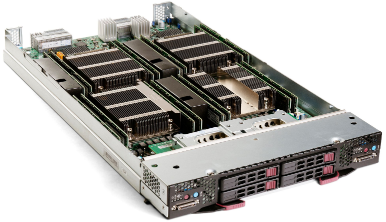
An MPI Dialect for MLIR

Tobias Grosser, University of Cambridge
Anton Lydike, University of Edinburgh

In cooperation with:
Jeff Hammond, NVIDIA, Chair of the MPI ABI Working Group

MLIR

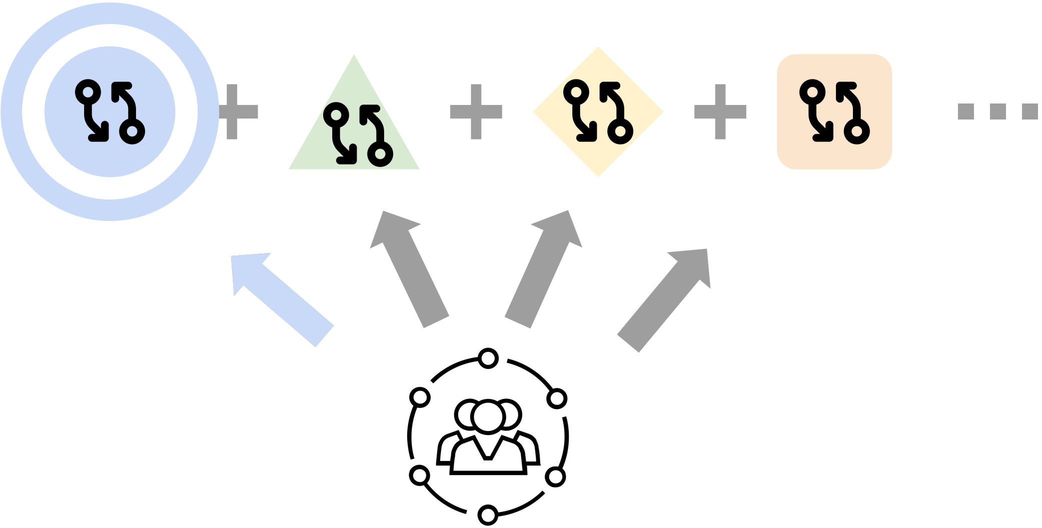


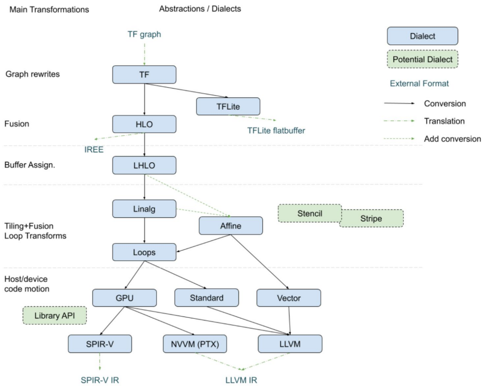
✓

MLIR

?

