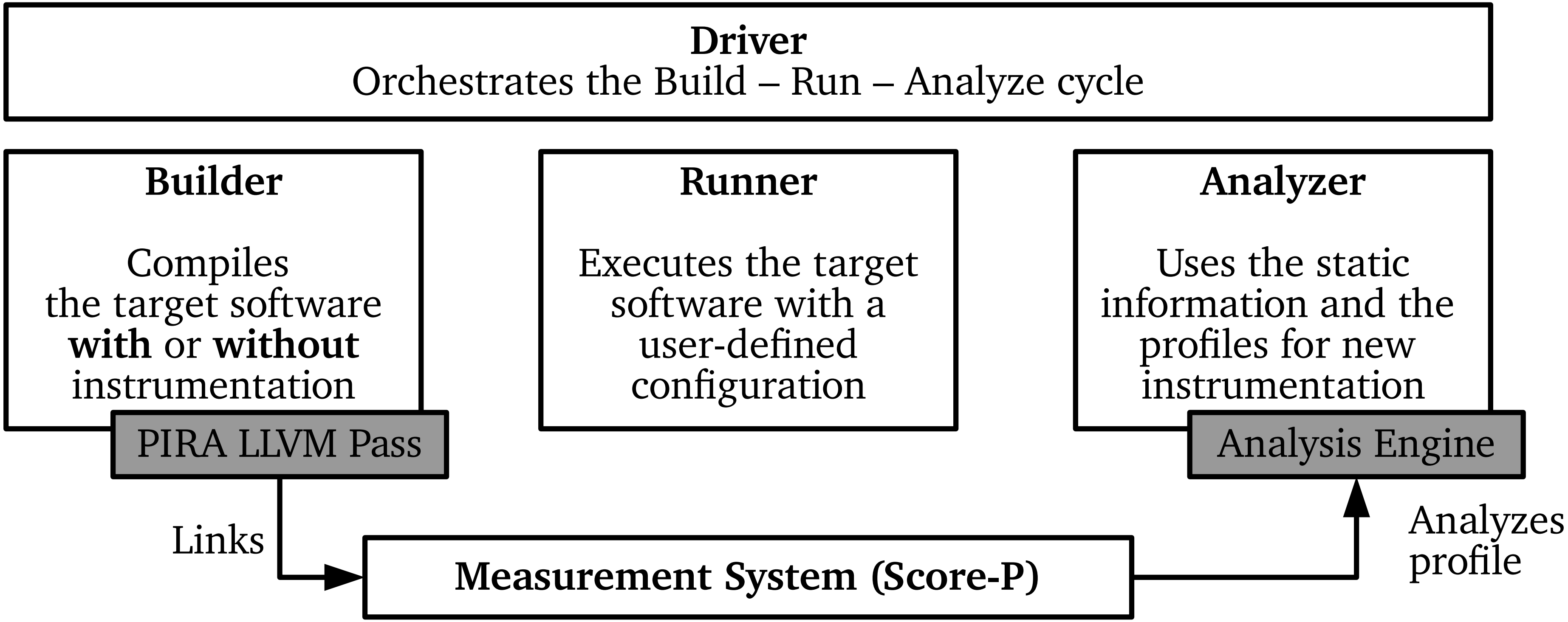
- Initially guess which functions of the application are relevant, i.e., are likely to consume a lot of runtime (static analysis)

- Iteratively refine the selection to those functions of the application that actually consume a lot of runtime (dynamic analysis)

- From functions already identified, expand selection and add functions heuristically to the instrumentation (static analysis)



