Investigating the Usage of MPI at Argument-Granularity in HPC Codes

Tim Jammer, Alexander Hück, Joachim Jenke and Christian Bischof
Scientific Computing @ TUDa & IT Center @ RWTH

tim.jammer@tu-darmstadt.de

https://github.com/tudasc/mpi-arg-usage

Motivation

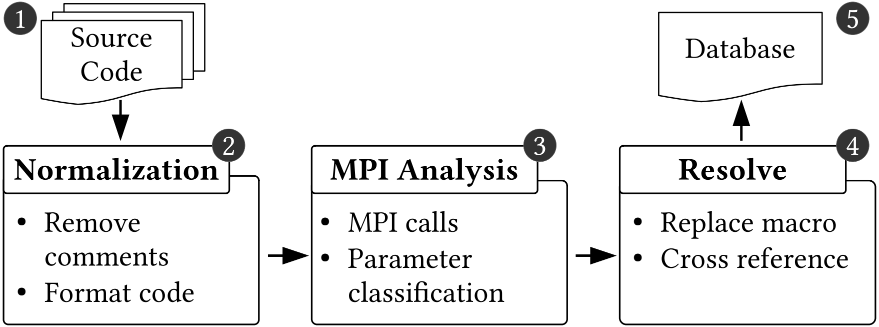
- Understanding MPI usage is important for
  - Optimization,
  - Correctness tools,
  - Benchmarks,
  - Education, ...
- Laguna et al. published a comprehensive analysis of MPI usage in open-source HPC codes.
- We present more detail about common usage patterns, as we take the arguments into account

Ignacio Laguna, Ryan Marshall, Kathryn Mohror, Martin Ruefenacht, Anthony Skjellum, and Nawrin Sultana.
A Large-Scale Study of MPI Usage in Open-Source HPC Applications.
In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19. ACM, 2019.

Example: Datatype Construction

```
MPI_Datatype struct_type, vector_type; // Opaque type handles
// Construct struct type:
MPI_Type_create_struct(num_members, block_length,
                       offsets, member_types, &struct_type);
// Construct vector of struct type:
MPI_Type_vector(count, block_length_v, stride,
                struct_type, &vector_type);
// Commit handle, so MPI library knows about vector type & send data:
MPI_Type_commit(&vector_type);
MPI_Send(buffer, 1, vector_type, ...);
```

Automatic Analysis of Source Code



Limitations of our approach

wrong results

```
MPI_Datatype struct_type, vector_type; // Opaque type handles
// Construct struct type:
MPI_Type_create_struct(num_members, block_length,
                       offsets, member_types, &struct_type);
// Construct vector of struct type:
MPI_Type_vector(count, block_length_v, stride,
                struct_type, &vector_type);
std::swap(vector_type,struct_type);
// Commit handle, so MPI library knows about vector type & send data:
MPI_Type_commit(&vector_type);
MPI_Send(buffer, 1, vector_type, ...);
```

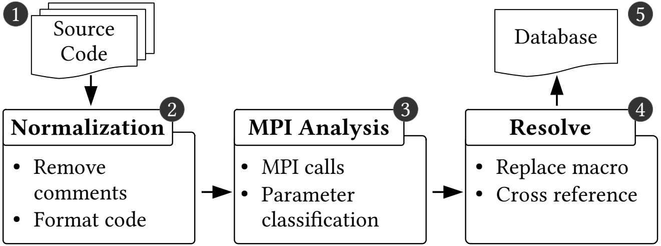
Limitations of our approach

correct but inaccurate results

```
MPI_Datatype get_datatype(){
 MPI_Datatype type_to_create;
 MPI_Type_create_struct(num_members, block_length,
                        offsets, member_types, &type_to_create);
 MPI_Type_commit(&type_to_create);
 return type_to_create;
}
void communicate(){
 MPI_Datatype type_to_use = get_datatype();
 MPI_Send(buffer, 1, type_to_use, ...);
}
```

