Compiler-aided type tracking for correctness checking of MPI applications

TECHNISCHE UNIVERSITÄT DARMSTADT

Alexander Hück, Jan-Patrick Lehr, Sebastian Kreutzer, Joachim Protze, Christian Terboven, Christian Bischof, Matthias S. Müller

Jan-Patrick Lehr (jan.lehr@sc.tu-darmstadt.de)

Correctness'18: Second International Workshop on Software Correctness for HPC Applications

MUST

MUST [1] is a scalable, dynamic MPI correctness checker developed at RWTH Aachen.

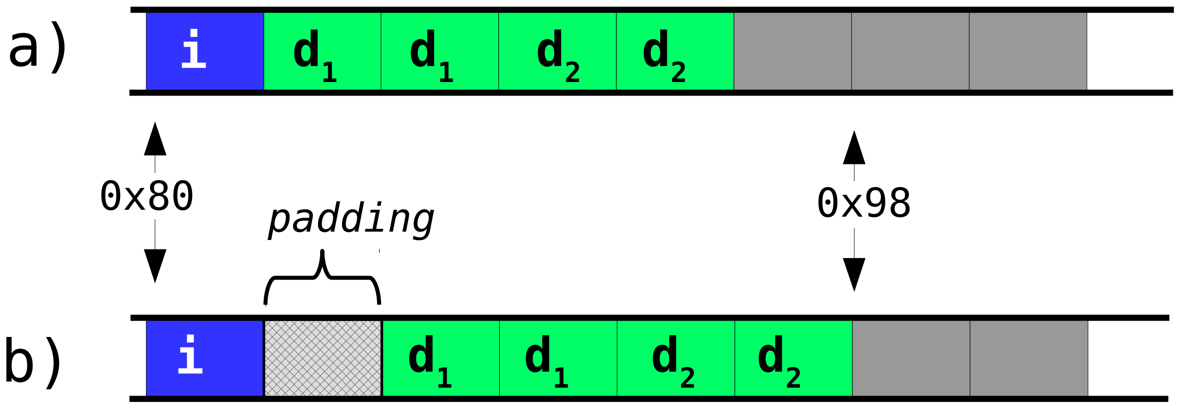
- Capabilities include, e.g.:
  - Deadlocks resulting from MPI calls
  - MPI datatype correctness checks, but only at phase two of the MPI message transfer phases

- https://doc.itc.rwth-aachen.de/display/CCP/Project+MUST

[1] T. Hilbrich, J. Protze, M. Schulz, B. R. de Supinski, and M. S. Müller, **“MPI runtime error detection with MUST: Advances in deadlock detection”**, Scientific Programming, vol. 21, no. 3-4, pp. 109–121, 2013.

Pitfalls of user-defined types

```
struct S {int a; double d[2];};
struct S s;
```



