Open Research Objects (ORO) Research Objects Reviewed (ROR)

EMPI: Enhanced Message Passing Interface in Modern C++

Majid Salimi Beni, Luigi Crisci and Biagio Cosenza

Department of Computer Science
University of Salerno, Salerno, Italy
msalimibeni@unisa.it

The 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid 2023)
Bangalore, India

Outline

- MPI and Modern C++
- EMPI (Message Passing Interface)
- EMPI Semantics:
  - Program Context
  - Message Group
  - Implicit / Explicit wait
- EMPI's Runtime Check Reduction
  - Customized OpenMPI
  - Constant Specialization
- EMPI vs MPI: a Showcase
- Performance Evaluation
  - Microbenchmarks
  - Applications
- Conclusion and Future Work

Message Passing Interface vs Modern C++

- MPI
  - Poor programmability
    - Old-fashioned C-based
    - Doesn't use modern language paradigms
  - Error-prone interface
    - Too many parameters

```
MPI_Isend( void *buf, int count, MPI_Datatype datatype, int
      dest, int tag, MPI_Comm comm, MPI_Request *request );
```

    - Lacking a matching wait for asynchronous calls
    - Unmatched wait
    - Data type mismatches

```
Rank 0                                  Rank 0
MPI_Send(to:1, type=MPI_INT)            MPI_Recv(from:0, type=MPI_FLOAT)
```

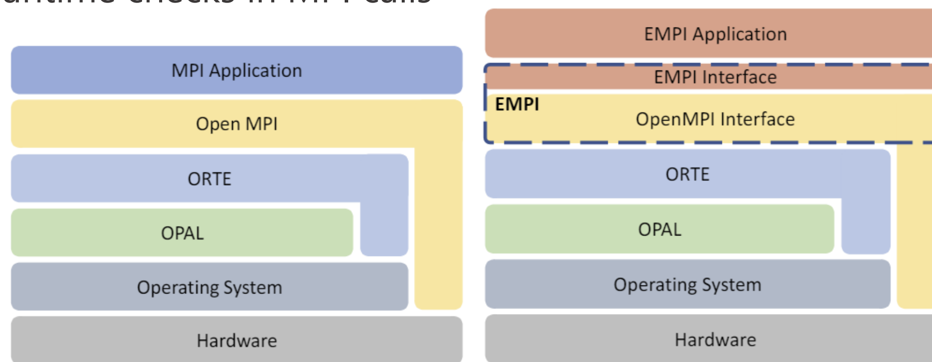
    - No Init/Finalize

- Modern C++
  - High productivity
  - Memory management
  - High performance
  - Many features
    - Template Metaprogramming
    - (Standard) library features
    - RAII, SFINAE, CTAD, Concepts, Lambda functions, Constraints, etc.

MPI's interface is holding it back!

Enhanced Message Passing Interface (EMPI)

- ❑ Exploits modern C++ features
- ❑ Mitigates programming errors
- ❑ Delivers competitive performance
- ❑ Unlike the state-of-the-art (e.g. MPL[1]), it's not just a C++ wrapper for MPI
- ❑ Directly coupled with a customized OpenMPI interface
  - ❑ Can directly interact with layers underneath
  - ❑ Skips some runtime checks in MPI calls



OpenMPI abstraction layer architecture   EMPI abstraction layer architecture

[1] Sayan Ghosh, Clara Alsobrooks, Martin Rufenacht, Anthony Skjellum, Purushotham V Bangalore, and Andrew Lumsdaine. "Towards modern C++ language support for MPI". In: 2021 Workshop on Exascale MPI (ExaMPI). IEEE. 2021, pp. 27–35.

EMPI Semantics: Program Context, Message Group

- ❑ Program Context

```
using namespace empi;
Context ctx(&argc, &argv);
```

  - ❑ Replaces `MPI_Init()` and `MPI_Finalize()`
  - ❑ Uses C++ RAII

✓ Forgetting to put MPI Init and MPI Finalize

✓ Minimizes the risk of leaking resources

- ❑ Message Group

```
message_group = ctx.create_message_group(comm);
message_group->run(
  [&](MessageGroupHandler <datatype,tag,size> &mgh){
   // Do Work and Communication
});
```

  - ❑ Communications with the same communicator
  - ❑ Enables **constant specialization**
    - ❑ Contains communications that have some parameters in common

✓ Reduces parameters passed to each call

✓ Type mismatch

✓ Invalid argument types

EMPI Semantics: Implicit and Explicit wait

- ❑ Implicit Wait

```
message_group->run_and_wait([&](MessageGroupHandler<char,
     notag, N> &mgh) {
  mgh.Ibcast(message, 0);
  // Do Some Work
}); //implicit wait here
```

