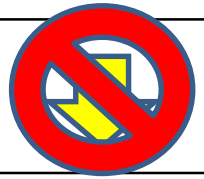
Signature Datatypes for Type Correct Collective Operations, Revisited

Jesper Larsson Träff
traff@par.tuwien.ac.at

TU Wien (Vienna University of Technology)
Faculty of Informatics, Institute of Computer Engineering
Research Group Parallel Computing

Erroneous MPI (MPI-3.1, p.33)

```
int a[10];
MPI_Send(a,10,MPI_INT,…,comm);
```



```
int b[10];
MPI_Recv(b,10*sizeof(int),MPI_BYTE,…, comm);
```

If you program like this, the rest of this talk is irrelevant

Except for the fact that

`size`(derived datatype) > `extent`(derived datatype)

for derived datatypes with overlapping entries

Correct, but lossy MPI

```
int a[10];
MPI_Send(a,10*sizeof(int),MPI_BYTE,…,comm);
```

```
int b[10];
MPI_Recv(b,10*sizeof(int),MPI_BYTE,…, comm);
```

Structured data (`int`) sent as uninterpreted Bytes. Important information on representation lost.

Correct, type safe, intended by MPI

```
int a[10];
MPI_Send(a,10,MPI_INT,…,comm);
```

```
int b[10];
MPI_Recv(b,10,MPI_INT,…, comm);
```

Structured data (`MPI_INT`) with meaning (`int`).

`MPI_INT` could be structured, heterogeneous, datatype…

The aim

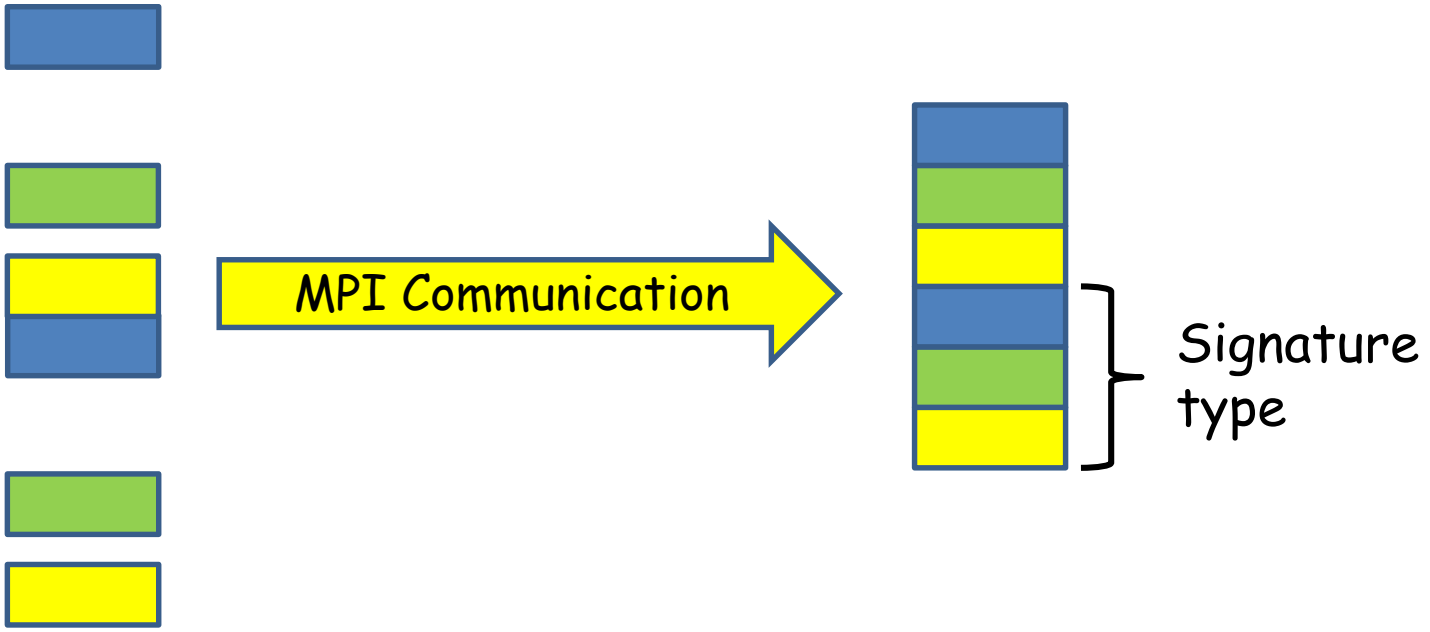
Signature types:

A means for library builders (& MPI implementers) to communicate and store intermediate data in a type safe manner; notably for functions with collective semantic requirements on matching data elements (in the MPI sense).