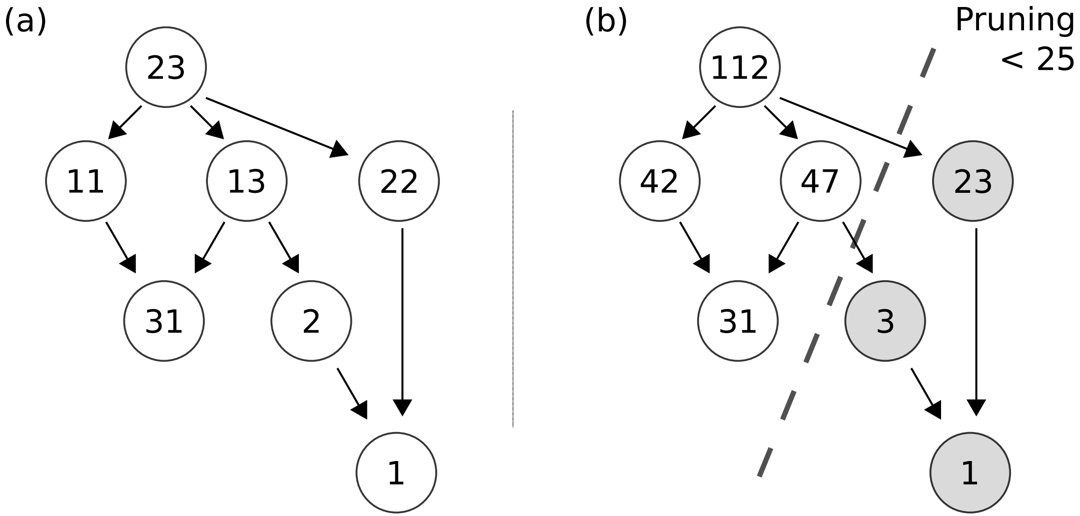
PIRA Architecture



PIRA Analysis Engine

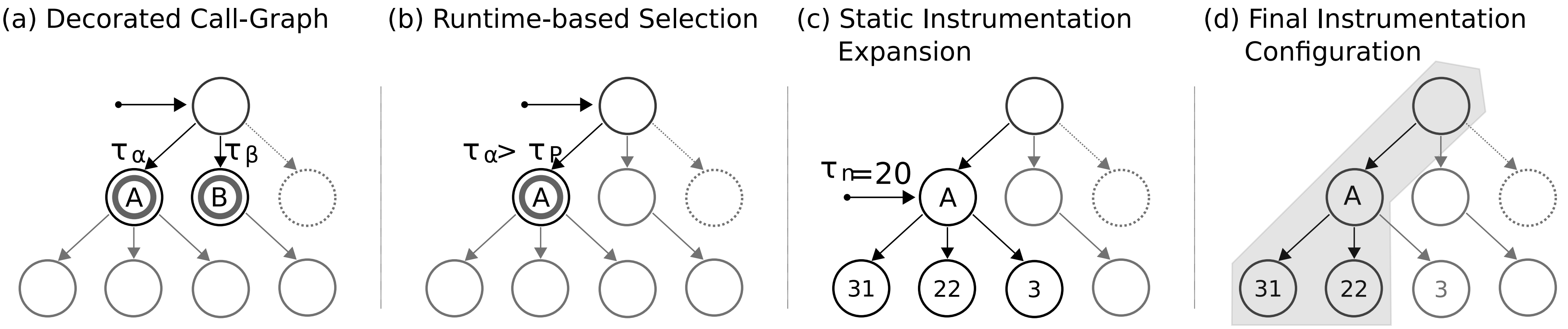
- Whole-program call-graph based analysis using different heuristics for selection
- Part of MetaCG [3] software package (https://github.com/tudasc/MetaCG)
- Static Selection based on statement aggregation idea [4]

Example



PIRA Analysis Engine

- Dynamic selection uses profile information, e.g., runtime for filtering out irrelevant functions
  - Irrelevant nodes are filtered from instrumentation
  - Relevant nodes are heuristically expanded to add further functions
- Hot-Spot analysis