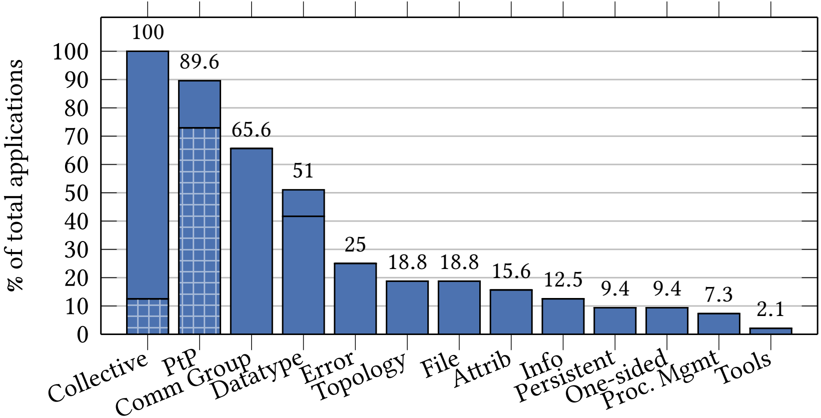
Automatic Analysis of Source Code

Analysis at source code level



- Allows to rapidly screen many applications
- no need to invoke the Preprocessor
- no need to manage dependencies
- no need to choose a build configuration

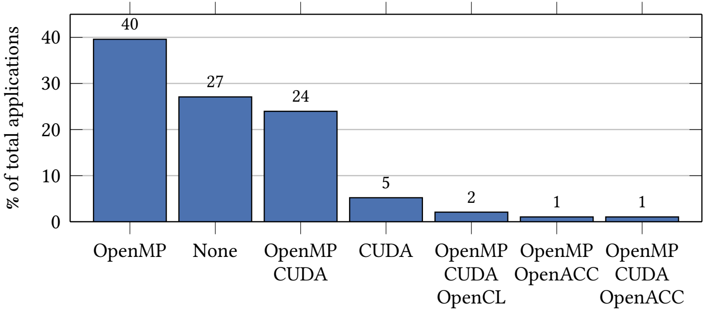
Overview: Used MPI features



min MPI version	# applications
1.0	35
2.0	35
3.0	25
4.0	0

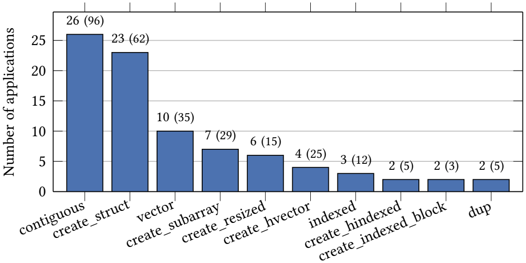
Hatched area denotes non-blocking operations

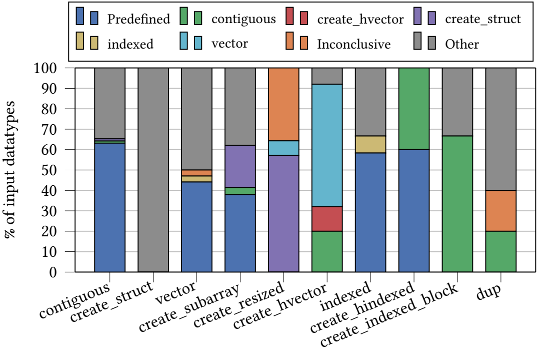
Usage of Hybrid Programming Models (MPI+X)



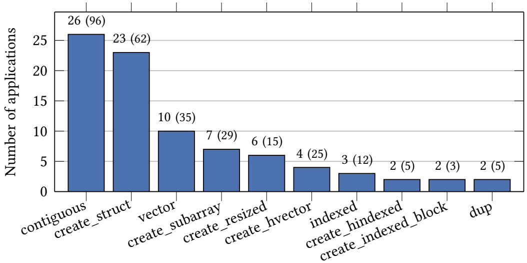
OpenMP Feature	# applications
Target Offload	8
Tasks & Taskloop	4