Issues:

- Do consistent signature types exist?
- How can signature types be computed? Efficiently?

Type safe: Communication as typed streams



(Non-)contiguous, derived datatype

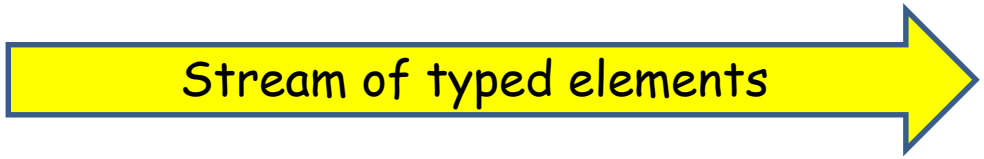
to be stored compactly in (intermediate) buffer

Signature type

Short(est possible) derived, contiguous MPI datatype describing the type signature of a stream of data

Structure of data, possibly non-contiguous

```
MPI_Send(buffer,count,datatype, …);
```



Joachim Protze, Tobias Hilbrich, Andreas Knüpfer, Bronis R. de Supinski, Matthias S. Müller: Holistic Debugging of MPI Derived Datatypes. IPDPS 2012: 354-365
Jesper Larsson Träff: A Library for Advanced Datatype Programming. EuroMPI 2016: 98-107

Consistent signature types for collective operations

Example: Broadcast can be implemented as scatter+allgather

```
MPI_Bcast(buffer,count,datatype,…) =
```


```
MPI_Scatter(buffer,count/size,datatype,…);
MPI_Allgather(buffer,count/size,datatype,…);
```

What is the right blocksize? What is the right datatype?

MPI processes may give different datatype and count arguments, as long as they all specify the same number of basic elements. How can processes agree on same signature type and block count (without extra communication)?

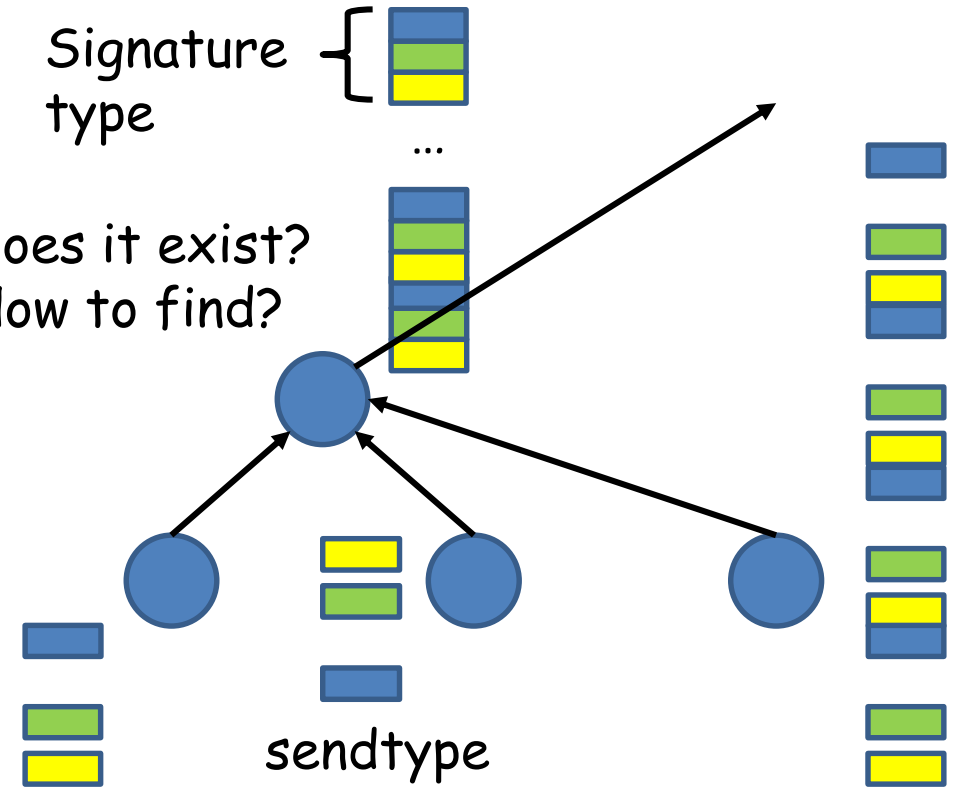
```
MPI_Bcast(12000,MPI_INT)
```

```
MPI_Bcast(6000,2*MPI_INT)  MPI_Bcast(2,6000*MPI_INT)

MPI_Bcast(400,30*MPI_INT)  MPI_Bcast(1,12000*MPI_INT)

MPI_Bcast(30,400*MPI_INT)
```