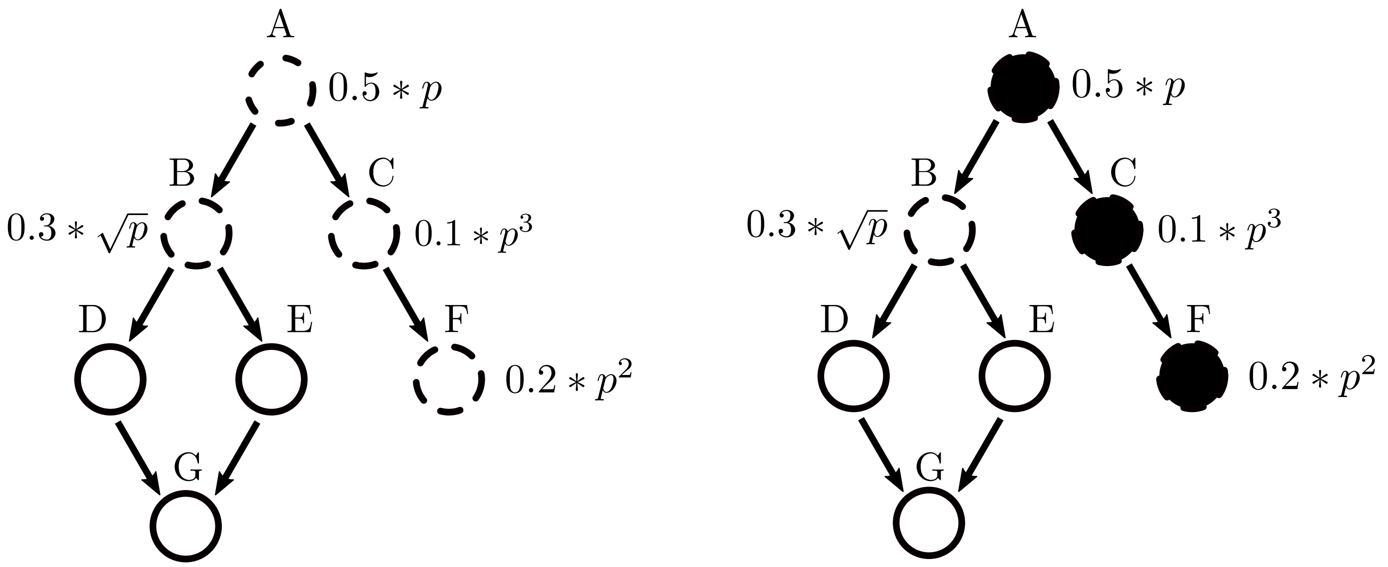


Hot-Spot Analysis Results

Benchmark	Paradigm	Number of Functions	Score-P Overhead	PIRA Overhead
SU 2	Sequential	15,775	4,571 %	74 %
433.milc	Sequential	313	307 %	18 %
473.astar	Sequential	334	3,349 %	205 %
126.lammps	MPI	1,449	2.5 %	0 %

PIRA Analysis Engine

- Dynamic analysis can be based on empirical performance models, i.e., scaling analysis
  - Only filter step is shown, as expansion step is according to hot-spot analysis



Scaling Analysis Results

TECHNISCHE UNIVERSITÄT DARMSTADT

Benchmark	Paradigm	Score-P w/ filtering	PIRA I Overhead	PIRA II Overhead
SU 2	Sequential	212 %	74 %	11 %
433.milc	Sequential	208 %	18 %	0 %
473.astar	Sequential	377 %	198 %	18 %
MILC	MPI	405 %	N / A	312 %

SC Scientific Computing

PIRA Overview

- A “profile-guided profiler” for C/C++ using Score-P
- Driver code implemented in Python 3
- Analysis Engine implemented in modern C++
  - Available as separate software package MetaCG [X]
- **Features**
  - Automatic function filtering and expansion
  - Automatic creation of MPI function filters for Score-P
  - Automatic creation of empirical performance models using Extra-P [5]
- **Currently in development**
  - Load imbalance detection
  - SimAnMo integration

Current Release
0.2

Next Release
0.3.0

Dependencies
- *Clang/LLVM 10*
- *Python 3*
- *OpenMPI (MPICH not tested)*
- *MetaCG*
- *Bear*
- *LLNL’s wrap.py*

PIRA
https://github.com/tudasc/pira
License BSD 3-Clause

References

- [1] JP Lehr, A Hück, and C Bischof, “PIRA: performance instrumentation refinement automation.” In: 5th ACM SIGPLAN International Workshop on Artificial Intelligence and Empirical Methods for Software Engineering and Parallel Computing Systems (AI-SEPS 2018). https://doi.org/10.1145/3281070.3281071
- [2] JP Lehr, A Calotoiu, C Bischof and F Wolf, "Automatic Instrumentation Refinement for Empirical Performance Modeling." In: 2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools). https://doi.org/10.1109/ProTools49597.2019.00011
- [3] JP Lehr, A Hück, Y Fischler, and C Bischof, “MetaCG: annotated call-graphs to facilitate whole-program analysis.” In the 11th ACM SIGPLAN International Workshop on Tools for Automatic Program Analysis (TAPAS 2020). https://doi.org/10.1145/3427764.3428320
- [4] C Iwainsky and C Bischof, "Calltree-Controlled Instrumentation for Low-Overhead Survey Measurements" In 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). https://doi.org/10.1109/IPDPSW.2016.54.
- [5] A Calotoiu, T Hoefler, M Poke, and F Wolf, “Using automated performance modeling to find scalability bugs in complex codes.” In: International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13). https://doi.org/10.1145/2503210.2503277

SimAnMo – Simulated Annealing Modeler [1],[2]

Creating models for the development of a program's runtime on the actual hardware depending on the input size.