- ✓ Ensures not having missing wait

  - ❑ A `wait_all()` is called automatically at the end of the lambda

- ❑ Explicit Wait and Automatic Request Handling

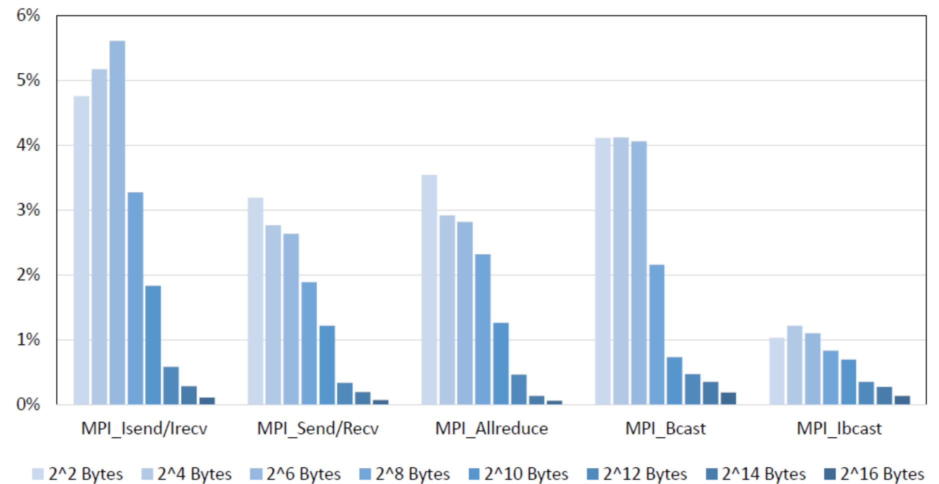
```
message_group->run(
  [&](MessageGroupHandler<char, Tag{0}, n> &mgh) {
  for (auto iter=0; iter<max_iter; iter++){
    mgh.Irecv(rbuff, prev);
    mgh.Irecv(rbuff, next);
    mgh.Isend(Sbuff, prev);
    mgh.Isend(Sbuff, next);
    mgh.waitall(); // Explicit wait
  }
});
```

- ✓ Prevents double request usage
- ✓ Minimizes the overhead of creating and deleting multiple requests

  - ❑ Automatically handles the request objects in a request pool

EMPI Runtime Check Reduction

- MPI communication primitives:
  - Checks + Communication
- These checks are to control:
  - If data type, message size, and communicator are defined
  - Required buffers are accessible
  - If parameters are valid values
- Some time is spent while doing checks!
  - More considerable for small messages
  - Affects applications dealing with many small messages (e.g., Stencil)
- Can we reduce function call latency?
  - Performing them statically



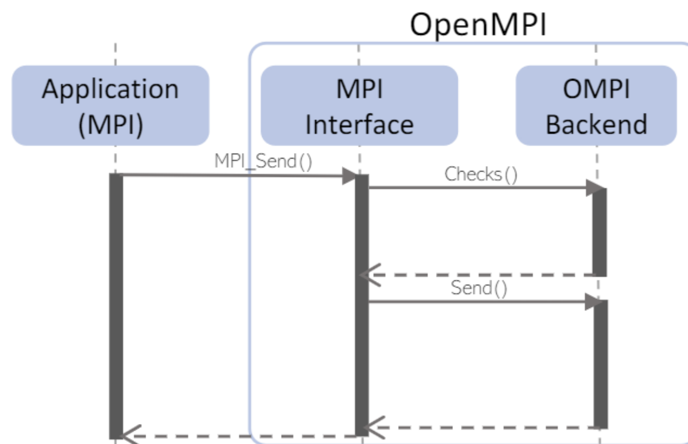
Percentages of checks to the overall time taken by different OMPI function invocations.

Message Group Constant Specialization
+
Customized OMPI

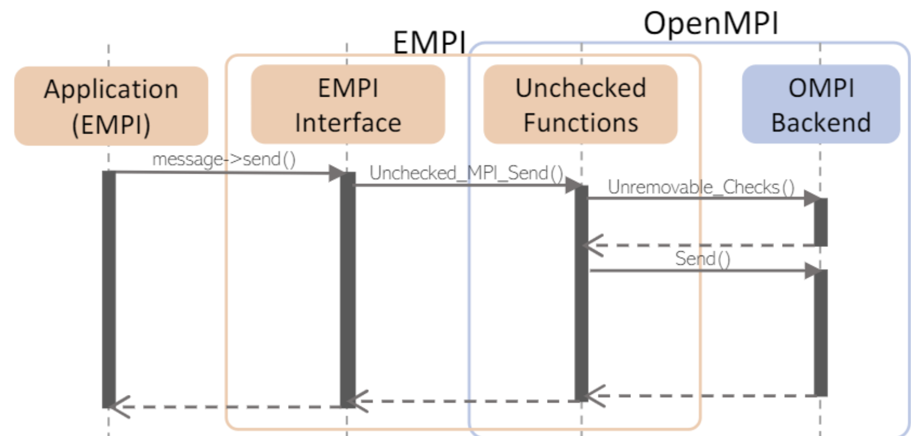
Customized OpenMPI for Check reduction

- ❑ Each communication primitive has a corresponding unchecked version
  - ❑ Delivers the same functionality as the OpenMPI function
  - ❑ Skips some of the runtime checks

```
MPI_Send() --> Unchecked_MPI_Send()
```



(a) MPI interface over OpenMPI call sequences.