Example: Tree algorithms for gather(v)



Problems:

- Intermediate nodes do not know number of elements of child nodes
- sendtype of intermediate node may be different from sendtypes of child nodes
- sendtype could have overlapping entries

Other use cases:

Pipelined algorithms (need to select block sizes consistently, need to allocate typed intermediate buffers), message-combining algorithms (contiguous, typed intermediate buffers), etc.

Consistent signature types exist for collective semantics

Definition: A function has collective (data) semantics if the number of elements (count,datatype) match EXACTLY between pairs of processes that are related (same number of elements, same basic datatypes)

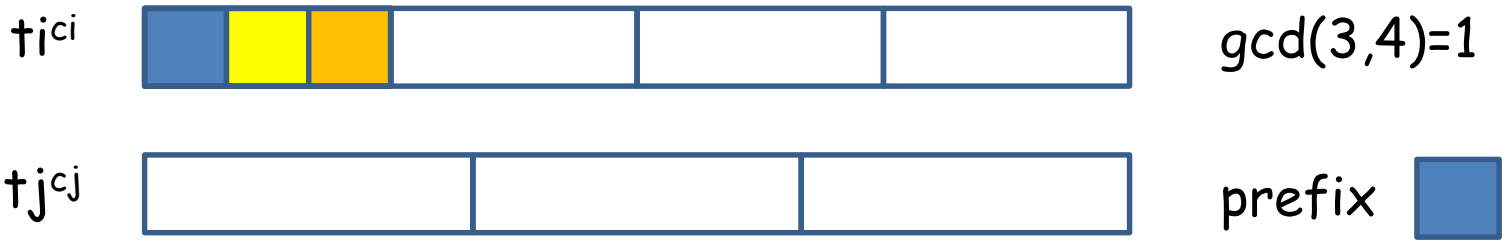
Definition: The signature of a stream of communicated data is a string of basic datatypes, e.g., MPI_INT MPI_INT MPI_DOUBLE MPI_INT MPI_INT…

Definition: The period of a string s is the shortest prefix r such that $s = r^k$

Proposition: Let (ci,ti) and (cj,tj) be counts and datatypes of two related processes in a function with collective semantics. Let ri and rj be the periods of the strings ti^{ci} and tj^{cj}. Then ri=rj

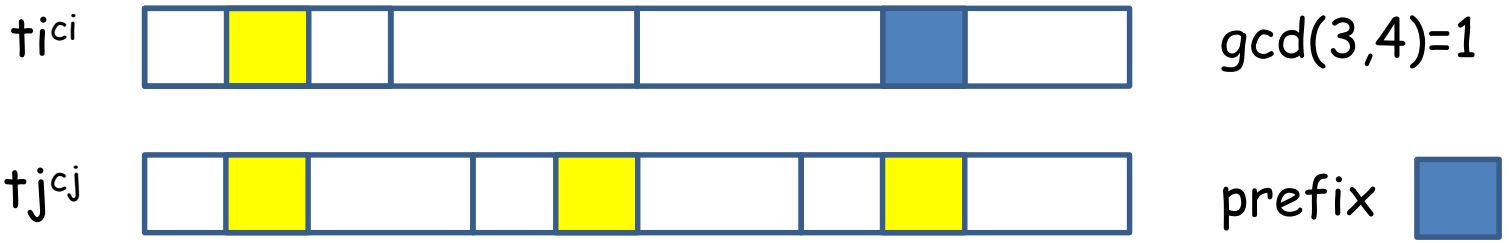
That is, the periods are unique and consistent. Computing signature types boils down to finding periods in strings of basic datatypes. Periods can be computed locally by the processes, no communication of type meta information needed, guaranteed by the collective semantics!

Proof idea/picture:
By contradiction, assume that ri and rj are different, then the prefix of length gcd(|ri|,|rj|) is a period for both strings $t_i^{c_i}$ and $t_j^{c_j}$, contradicting that either ri or rj were shortest.