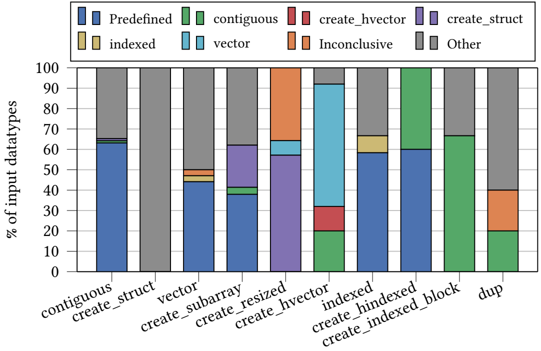
Creation of Derived Types





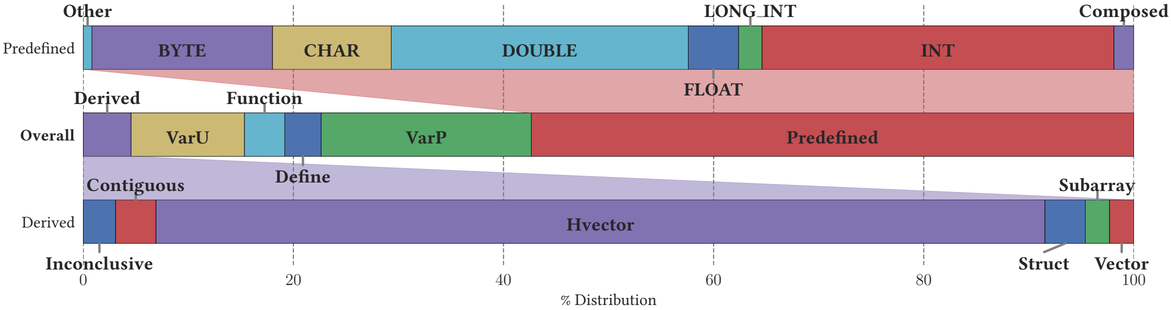
Creation of Derived Types





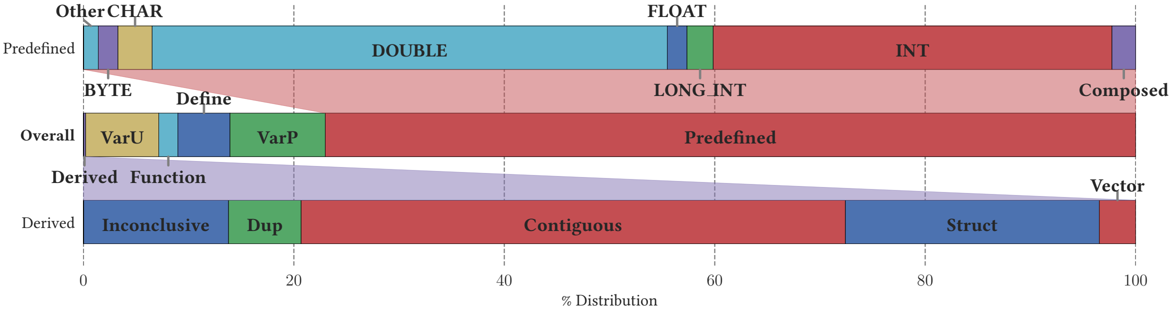
- $\geq$ 17% of type creations are nested

Datatype Usage in Pt2Pt communication



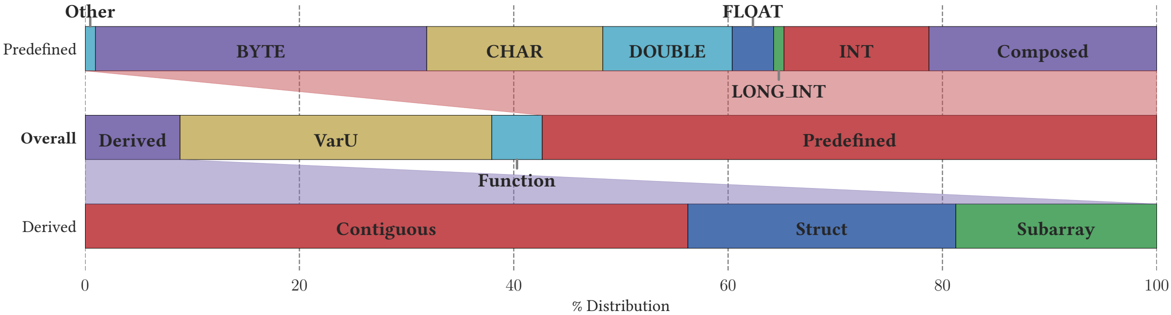
- VarU: Variable with no cross-referencing, possibly derived type
- VarP: Variable with no cross-referencing, definitely a predefined type