(b) EMPI with unchecked semantic call sequences.

Customized OpenMPI for Check reduction

```
int MPI_IUsend(const void *buf, int count, MPI_Datatype type, int dest,
               int tag, MPI_Comm comm, MPI_Request *request)
{
    int rc = MPI_SUCCESS;

    SPC_RECORD(OMPI_SPC_ISEND, 1);

    MEMCHECKER(
        // memchecker_datatype(type);
        memchecker_call(&opal_memchecker_base_isdefined, buf, count, type);
        // memchecker_comm(comm);
    );

    if ( MPI_PARAM_CHECK ) {
        // OMPI_ERR_INIT_FINALIZE(FUNC_NAME);
        if (ompi_comm_invalid(comm)) {
            return OMPI_ERRHANDLER_INVOKE(MPI_COMM_WORLD, MPI_ERR_COMM, FUNC_NAME);
        // } else if (count < 0) {
        //     rc = MPI_ERR_COUNT;
        // } else if (MPI_DATATYPE_NULL == type || NULL == type) {
        //     rc = MPI_ERR_TYPE;
        // } else if (tag < 0 || tag > mca_pml.pml_max_tag) {
        //     rc = MPI_ERR_TAG;
        } else if (ompi_comm_peer_invalid(comm, dest) &&
                   (MPI_PROC_NULL != dest)) {
            rc = MPI_ERR_RANK;
        ⋮
```

Message Group Constant Specialization

- In MPI, A group of communications often use the same parameters
  - Data type, size, tag, etc.
  - We put them within one message group
- Constant Specialization
  - Constants whose values can be set dynamically during the execution of the program
- Unchecked primitives (Customized OMPI)
  - We skip some of the checks in each message group
    - For the constant parameters
  - We perform them in the constructor of each message group **only once**

EMPI example:

```
Context ctx(&argc, &argv);
ctx.create_message_group(comm)->run(
  [&](MessageGroupHandler<char,Tag{0},n> &mgh) {
  if(mgh.rank() == 0){
    mgh.send(message,1);
    mgh.recv(message,1,status);
  }else{
    mgh.recv(message,0,status);
    mgh.send(message,0);
  }
  mgh.barrier();
});
```

Send/Recv are mapped to Unchecked functions

EMPI vs MPI: a Showcase

MPI ping/pong example:

```
MPI_Init(&argc, &argv);
MPI_Comm_size(comm, &procs);
MPI_Comm_rank(comm, &myid);
if (myid == 0) {
  MPI_Send(arr, n, MPI_CHAR, 1, 0, comm);
  MPI_Recv(arr, n, MPI_CHAR, 1, tag, comm, &status);
} else { // Node rank 1
  MPI_Recv(myarr, n, MPI_CHAR, 0, tag, comm, &status);
  MPI_Send(myarr, n, MPI_CHAR, 0, 1, comm);
}
MPI_Barrier(comm);
MPI_Finalize();
```

EMPI ping/pong example:

```
Context ctx(&argc, &argv);
ctx.create_message_group(comm)->run(
  [&](MessageGroupHandler<char,Tag{0},n> &mgh) {
  if(mgh.rank() == 0){
    mgh.send(message,1);
    mgh.recv(message,1,status);
  }else{
    mgh.recv(message,0,status);
    mgh.send(message,0);
  }
  mgh.barrier();
});
```

Communicator is the same for all the communications within this Message Group

Constant for the message group communications

We have fewer parameters now

Performance Evaluation - Microbenchmarks

- ❑ EMPI shows very competitive performance with vanilla OpenMPI
- ❑ EMPI shows higher performance than MPL[1] (the state of the art)

Bcast (2^2 Bytes)

OMPI EMPI MPL

Time (ms)

N. Processes

Bcast (2^8 Bytes)

OMPI EMPI MPL

Time (ms)

N. Processes

Bcast (2^{16} Bytes)

OMPI EMPI MPL

Time (ms)

N. Processes

Ibcast (2^2 Bytes)

OMPI EMPI MPL

Time (ms)

N. Processes

Ibcast (2^8 Bytes)

OMPI EMPI MPL

Time (ms)