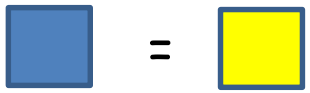


Need to prove that

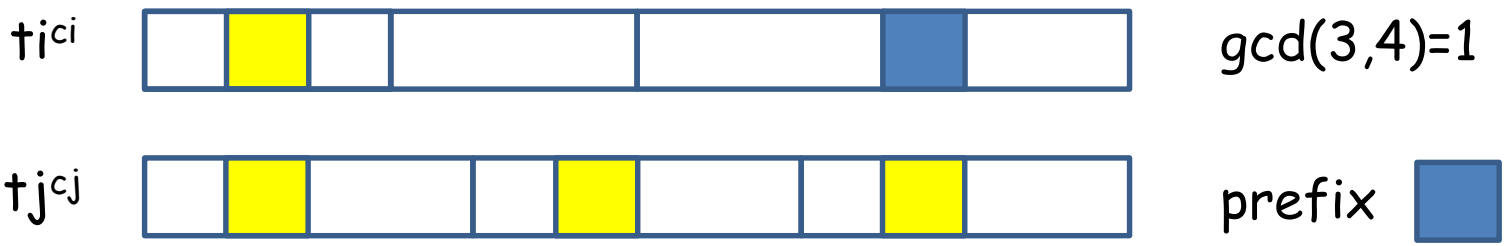
Proof idea/picture:
By contradiction, assume that ri and rj are different, then the prefix of length gcd(|ri|,|rj|) is a period for both strings $t_i^{c_i}$ and $t_j^{c_j}$, contradicting that either ri or rj were shortest.



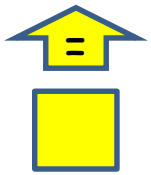
So



Proof idea/picture:
By contradiction, assume that ri and rj are different, then the prefix of length gcd(|ri|,|rj|) is a period for both strings $t_i^{c_i}$ and $t_j^{c_j}$, contradicting that either ri or rj were shortest.



In general, by elementary number theory, for prefix |u|=gcd(|ri|,|rj|) , there exist x,y such that k|u|+x|ri|=y|rj|.



Observation:

Finding the period in a string of length n can be done in O(n) time steps. The Knuth-Morris-Pratt (KMP) preprocessing step does essentially that, see standard books and papers on text processing (and the implementation in this paper).

Proposition: A signature (type) for a stream (c,t) can be computed in linear time in the stream length, $O(|t^c|)$ time steps.

Library developers: Collective (data) semantics is a strong and very useful requirement.

For rooted tree algorithms:

Corollary: Let (ci,ti), (cj,tj) and (c,t) be counts and datatypes of three processes in a function with collective semantics where i and j are both related to a root process with (c,t) but not to each other. Let ri and rj be the periods of the strings ti^{ci} and tj^{cj}. Then ri=rj

This follows by transitivity. And shows that periods computed locally by the processes can be used as consistent signature types in tree algorithms for rooted collectives (`MPI_Gatherv` example).

Negative corollary: Message-combining algorithms for `MPI_Alltoallw` cannot be implemented in a type safe manner (without communication of type meta-information)

There is no common "root" guaranteeing existence of consistent signature types for unrelated processes i and j. Process j cannot type safely store intermediate data from process i.

Persistence (MPI 4.0) can help:
Type information can be exchanged between unrelated processes at `MPI_Alltoallw_init` time.

Is this good enough?

Desire 1:
Compute the signature type directly and efficiently from any MPI derived datatype.

MPI derived datatypes constructed with the MPI constructors (vectors, index types, struct types, …) are represented by trees or DAGs of some size T (number of constructors).

Observation:
The string s represented by a derived datatype T can be arbitrarily larger than (not bounded in) T, $|s| \gg |T|$. This is due to the unbounded repetition counts in the MPI constructors. Thus $O(|s|)$ algorithms can be very slow compared to $O(T)$.

Desire 2:
We want the most compact and efficient representation of the signature type

Also the problem of computing a smallest tree T' representing a string s is difficult, e.g., $O(|s|^4)$.

Robert Ganian, Martin Kalany, Stefan Szeider, Jesper Larsson Träff: Polynomial-Time Construction of Optimal MPI Derived Datatype Trees. IPDPS 2016: 638-647

Desire 1:
Finding periods in strings in some compact representation is difficult, there seems to be no O(T) algorithm for strings represented by trees T.