Datatype Usage in collective communication



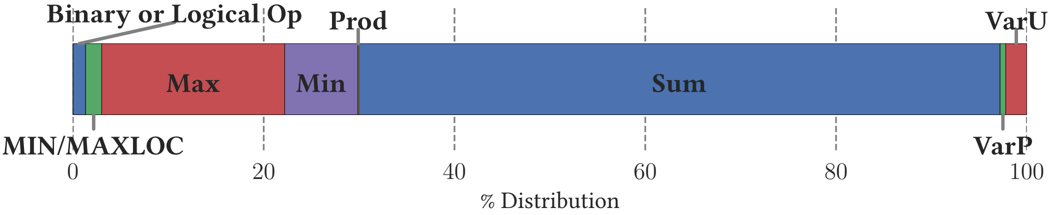
- VarU: Variable with no cross-referencing, possibly derived type
- VarP: Variable with no cross-referencing, definitely a predefined type

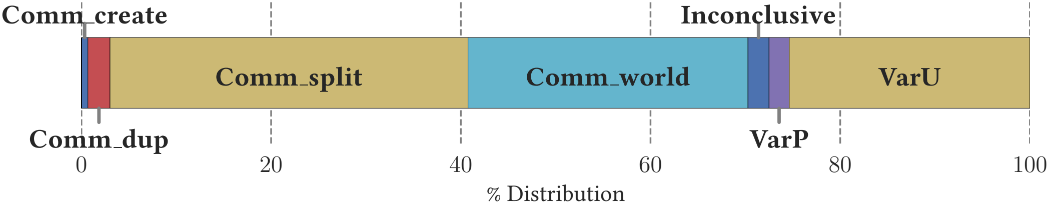
Datatype Usage in I/O operations



- VarU: Variable with no cross-referencing, possibly derived type
- VarP: Variable with no cross-referencing, definitely a predefined type

Usage of MPI_Op and Communicator for collectives





Usage of Tag, Rank and Count

- **Count**: When a derived datatype is used: `'1'` is usually used as the count argument ($\approx 90\%$)
- **Count**: When communicating an integer, a constant count is more likely to be used ($\approx 57\%$ compared to $\approx 10\%$ for floating point types)
- **Count**: $\approx \frac{2}{3}$ of collectives use a constant count (which is usually `'1'`, $\approx 88\%$)
- **Rank**: Collective operations usually use `'0'` as rank argument ($\approx 80\%$)
- **Tag**: Wildcards (`MPI_ANY_TAG` and `MPI_ANY_SOURCE`) are rare ($\approx 3\%$)

Conclusion

- No huge changes regarding the call counts since Laguna et al.'s study

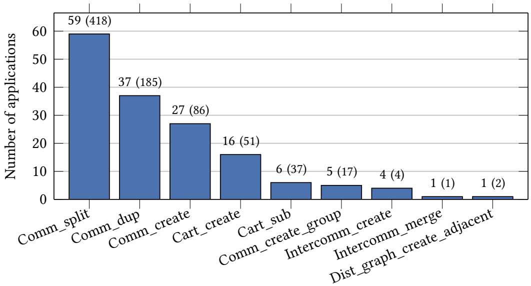
- MPI_Type_contigous is the most common derived datatype
- Communication with derived datatypes usually use a count of 1 ($\approx$ 90%)
- MPI_Sum is the most common reduction operation ($\approx$ 60%)
- Usually reduction operations are done to rank 0 ($\approx$ 80%)
- MPI_ANY_SOURCE and MPI_ANY_TAG wildcards are rare ($\approx$ 3%)