Scope and Goals





Scope and Goals

```c
char message[20];
int myrank;
```

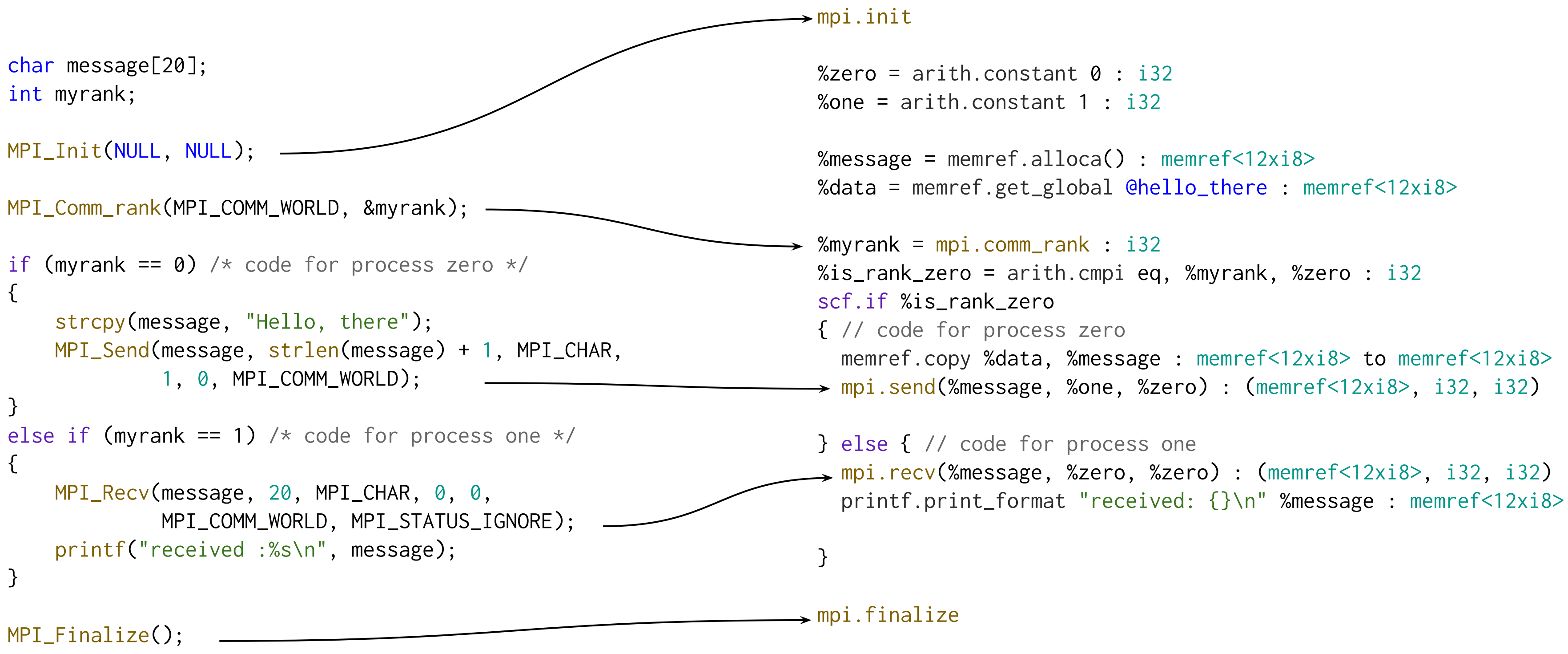
```
myrank = 0

      message =  "Hello, there"

      MPI_Send message to rank 1
```

```
myrank = 1

      MPI_Recv into message from rank 0

      message =  "Hello, there"
```

Scope and Goals

```
char message[20];
int myrank;

MPI_Init(NULL, NULL);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

if (myrank == 0) /* code for process zero */
{
    strcpy(message, "Hello, there");
    MPI_Send(message, strlen(message) + 1, MPI_CHAR,
             1, 0, MPI_COMM_WORLD);
}
else if (myrank == 1) /* code for process one */
{
    MPI_Recv(message, 20, MPI_CHAR, 0, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("received :%s\n", message);
}

MPI_Finalize();
```

Scope and Goals



```c
char message[20];
int myrank;

MPI_Init(NULL, NULL);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

if (myrank == 0) /* code for process zero */
{
    strcpy(message, "Hello, there");
    MPI_Send(message, strlen(message) + 1, MPI_CHAR,
             1, 0, MPI_COMM_WORLD);
}
else if (myrank == 1) /* code for process one */
{
    MPI_Recv(message, 20, MPI_CHAR, 0, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("received :%s\n", message);
}

MPI_Finalize();
```

```
mpi.init

%zero = arith.constant 0 : i32
%one = arith.constant 1 : i32

%message = memref.alloca() : memref<12xi8>
%data = memref.get_global @hello_there : memref<12xi8>

%myrank = mpi.comm_rank : i32
%is_rank_zero = arith.cmpi eq, %myrank, %zero : i32
scf.if %is_rank_zero
{ // code for process zero
  memref.copy %data, %message : memref<12xi8> to memref<12xi8>
  mpi.send(%message, %one, %zero) : (memref<12xi8>, i32, i32)

} else { // code for process one
  mpi.recv(%message, %zero, %zero) : (memref<12xi8>, i32, i32)
  printf.print_format "received: {}\n" %message : memref<12xi8>

}

mpi.finalize
```

```
MPI_Init
MPI_Comm_rank
MPI_Send
MPI_Recv
MPI_Finalize
```