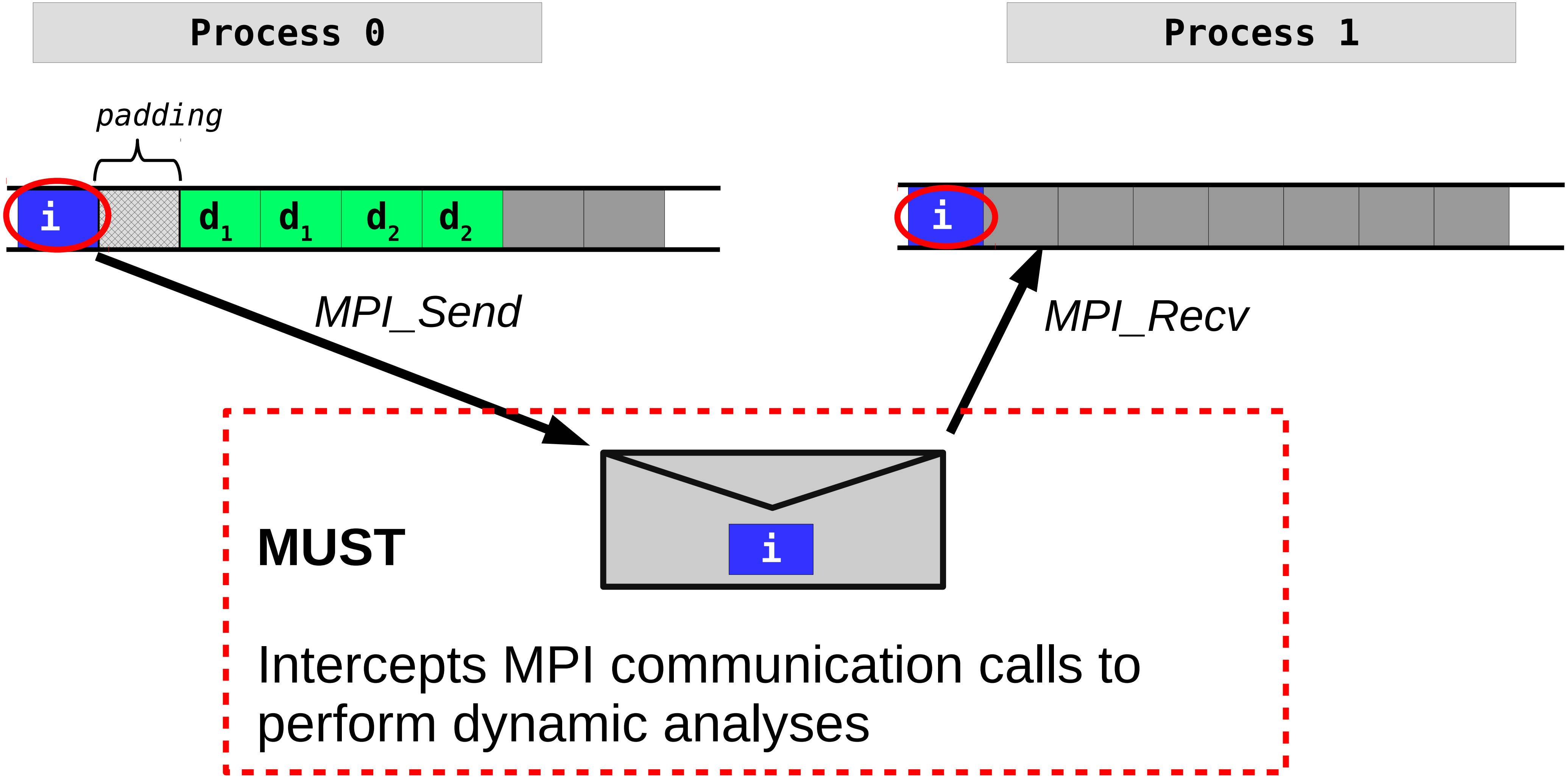
- Two potential memory layouts on some architectures for **`struct S`**

Scenario 1/2:
MPI Send & Receive

```
struct S {int a; double d[2];};
struct S s;
struct S *pS = &s;

MPI_Send(pS, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
MPI_Recv(pS, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
```