https://github.com/tudasc/simanmo

License MIT

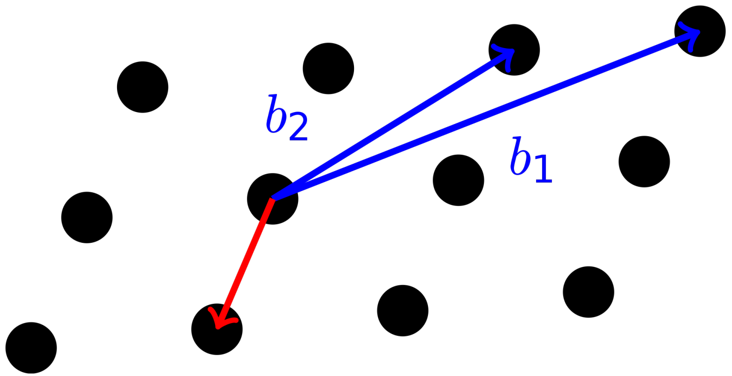
Michael Burger

michael.burger@tu-darmstadt.de

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Each encryption / signature scheme based on mathematical problem
- Parameters for the problem generation determine the scheme's
  - Security
  - Efficiency
- For practicability, secure but efficient schemes required
- For cryptanalysis, reliable models for time to solve the problems are required
- Problems are not solvable by definition



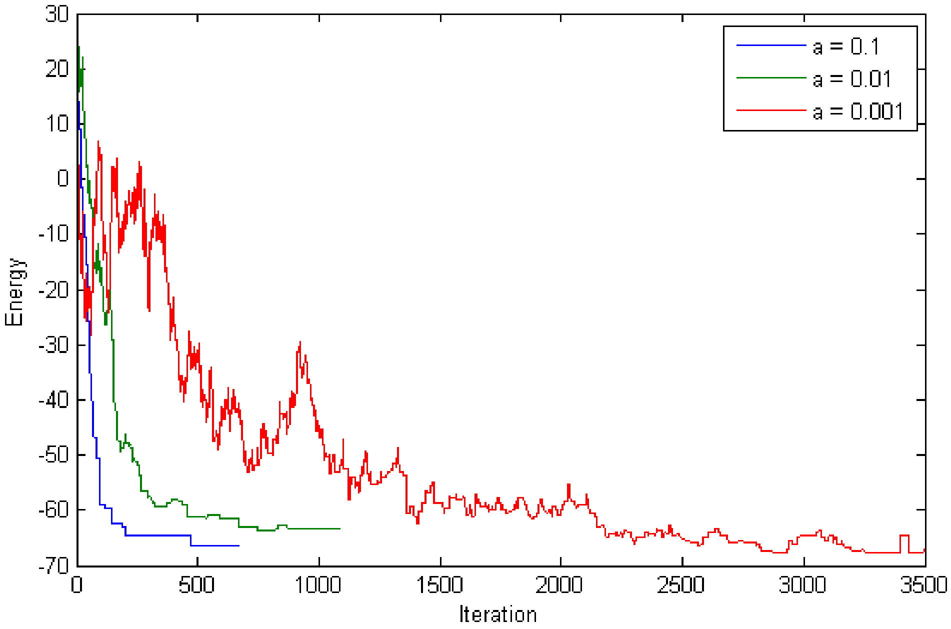
Shortest Vector Problem (SVP) from lattice-based cryptography [3]

SC Scientific Computing

Concepts of SimAnMo

TECHNISCHE UNIVERSITÄT DARMSTADT

- Two step approach:
  - Collect training data for small instances
  - Generate / Evaluate model
- Parallel simulated annealing to heuristically minimize model "cost"
- Easy extendable and adaptable [2] (model types, cost metrics)
- Highly parameterizable procedure
- Automated result-report generation

