# Proposal for MPI-3:
# Allowing MPI macro interfaces

Rainer Keller, Jeff Squyres, Rich Graham, Hubert Ritzdorf

HLRS, Cisco, ORNL, NEC

**Abstract.** The current MPI-2 standard disallows most functionality to be implemented as macros. Whilst this allows function replacement inside binaries with tools based on the PMPI-interface, this restriction hinders MPI-implementations to further reduce overhead of MPI-calls. This paper proposes an extension to MPI-2, which would allow applications to pass hints to the MPI-implementation, which routines may be optimized and further, what kind of MPI-functionality is not used in the application.

## 1 Introduction

The MPI-2 [2] standard defines in section 4.17 a restriction on MPI implementations: Apart from the timing functions `MPI_Wtime` and `MPI_Wtick`, only the conversion functions `MPI_Group_f2c`, etc. in section 4.12.4 may be implemented as macros.

This is to guarantuee tools based on the profiling interface to hook wrappers into the call-chain based on the profiling interface (PMPI). This functionality however is mainly required for debugging and performance analysis purposes, which is routinely perfomed in the earlier stages of programming life-cycle. The general case, during application deployment, applications will not make use of this functionality. Therefore, this requirement may unnecessarily hinder the performance of applications.

By alleviating the rule, and allowing (certain) functions to be called as macros, there are a certain optimization possibilities, not yet achievable through compilers, or libraries. The most obvious reason would be the reduction of call overhead. However, there are a few further possibilities for optimization as the following will show.

## 2 Hints to allow macro-based inlining

In order to allow compilers to do a better job in optimizing code, is giving them more information. For a library-based implementation such as MPI, one may think of parameters as `const`, using `restrict` or augmenting the semantic of functions with compiler `attributes`.

Allowing inlining of one or more levels of the MPI-implementation may enable the compiler to delete code-paths, therefore reducing the amount of branches on the fast-path (e. g. knowing source process is not `MPI_ANY_SOURCE` or destination process is not `MPI_PROC_NULL`), or reducing copy operations of data-structures (e. g. `MPI_Status`).

## 3 Using specialized functionality

Furthermore, the user could specify application specific behaviour to select specialized functions:

### 3.1 All MPI functions as inline

```
#define MPI_HINT_INLINE
#include "mpi.h"
```

This allows the inlining of all MPI-functions. This will allow optimization of MPI-internal states. However, this disables the usage of PMPI-enabled tools, like performance analysers or debuggers.
Advice to Users:
In each module compiled, the user needs to specify this hint.
Advice to the implementor:
The implementor should document, which functions are inlinable.

### 3.2 No usage of multiple threads with `MPI_INIT_THREAD`

```
#define MPI_HINT_NO_THREADS
#include "mpi.h"
```

Together with `MPI_HINT_INLINE`, this hint may allow the compile time decision to use or not use functionality for threaded execution in inlined functions.

### 3.3 No usage of `MPI_ANY_SOURCE`

```
#define MPI_HINT_NO_ANY_SOURCE
#include "mpi.h"
```

This hint may allow the selection of specialized recv-functions with a streamlined code-path, which does not need to test for msg from other source processes.

## 4 Performance

The following performance results were gathered on two nodes of a dual-socket Intel Xeon WoodCrest cluster (2,6 GHz) with an Infiniband DDR Interconnect using the Netpipe-3.6.2 benchmark using the Open MPI implementation based on SVN-trunk r17537. No other command-line parameters such as `mpi_pinning` were provided.

In the following, the hints `MPI_HINT_INLINE` and `MPI_HINT_NO_THREADS` were specified, and the MPI-calls `MPI_Send`, `MPI_Recv`, `MPI_Isend`, `MPI_Irecv` and `MPI_Wait` were patched to be inlined. Additionally, Open MPI provides the means to circumenvent the modular component architecture [1]. In our case, the `ob1` Packet Management Layer (PML) was selected per default using the `configure`-flag `--enable-mca-direct=pml-ob1`. By changing the implementation to inline the underlying packet management functions (e.g. `mca_pml_ob1_irecv`), further improvements were gained.