MPI Communication & MUST

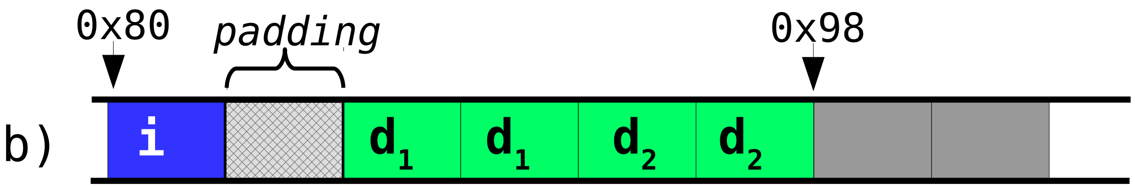


MUST

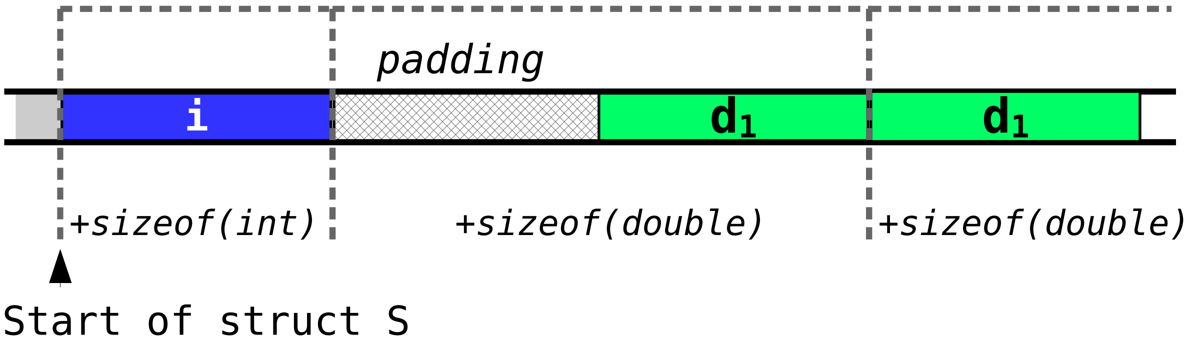
Intercepts MPI communication calls to perform dynamic analyses

Scenario 2/2:
MPI Send & Receive

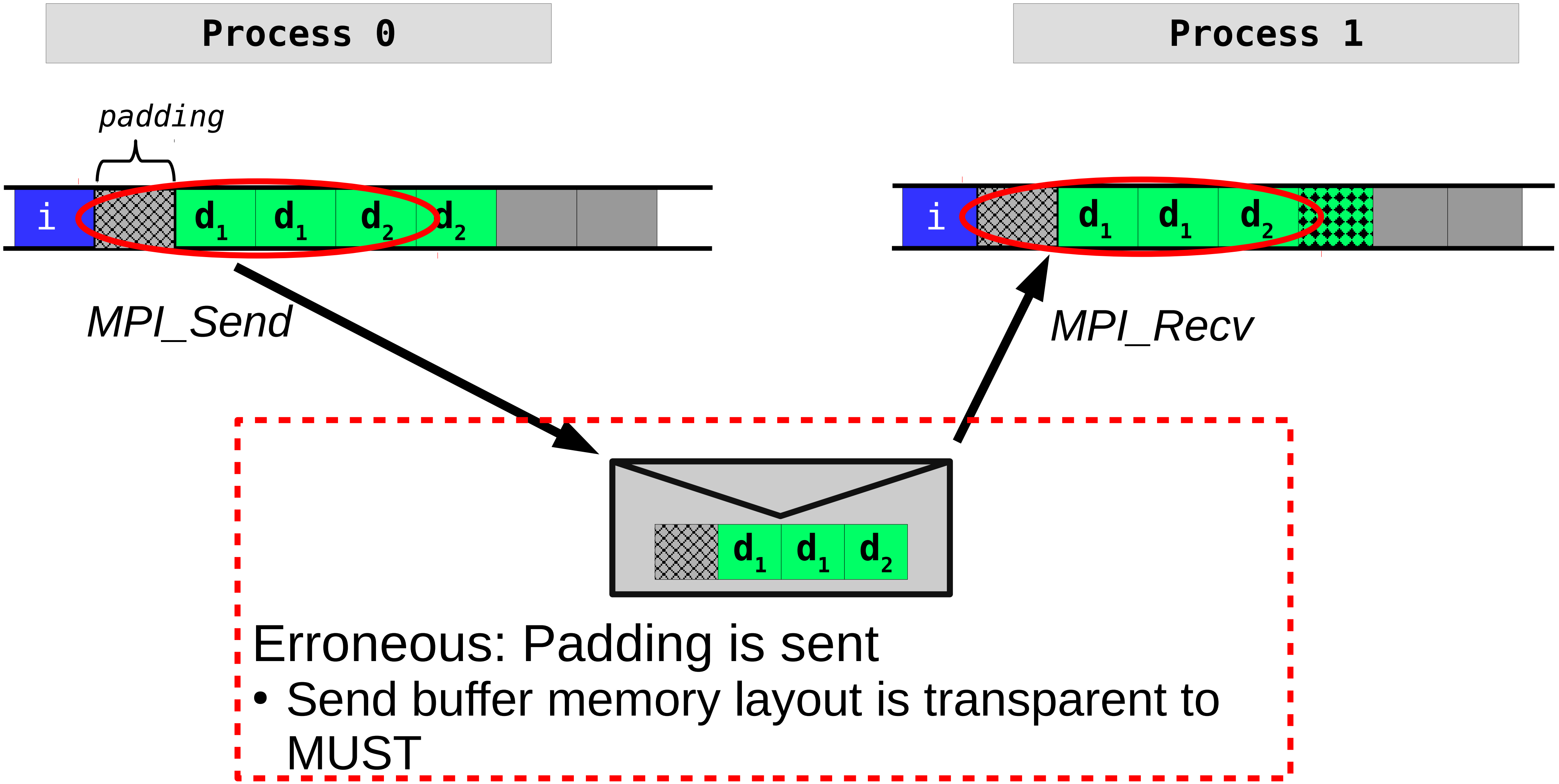
- `struct S {int a; double d[2];};`



```
MPI_Send(pS+sizeof(int),2,MPI_DOUBLE,1,0,MPI_COMM_WORLD);
MPI_Recv(pS+sizeof(int),2,MPI_DOUBLE,1,0,MPI_COMM_WORLD);
```