Modelling MPI

Modelling MPI

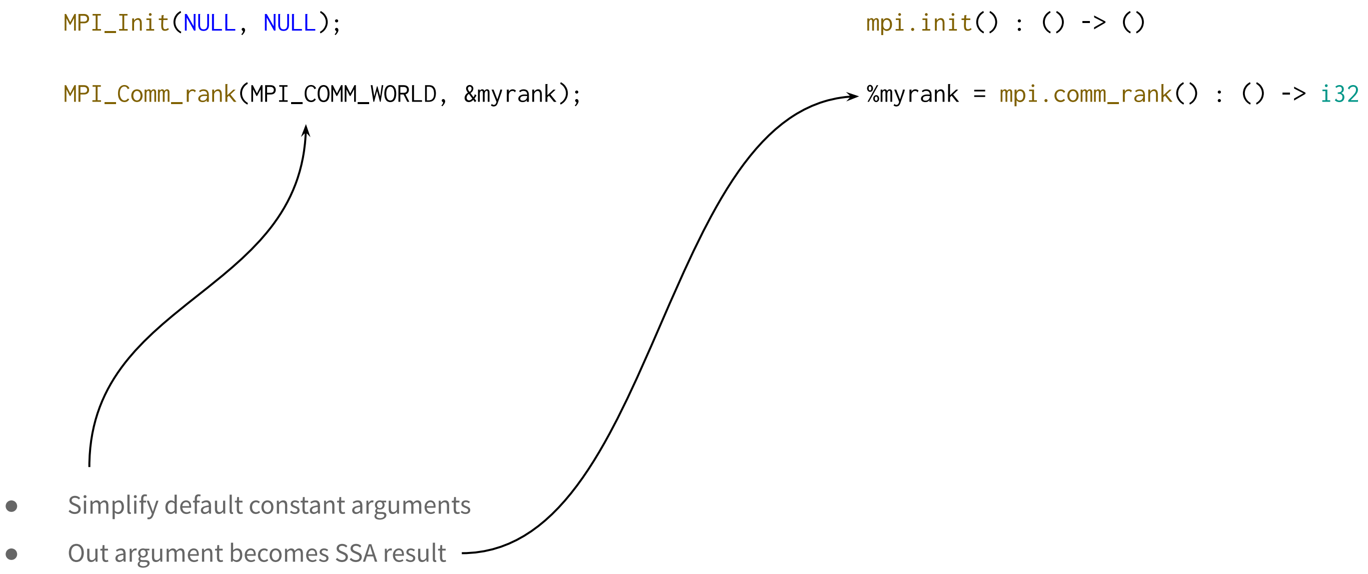
```
MPI_Init(NULL, NULL);
```

```
mpi.init() : () -> ()
```

- Simplify default constant arguments

Modelling MPI

MPI_Init ✓
MPI_Comm_rank ✓
MPI_Send
MPI_Recv
MPI_Finalize



```
MPI_Init(NULL, NULL);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

```
mpi.init() : () -> ()

%myrank = mpi.comm_rank() : () -> i32
```

- Simplify default constant arguments
- Out argument becomes SSA result

MPI_Init ✓

MPI_Comm_rank ✓

MPI_Send ✓

MPI_Recv

MPI_Finalize

Modelling MPI

```
MPI_Init(NULL, NULL);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

MPI_Send(message, strlen(message) + 1, MPI_CHAR,
         1, 0, MPI_COMM_WORLD);
```

```
mpi.init() : () -> ()

%myrank = mpi.comm_rank() : () -> i32

mpi.send(%message, %one, %zero)
  : (memref<12xi8>, i32, i32) -> ()
```

- Simplify default constant arguments
- Out argument becomes SSA result
- Pointer + Size + Datatype = memref

MPI_Init ✓
MPI_Comm_rank ✓
MPI_Send ✓
MPI_Recv ✓
MPI_Finalize

Modelling MPI

```
MPI_Init(NULL, NULL);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

MPI_Send(message, strlen(message) + 1, MPI_CHAR,
         1, 0, MPI_COMM_WORLD);

MPI_Recv(message, 20, MPI_CHAR, 0, 0,
         MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```
mpi.init() : () -> ()

%myrank = mpi.comm_rank() : () -> i32

mpi.send(%message, %one, %zero)
  : (memref<12xi8>, i32, i32) -> ()

mpi.recv(%message, %zero, %zero)
  : (memref<12xi8>, i32, i32) -> ()
```

- Simplify default constant arguments
- Out argument becomes SSA result
- Pointer + Size + Datatype = memref

MPI_Init ✓
MPI_Comm_rank ✓
MPI_Send ✓
MPI_Recv ✓
MPI_Finalize ✓

Modelling MPI

```
MPI_Init(NULL, NULL);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

MPI_Send(message, strlen(message) + 1, MPI_CHAR,
         1, 0, MPI_COMM_WORLD);

MPI_Recv(message, 20, MPI_CHAR, 0, 0,
         MPI_COMM_WORLD, MPI_STATUS_IGNORE);