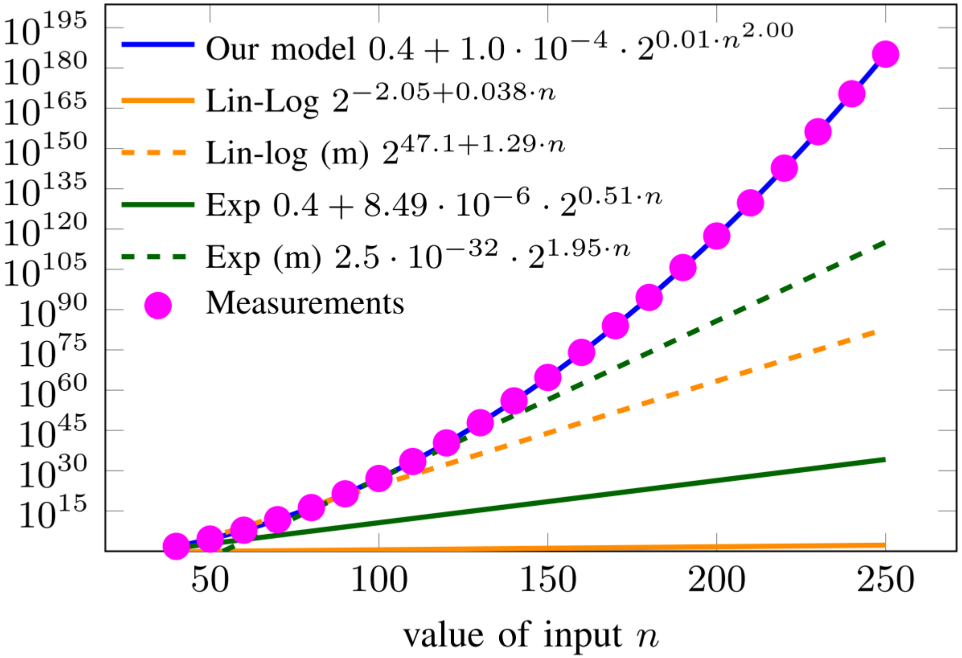


https://web.media.mit.edu/~patorpey/classes/mas864/ps9/annealing.html

Scientific Computing

Results for unusual algorithms

- Runtimes growing super-exponentially are modeled accurately
- Very important for cryptanalysis
- Example: Enumeration algorithm (ENUM) for finding the shortest vector in the n-dimensional grid
  - Our Model predicts the measurements correctly
  - The two standard approaches Lin-Log and Exp fail
  - Even if they get more points for model generation, corresponding to (m) suffix in model name



Conclusion

SimAnMo enables reliable model generation with little user effort

Development related information:

- Support for Microsoft Visual Studio, Makefile and Eclipse Project, C++17
- Relies in Eigen, ALGLIB and (optionally) NAG

Roadmap:

- Integrating factorial runtime models
- Employing SimAnMo in PIRA

https://github.com/tudasc/simanmo

License MIT

References

[1] M. Burger, C. Bischof, A. Calotoiu, T. Wunderer and F. Wolf, "**Exploring the Performance Envelope of the LLL Algorithm**" In: 2018 IEEE International Conference on Computational Science and Engineering (CSE), Bucharest, 2018, pp. 36-43, doi: **10.1109/CSE.2018.00012**

[2] M. Burger, G. N. Nguyen and C. Bischof, "**Developing Models for the Runtime of Programs With Exponential Runtime Behavior**" In: 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), GA, USA, 2020, pp. 109-125, doi: **10.1109/PMBS51919.2020.00015**

[3] Burger M., Bischof C., Krämer J., "**p3Enum: A New Parameterizable and Shared-Memory Parallelized Shortest Vector Problem Solver**" In: Rodrigues J. et al. (eds) Computational Science – ICCS 2019. ICCS 2019. Lecture Notes in Computer Science, vol 11540. Springer, Cham. **https://doi.org/10.1007/978-3-030-22750-0_48**

TECHNISCHE UNIVERSITÄT DARMSTADT

MACH - MPI Assertion Checking

checks if the newly defined communicator assertions
(current MPI draft specification) hold for a MPI program

MACH

https://github.com/tudasc/mach

License Apache 2.0

Tim Jammer tim.jammer@tu-darmstadt.de

Scientific Computing

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- The 2019 draft specification of MPI defines new communicator info hints:
  - mpi_assert_allow_overtaking
  - mpi_assert_exact_length
  - mpi_assert_no_any_tag
  - mpi_assert_no_any_source
- If these assertions are given a more optimized implementation man be used

Scientific Computing

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- The 2019 draft specification of MPI defines new communicator info hints:
  - mpi_assert_allow_overtaking
  - mpi_assert_exact_length
  - mpi_assert_no_any_tag
  - mpi_assert_no_any_source
- If these assertions are given a more optimized implementation man be used

- We propose to automatically detect if these assertions hold using a Clang/LLVM compiler pass

⇒ This saves the developers effort to manually check if these assertions hold

SC Scientific Computing

Usage

TECHNISCHE UNIVERSITÄT DARMSTADT

- MACH is implemented as an LLVM analysis pass
- Usage is quite straightforward:
- mpicc -cc=clang -O2 -Xclang -load -Xclang path/to/libmpi_assertion_checker.so src.c
- Usage via LLVM's **opt** tool is also possible

  Example:

```
user@system:~/path$ mpicc -cc=clang -O2 -Xclang -load -Xclang path/to/
    libmpi_assertion_checker.so test.c
No conflicts detected, try to use mpi_assert_allow_overtaking for better performance
You can also safely specify mpi_assert_no_any_tag for better performance
You can also safely specify mpi_assert_no_any_source for better performance
You can also safely specify mpi_assert_exact_length for better performance
Successfully executed the pass
```


Scientific Computing

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Evaluated our tool using 48 different small self-written MPI programs.
- Specifically designed to test various cases of mpi_allow_overtaking
- ✓ Minimal impact on compilation time
  - -ftime-report reports that our pass uses only 0.3% (0.0014 seconds) of the compilation time for test program (≈ 400 lines of code)