MPI Communication & MUST



Erroneous: Padding is sent

- Send buffer memory layout is transparent to MUST

Requirements & Approach

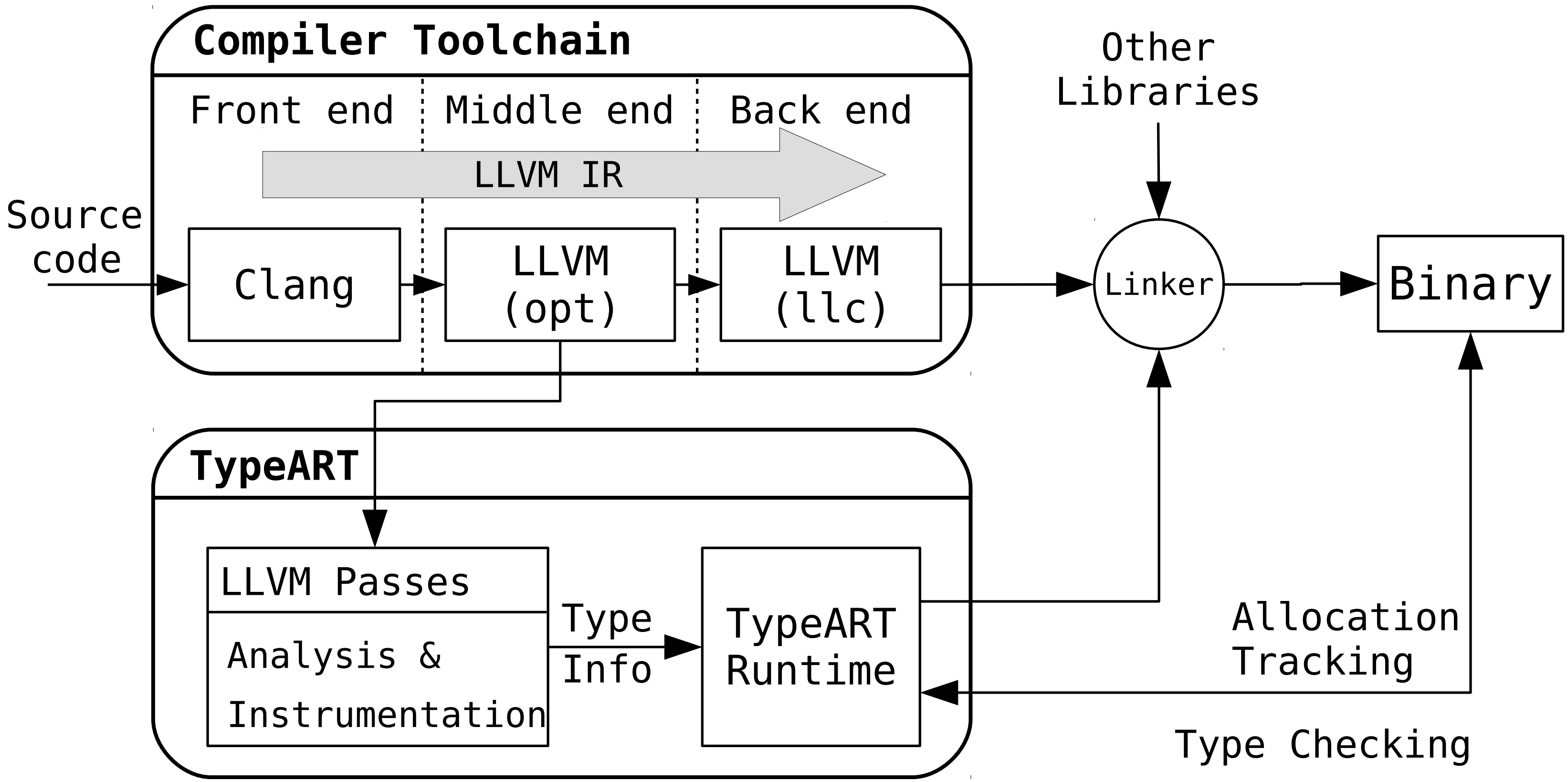
Requirements

- Ability to provide type information for arbitrary memory address at runtime
  - Solution: Store the type (built-in and user-defined) and the extent for every memory allocation relevant to MPI

Approach

- Instrument relevant allocations at compile time with calls to our runtime library to track and provide necessary type information to MUST

TypeART – Workflow



TypeART – Instrumentation

C code

```
float *pd = (float*) malloc(n * sizeof(float));
MPI_Send(pd, n, MPI_FLOAT, 0, 1, MPI_COMM_WORLD);
```

LLVM IR code of malloc

```
%1 = call i8 * @malloc(i64 %0) ; %0 = n*sizeof(float)
%2 = bitcast i8 * %1 to float *
```

TypeART – Instrumentation

C code

```
float *pd = (float*) malloc(n * sizeof(float));
MPI_Send(pd, n, MPI_FLOAT, 0, 1, MPI_COMM_WORLD);
```

LLVM IR code of malloc

```
%1 = call i8 * @malloc(i64 %0) ; %0 = n*sizeof(float)
```

```
%2 = udiv i64 %0, 4         ; %2 = %0/sizeof(float) = n
call void @__typeart_alloc(i8 * %1,i32 5,i64 %2,i32 0)
```