Best known results (to me) are via Straight Line Programs (SLP).

(thanks to Markus Lohrey, University of Siegen, for making me aware of the power and problems of SLP)

Straight-Line Programs (SLP) for representing strings

X_0 -> a
X_1 -> b
...
X_k -> $X_i X_j$ $\quad$ $i,j < k$

An MPI derived datatype tree T with node repetition counts can be represented by an SLP with O(T log C) productions, where C is the largest repetition count in T.

Theorem: The period of a string s represented by an SLP with t productions, $t=O(T \log C)$, can be found in $O(t^2h)$ time steps, where h is the height of the derivation tree.

We have not implemented this…

Interesting, difficult problem: Can we do better for MPI derived datatypes?

Tomohiro I, Wataru Matsubara, Kouji Shimohira, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda, Kazuyuki Narisawa, Ayumi Shinohara: Detecting regularities on grammar-compressed strings. Inf. Comput. 240: 74-89 (2015)

What we do

For MPI derived datatype T describing string s:

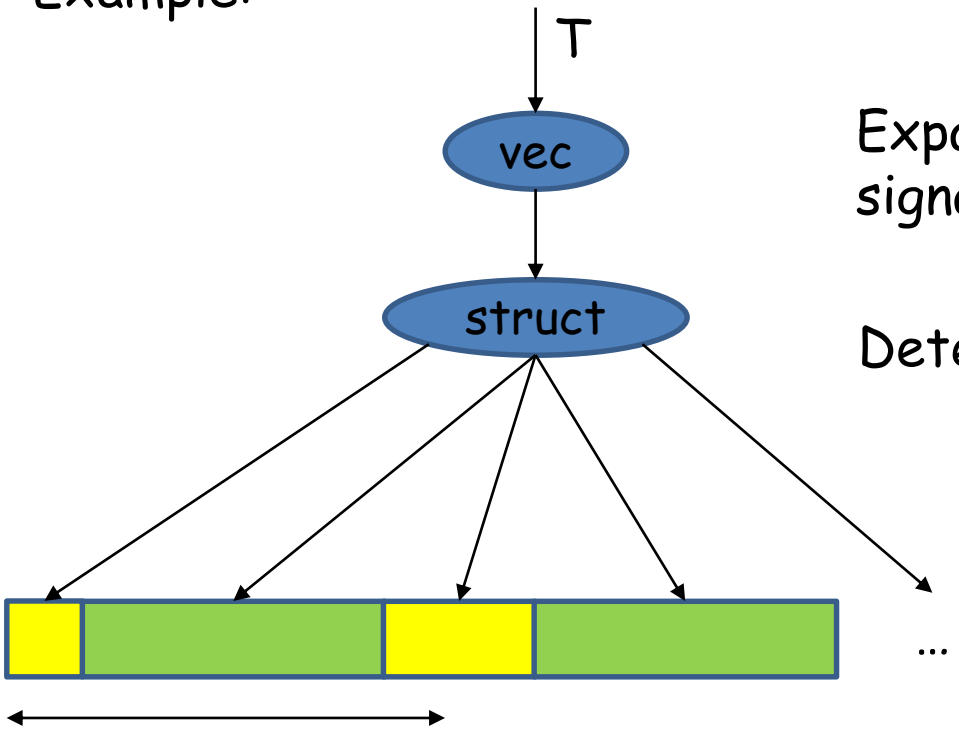
- If T is homogeneous (all elements in s have same basetype), signature type is (|s|,basetype). This can be determined by a single traversal of T in O(T) steps.
- Otherwise, expand T into s, find period r of s with r^k=s, build signature type T' from T by traversing the first |r| elements of T, signature type is (k,T').

The resulting signature type is no larger than the user-defined T.

In next release of this library:

Jesper Larsson Träff: A Library for Advanced Datatype Programming. EuroMPI 2016: 98-107

Example:



Expand T into contiguous type signature

Determine period

Length of period determined by KMP

Build signature type

Signature type is substructure of T: contig(struct(…))

The MPI standard

Native functionality for operations on type maps and derived datatypes is weak:

- Is this type homogeneous?
- What is the i'th element, where is it?
- Decoding functionality is terrible.
- …

Signatures as precomputed attributes could be helpful to library builders.

See more discussion in datatype library paper, ongoing

> Jesper Larsson Träff: A Library for Advanced Datatype Programming. EuroMPI 2016: 98-107