The Latency shows the 1-Byte output of Netpipe with no additional arguments, the ∞ Bandwidth is for 8 MB message size in Megabits/second. As expected, the bandwidth does not change; the latency does not really improve, either...

| | 1-Byte Latency | | ∞ Bandwidth | | File-Size in Bytes | |
|---|---|---|---|---|---|---|
| Options | Std. | Inlined | Std. | Inlined | Std. | Inlined |
| None | 3.31 $\mu$s | 3.29 $\mu$s | 9686.58 Mbps | 9681.34 Mbps | 43455 B | 81499 B |
| Direct PML-OB1 | 3.33 $\mu$s | 3.29 $\mu$s | 9682.17 Mbps | 9678.00 Mbps | 43479 B | 132887 B |
| +No MPI Checks | 3.33 $\mu$s | 3.26 $\mu$s | 9672.47 Mbps | 9679.80 Mbps | 43503 B | 118124 B |

**Table 1.** Timing with various combinations of Configure-options

## 5 Consequences of Non-conformance

MPI-implementations not implementing this feature do **not** change in behaviour. Similarly any application compiled with a certain version of an MPI implementation using the functionality will still function with any binary compatible version of the same MPI implementation. The only restriction is the loss of PMPI functionality.

Naturally, MPI implementations will need to expose certain levels of their source base to the user. This might involve cleaning up header files, separating files by functionality or, in the case of commercial MPI implementations, obfuscating source code of header files.

## 6 Further Issues

These issues were raised in email discussions:

1. Add `MPI_Info` as argument to `MPI_Init`
   Providing this (or similar) information upon startup at runtime would be feasible in some cases. There are advantages and some disadvantages (are there more):
   + Consistent with the previous style to extend the API

+ Allow runtime changing of the app, instead of compile once and run forever
+ Keep the possibility of the PMPI-interface wrapping the app
- No inlining and further optimization by the compiler
- Need all the `MPI_Info` functionality before `MPI_Init`, therefore
- Possibly not all information available (e.g. do we have the shared library with functions that optimize `MPI_HINT_NO_ANY_SOURCE`??)
- Still the issue with other (PMPI?) libraries that *might* use `MPI_ANY_SOURCE`... (see issues 2 and 3).

Probably, both ways (compile-time hints and run-time info) could co-exist. If so, they should however provide similar "hints" or assertions (or whatever they will be called ,-])

2. The user specifies `MPI_HINT_NO_ANY_SOURCE`, but calls into libraries, that *may* use `MPI_ANY_SOURCE`.
   As far as I understand, this is not an issue when combining `MPI_HINT_NO_ANY_SOURCE` with `MPI_HINT_INLINE` at compile-time?
   As noted, the MPI-library would need to have specialized internal recv-functions (in ompi another path down the PML...):
   – In the case of `MPI_HINT_INLINE`, this would be trivial at compile-time and both the user-app and the library could co-exist.
   – Without `MPI_HINT_INLINE`, the MPI-library would need to
   ```
   #define MPI_Recv   MPI_Recv_no_any_source
   #define MPI_Sendrecv    and so on...
   ```
   which would disallow the PMPI-Interface once again ,-]
3. A hint like `MPI_HINT_NO_ANY_SOURCE` should be per-communicator.
   While I agree with You, that this could be feasible, I believe that this is the sort of differentation that makes the fast-path handling so difficult in the first place ,-] So, one would need per-communicator P2P recv-functions (and send?) functions.
   + The MPI-lib would have all the information for `MPI_Info` at run-time.
   - Additional communicator creation routines with `MPI_Info` (too cumbersome??)
   - The user should be aware, that he may *not* use a `MPI_HINT_ANY_SOURCE` on a communicator and *then* pass the communicator to libraries (as they may use `MPI_ANY_SOURCE`). So, this would be an advice to users...
4. Adding MPI-semantic awareness into compilers.

5. Code size increase of `MPI_HINT_INLINE`?
   As of now, the code size by `MPI_HINT_INLINE` will be a major concern. I have to investigate, how I may decrease the size of the include-files...
6. Noise in the measurment, Understanding the data.
   The measurement was done on an (empty?) cluster of Intel Xeon Woodcrest over IB-DDR HCAs from Mellanox (25204) without HCA-memory... The nodes are not shared. but there still may be other users on the cluster influencing the measurements... The timings were done with netpipe-3.6.2.

They are not stable, depending on which nodes I receive (on the same leafe-node??), so they vary +0.04usecs when I am on different nodes. When on the same two compute nodes, the measurements are "stable" (+/-0.02 usecs) but at least constistent in that the inline stuff seems to be a bit faster... Nevertheless, this is a weak point, still!
7. Moving to wider group (like MPI3-subsetting).

## References

1. Edgar Gabriel and et al. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In D. Kranzlmüller, P. Kacsuk, and J.J. Dongarra, editors, *Proceedings of the* 11$^{th}$ *European PVM/MPI Users' Group Meeting*, volume 3241 of *Lecture Notes in Computer Science (LNCS)*, pages 97–104, Budapest, Hungary, September 2004. Springer.
2. Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, July 1997. `http://www.mpi-forum.org`.