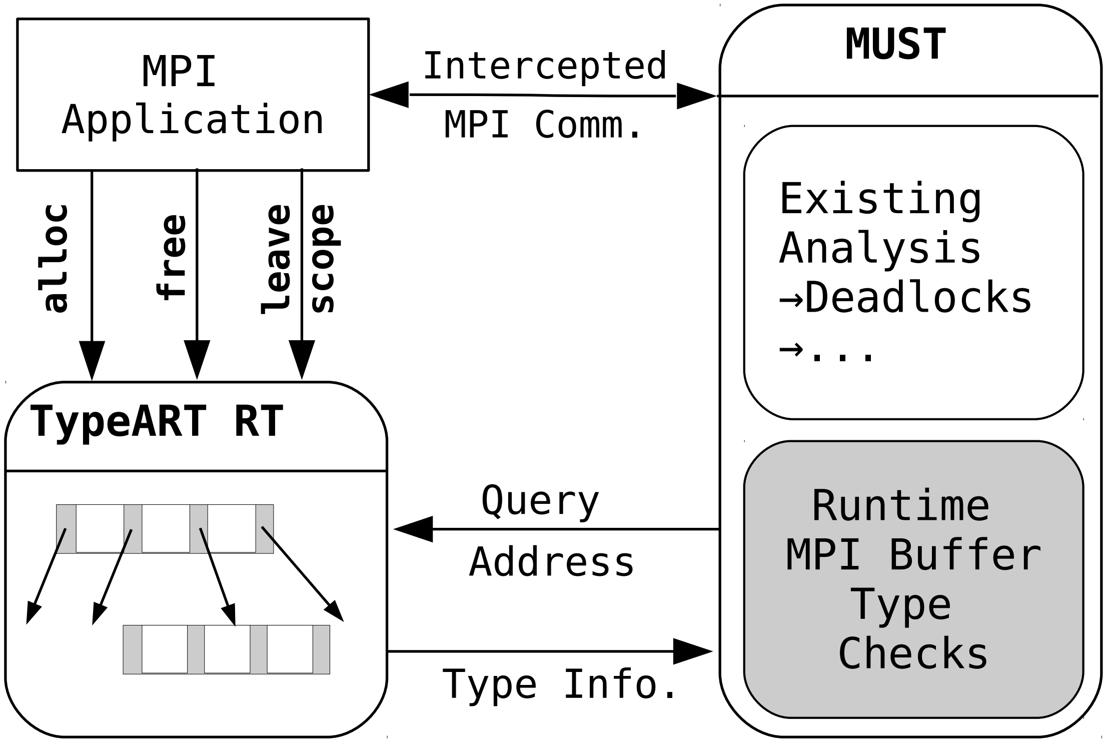
```
%3 = bitcast i8 * %1 to float *
```

TypeART – MUST interplay

- TypeART provides C interface for type