- ✓ All cases correct for mpi_no_any_tag
- ✓ All cases correct for mpi_no_any_source
  - simple constant checking
  - no need for extensive evaluation
- ✓ All cases correct for mpi_exact_length
  - more refined implementation desirable
  - with more extensive evaluation
- Most cases correct for mpi_allow_overtaking
  - ✓ our tool only suggests using the assertion when it is safe to do so
  - × but it misses some of the cases where one can specify the assertion (see next slides)

Scientific Computing

Missed cases of allow_overtaking

Ring communication scheme

```
int pre = (rank + 1) % comm_size;
int next = (rank - 1) % comm_size;
// "forward" communication
MPI_Send(&buffer, 1, MPI_INT, next, TAG,
    MPI_COMM_WOLRD);
MPI_Recv(&buffer, 1, MPI_INT, next, TAG,
    MPI_COMM_WOLRD ,MPI_STATUS_IGNORE);
// "backward" communication
MPI_Send(&buffer, 1, MPI_INT, pre, TAG,
    MPI_COMM_WOLRD);
MPI_Recv(&buffer, 1, MPI_INT, pre, TAG,
    MPI_COMM_WOLRD, MPI_STATUS_IGNORE);
```

Listing 1: Ring communication scheme

(pretend that one rank communicates "backward" first to avoid deadlock)