N. Processes

Ibcast (2^{16} Bytes)

OMPI EMPI MPL

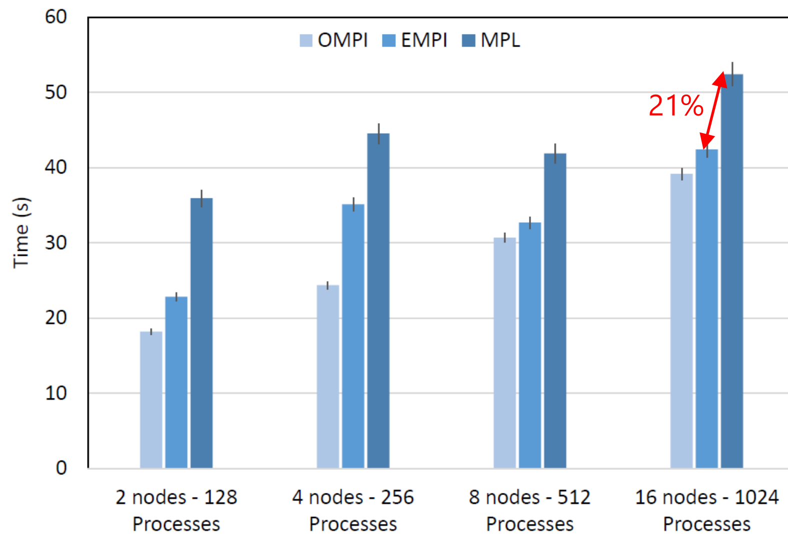
Time (ms)

N. Processes

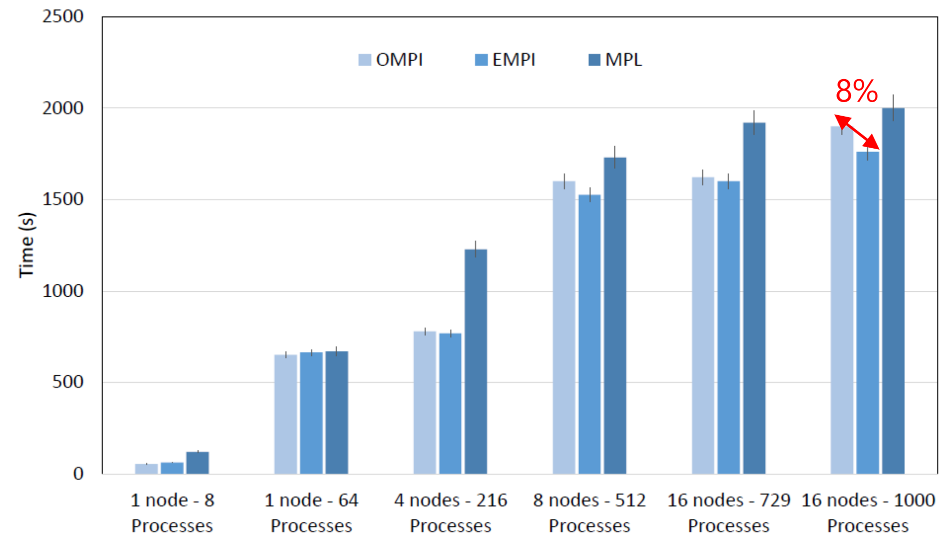
[1] Sayan Ghosh, Clara Alsobrooks, Martin Rufenacht, Anthony Skjellum, Purushotham V Bangalore, and Andrew Lumsdaine. "Towards modern C++ language support for MPI". In: 2021 Workshop on Exascale MPI (ExaMPI). IEEE. 2021, pp. 27–35.

Performance Evaluation - Applications

- Vibrating String
  - Strong scaling on up to 1024 processes
  - Competitive performance with OpenMPI
  - Better performance than the State of the art
- LULESH
  - Weak scaling on up to 1000 processes
  - EMPI is performing even better than OpenMPI
  - Iteratively sends small messages
    - The same size, type, and communicator



Vibrating String mini-app



LULESH

Conclusion and Future Work

- EMPI
  - Improves programmability thanks to C++ features
  - Less error-prone codes
  - Decreases the code's complexity
  - Competitive performance with OpenMPI
  - Higher performance than state of the art
- EMPI has passed the Artifact Evaluation
  - Artifact: https://doi.org/10.5281/zenodo.7727977
  - Ongoing project: https://github.com/unisa-hpc/empi
- Future Directions
  - Providing support for more MPI features
    - Handling complex data types
  - Exploiting latest C++ features
  - GPU support

EMPI: Enhanced Message Passing Interface in Modern C++

Majid Salimi Beni, Luigi Crisci, Biagio Cosenza

The 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid 2023)

Bangalore, India
May 1-4, 2022

Reach me at:
msalimibeni@unisa.it