information query at runtime

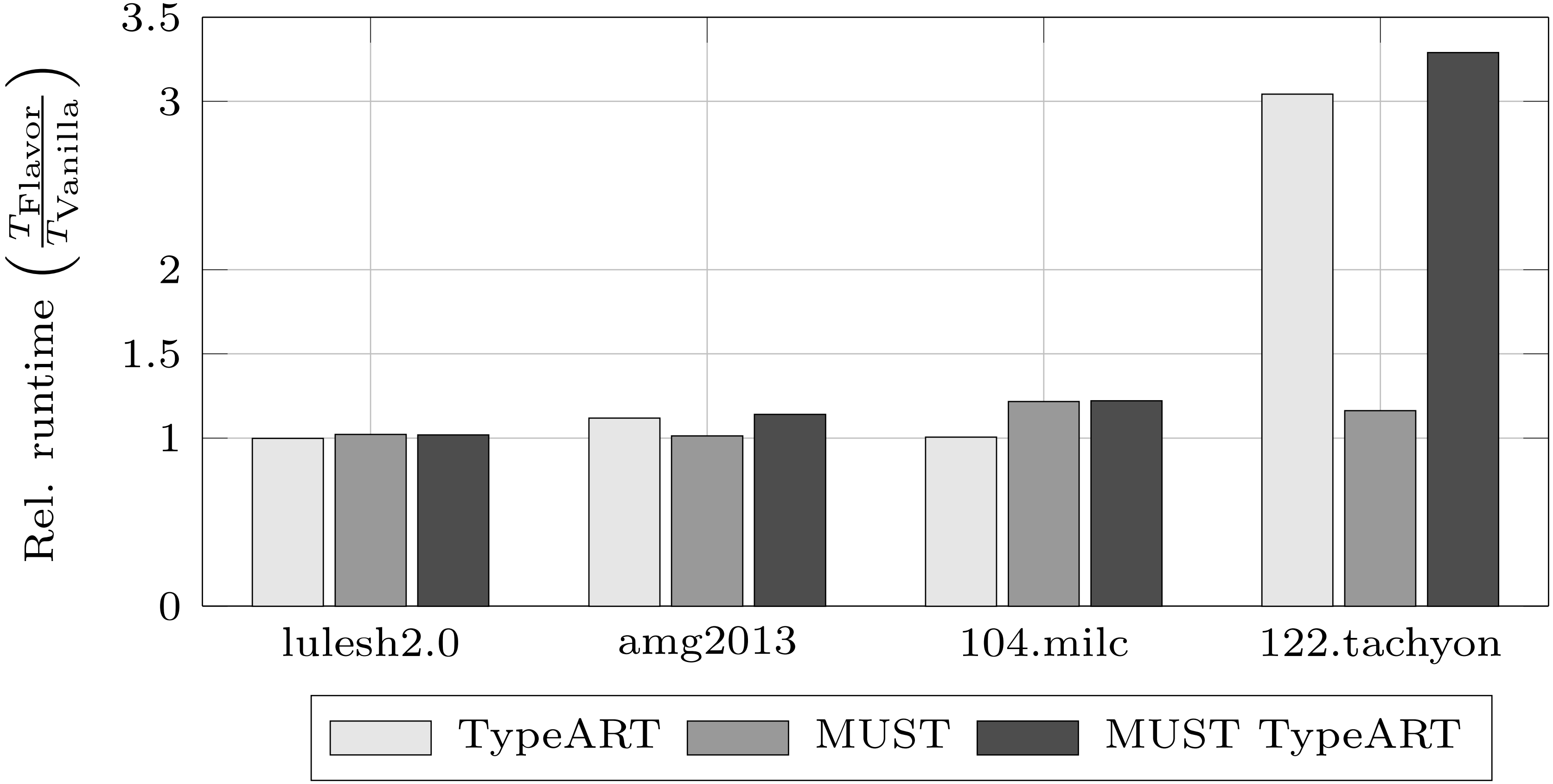
Inserted function calls to RT

- **`alloc`**: memory allocations