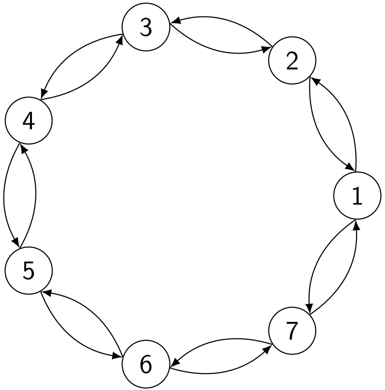


Figure: Ring communication scheme

Missed cases of allow_overtaking

Ring communication scheme

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int pre = (rank + 1) % comm_size;
int next = (rank - 1) % comm_size;
// "forward" communication
MPI_Send(&buffer, 1, MPI_INT, next, TAG,
    MPI_COMM_WOLRD);
MPI_Recv(&buffer, 1, MPI_INT, next, TAG,
    MPI_COMM_WOLRD, MPI_STATUS_IGNORE);
// "backward" communication
MPI_Send(&buffer, 1, MPI_INT, pre, TAG,
    MPI_COMM_WOLRD);
MPI_Recv(&buffer, 1, MPI_INT, pre, TAG,
    MPI_COMM_WOLRD, MPI_STATUS_IGNORE);
```

Listing 2: Ring communication scheme

(pretend that one rank communicates "backward" first to avoid deadlock)

- Static analysis can not prove that pre and next are different
  - if executed with 2 ranks they are same
- Therefore our tool has to assume the sending operations may conflict
- Using different message tags for "forward" and "backward" communication mitigates this problem

SC Scientific Computing

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Our prototype implemenation shows that detecting if the newly defined communicator info hints hold is possible by static analysis only in many cases
- We plan to extend our tool, so that it insert the specification of the assertion if it holds
- Our code is available online:

MACH

https://github.com/tudasc/mach

License Apache 2.0

SC Scientific Computing

TECHNISCHE UNIVERSITÄT DARMSTADT

TypeART

LLVM-based type and memory allocation tracking sanitizer

TypeART

https://github.com/tudasc/typeart

License BSD 3-Clause

Alexander Hück **ahueck** **alexander.hueck@tu-darmstadt.de**

Scientific Computing

Type Unsafe APIs in HPC

Many libraries provide low-level C-APIs based on a type-less void pointer

1. MPI:

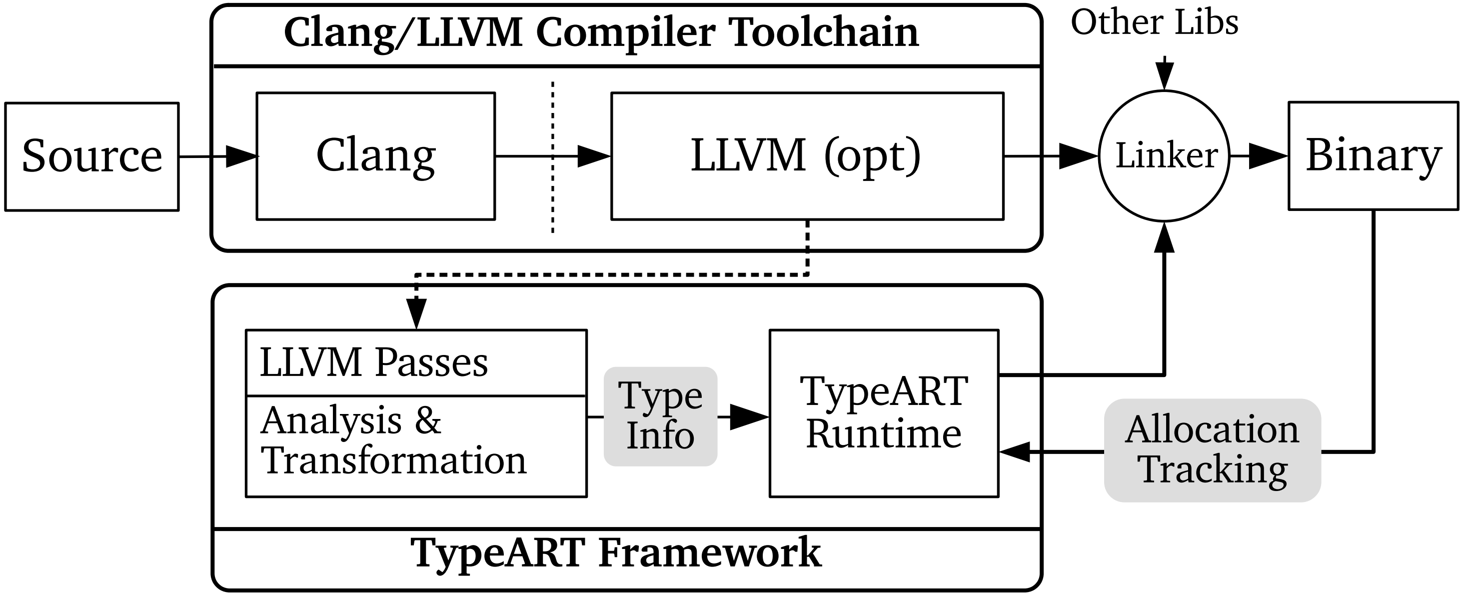
MPI_Send(buffer, n, **MPI_DOUBLE**, ...)

1. Data is specified as a type-less void* buffer
2. Data length and type is user-specified

2. Checkpointing:

VELOC_Mem_protect(id, buffer, n, **sizeof(double)**)

TypeART Framework

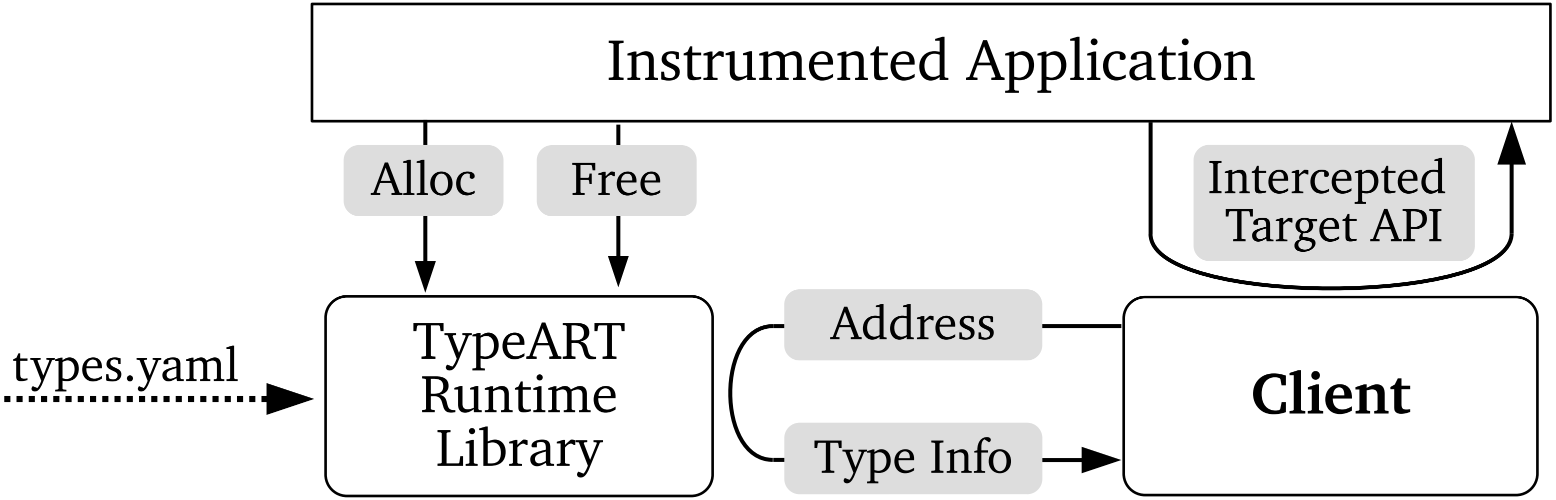


Typical usage with LLVM **opt** in three stages (see demo folder in our repository)

→ *Pseudo:* **opt** -load *typeart -heap* | **opt** -O3 | **opt** -load *typeart* -stack