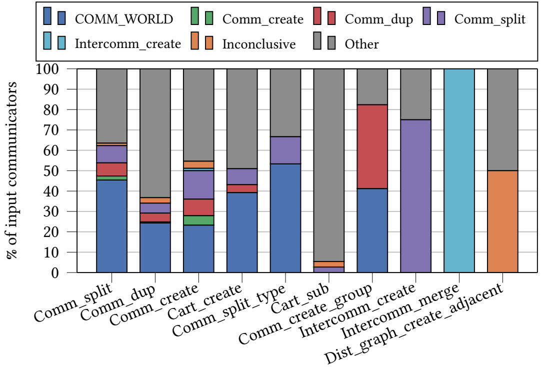
https://github.com/tudasc/mpi-arg-usage

Backup slides

Following slides are backup slides, please refer to the paper for more detailed results

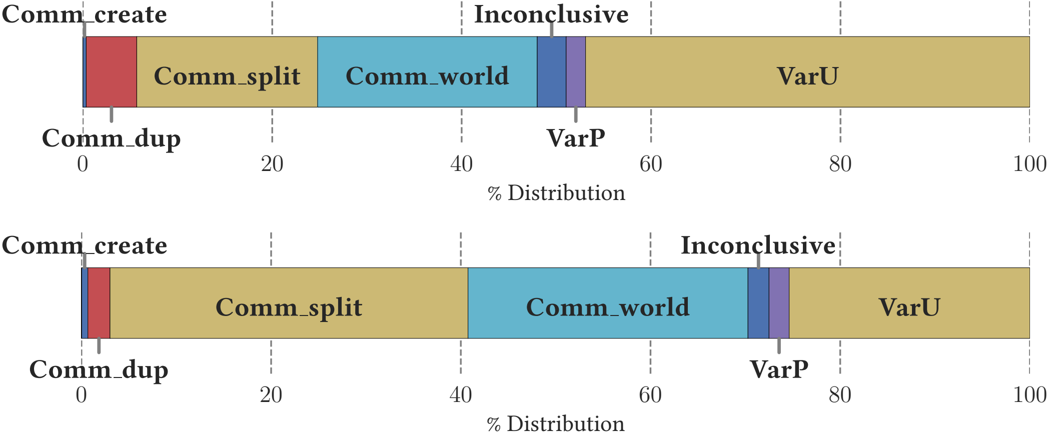
Creation of Communicators





Usage of Communicators

top: Pt2Pt bottom: collective



MPI Code Smells

Function pairs needed together		Pair	Single
`Type_Commit`	free	33	10
Comm creation	free	46	13
`Op_create`	free	17	1
Persistent Operation	free	8	1
Win creation	free	5	1
`Type_create_h...`	`MPI_Get_address`	4	2
One-sided operation	synchronization	3	0

```
MPI_Send(..., sizeof(size_t) == 4 ? MPI_INT : MPI_DOUBLE, ...);
```

Correctness Benchmark Comparison

		CorrBench				MBI			
		RC	RnC	nRC	topH (%)	RC	RnC	nRC	topH (%)
Erroneous test cases	Overall	39	192	7	3	61	172	10	10
	PtP	9	22	0	60	11	20	0	53
	One-sided	7	16	0	55	9	14	0	72
	File I/O	0	31	0	0	0	31	0	0
	Collective	7	19	0	46	22	4	1	84
	Comm Group	3	21	0	25	7	22	0	42
	Datatype	6	15	0	60	3	18	0	30
	Topology	0	16	0	0	2	14	0	25
	Persistent	0	7	0	0	3	4	0	66
	Proc. Mgmt	0	7	0	0	0	7	0	0

Correctness Benchmark Comparison

		CorrBench				MBI			
		RC	RnC	nRC	topH (%)	RC	RnC	nRC	topH (%)
Correct test cases	Overall	139	92	56	30	61	172	10	10
	PtP	27	4	0	93	11	20	0	53
	One-sided	21	2	10	100	9	14	0	72
	File I/O	0	31	0	0	0	31	0	0
	Collective	26	0	10	100	22	4	1	84
	Comm Group	15	9	1	67	7	22	0	42
	Datatype	20	1	4	90	3	18	0	30
	Topology	3	13	3	13	2	14	0	25
	Persistent	4	3	0	67	3	4	0	66
	Proc. Mgmt	0	7	0	0	0	7	0	0