- **`free`**: heap memory de-allocation
- **`leave scope`**: Stack memory deallocation



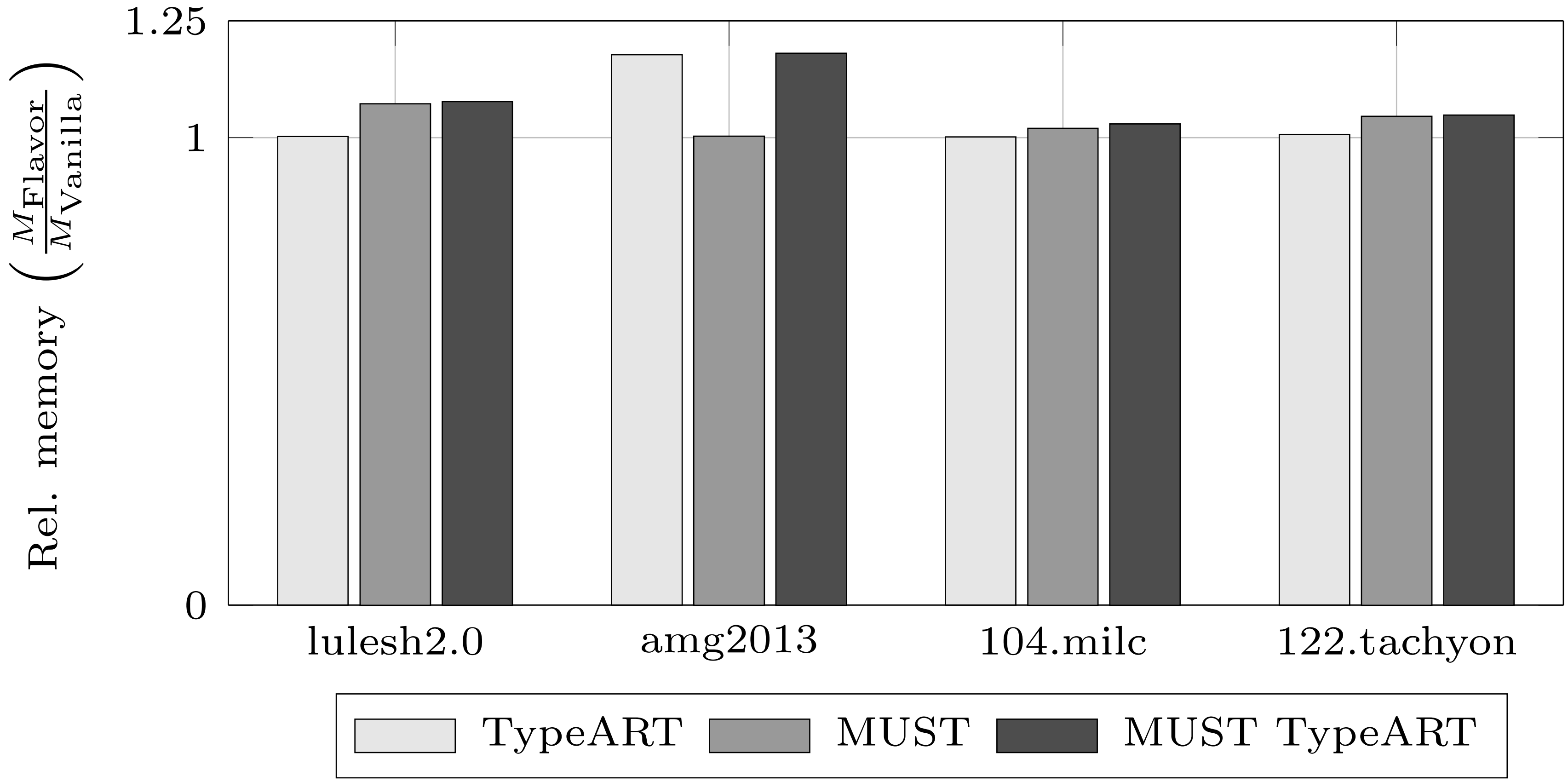
TypeART – Evaluation (1/5)