- Heap is done first as type information may be lost during optimization

Runtime Checks



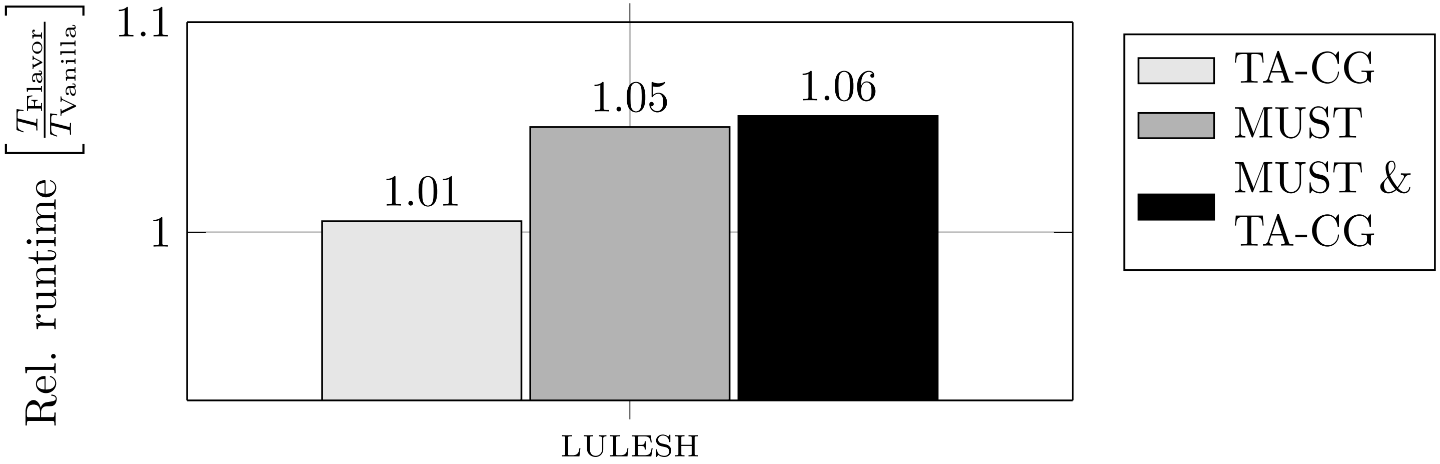
Client uses the information collected by TypeART (runtime)

- MPI: MUST – a dynamic MPI correctness checker **[1, 2]**
- Checkpointing: A wrapper around a checkpoint library **[3]**

Brief Evaluation

Runtime/memory impact depends on number of tracked allocations

- Runtime overhead for small LULESH run, checking MPI communication with MUST (see **[2]**)



- Memory overhead **< 1.1**

Conclusion

TypeART can ensure type-safety for applications that use low-level APIs

Development related information:

- LLVM 10, CMake and (mostly) C++17
- CI pipeline using GitHub Actions, code coverage using Coveralls

Roadmap (coming soon'ish):

- Support for OpenMP and explicit support for array cookies (C++)

TypeART

https://github.com/tudasc/typeart

References

[1] A. Hück, J.-P. Lehr, S. Kreutzer, J. Protze, C. Terboven, C. Bischof, M. S. Müller. “**Compiler-aided type tracking for correctness checking of MPI applications**” In: 2nd International Workshop on Software Correctness for HPC Applications (Correctness). IEEE, 2018, pp. 51–58. DOI: **10.1109/Correctness.2018.00011**

[2] A. Hück, J. Protze, J.-P. Lehr, C. Terboven, C. Bischof, M. S. Müller. “**Towards compiler-aided correctness checking of adjoint MPI applications**” In 4th International Workshop on Software Correctness for HPC Applications (Correctness). IEEE/ACM, 2020, pp. 40–48. DOI: **10.1109/Correctness51934.2020.00010**

[3] J.-P. Lehr, A. Hück, M. Fischer, C. Bischof. “**Compiler-assisted type-safe checkpointing**” In 35th International Conference on High Performance Computing. Springer, 2020, pp. 5–18. DOI: **10.1007/978-3-030-59851-8_1**

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

PIRA
https://github.com/tudasc/pira
License | BSD 3-Clause

"Profile-guided profiling for C/C++"

SimAnMo
https://github.com/tudasc/SimAnMo
License | MIT

"Empirical performance modeling"

MACH
https://github.com/tudasc/mach
License | Apache 2.0

"MPI Assertion Checking"

TypeART
https://github.com/tudasc/typeart
License | BSD 3-Clause

"C/C++ type and memory allocation tracking"