- Induced runtime overhead



TypeART – Evaluation (2/5)

- Induced memory overhead



TypeART – Evaluation (3/5)

- Overall memory allocation sites found statically during compilation with TypeART LLVM passes

	Memory Operations				User-def
	Heap	Free	Stack (%)	Global (%)	Types
lulesh2.0	14	6	54 (21.0)	80 (100)	10
amg2013	1,491	1,152	958 (40.7)	653 (99.4)	61
104.milc	91	64	207 (21.3)	736 (95.4)	25
122.tachyon	80	51	579 (2.0)	372 (97.3)	50

TypeART – Evaluation (4/5)

- Dynamically tracked memory allocation statistics at runtime

	Traced Memory Operations				
	Total Global	Total Heap	Total Stack	Max. Stored Heap	Max. Stack Depth
lulesh2.0	0	525,060	34,149	76	21
amg2013	1	27,587,586	2,943	20,736,474	80
104.milc	34	41,638	5,876	79	26
122.tachyon	10	13,759	78,307,707	13,677	277

TypeART – Evaluation (5/5)

- MPI type related checks performed by MUST

	MPI Type Checks		
	Total	Unique Checked	Missed
lulesh2.0	40,694	16	0
amg2013	1,906	542	0
104.milc	9,206	84	0
122.tachyon	482	482	0

Conclusion

We implemented a type tracking system for MPI-relevant data allocations and extended MUST to use the available information to detect type mismatches.

- The approach uses LLVM to determine the respective memory allocation location and inserts instrumentation for tracking the dynamic type information
- No harmful MPI problems were found in the evaluation
- The runtime overhead is generally reasonable
  - Three test cases: Runtime overhead less than 1.5x
  - Factor 3.3 due to more than 78 million tracked stack allocations

TypeART – Future Work

- MPI + OpenMP
  - Thread safety of TypeART RT
    - Introduce thread-local stack allocation tracking in addition to global heap tracking

Acknowledgements

This work was partly funded by the Hessian LOEWE initiative within the Software-Factory 4.0 project.

The work of Jan-Patrick Lehr is supported by the Graduate School of Computational Engineering within the Excellence Initiative of the German Federal and State government.