MPI_Finalize();
```

```
mpi.init() : () -> ()

%myrank = mpi.comm_rank() : () -> i32

mpi.send(%message, %one, %zero)
  : (memref<12xi8>, i32, i32) -> ()

mpi.recv(%message, %zero, %zero)
  : (memref<12xi8>, i32, i32) -> ()

mpi.finalize() : () -> ()
```

- Simplify default constant arguments
- Out argument becomes SSA result
- Pointer + Size + Datatype = memref

What’s Next

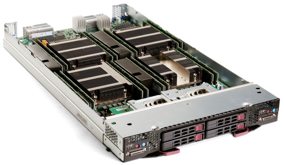
- Use this as a starting off point for
  - Nonblocking sends
  - Return value handling
  - Custom communicators
  - Collectives
  - …
- Start working on a lowering
  - Support for strided memrefs
  - Error handling
  - …

Conclusions

- We present a minimal first draft of an MPI dialect design
- We plan to continue development in small incremental PRs
- Our long-term goal is to build a stack of dialects for distributed computing

Conclusions

```
mpi.send(%mlir, %future, %tag)
```



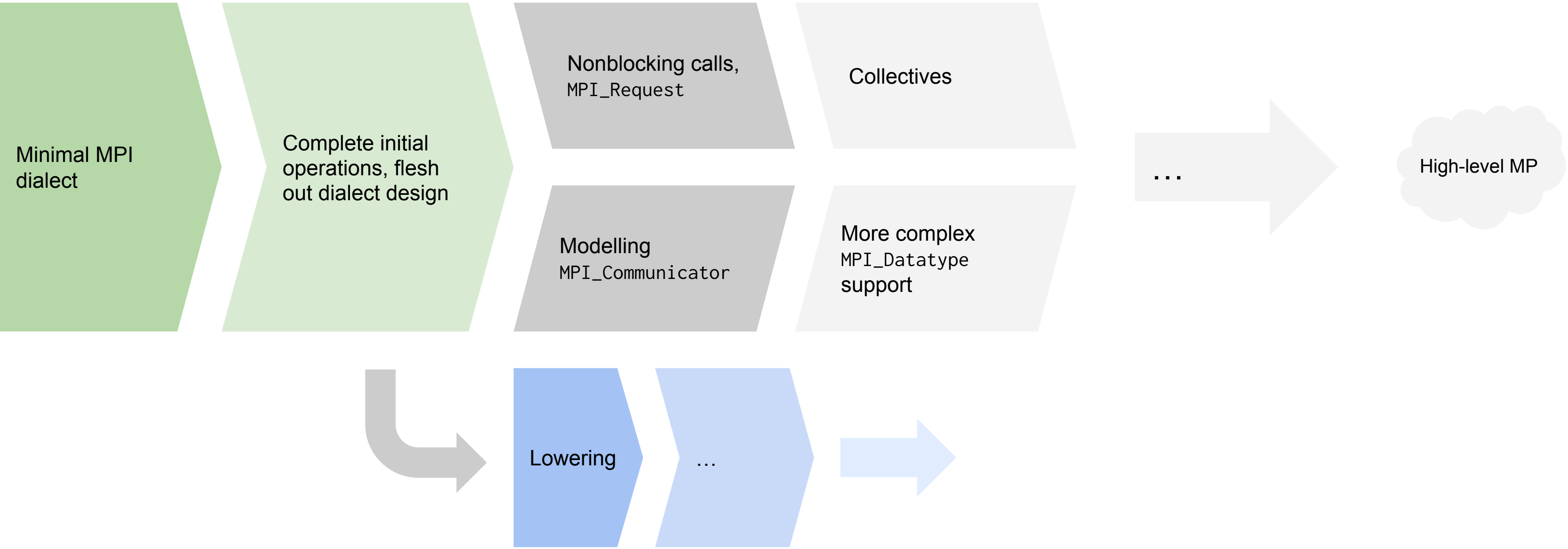
```
mpi.recv(%mlir, %past, %tag)
```

Status

Thing	**Status**
Init, Finalize, Comm_rank, Send, Receive	PR Ready
Lowering to MPICH	Draft
Lowering to OpenMPI	Draft

Spare Slides

Roadmap


Minimal MPI dialect
Complete initial operations, flesh out dialect design
Nonblocking calls, MPI_Request
Modelling MPI_Communicator
Collectives
More complex MPI_Datatype support
…
High-level MP
Lowering
…

Strided memrefs

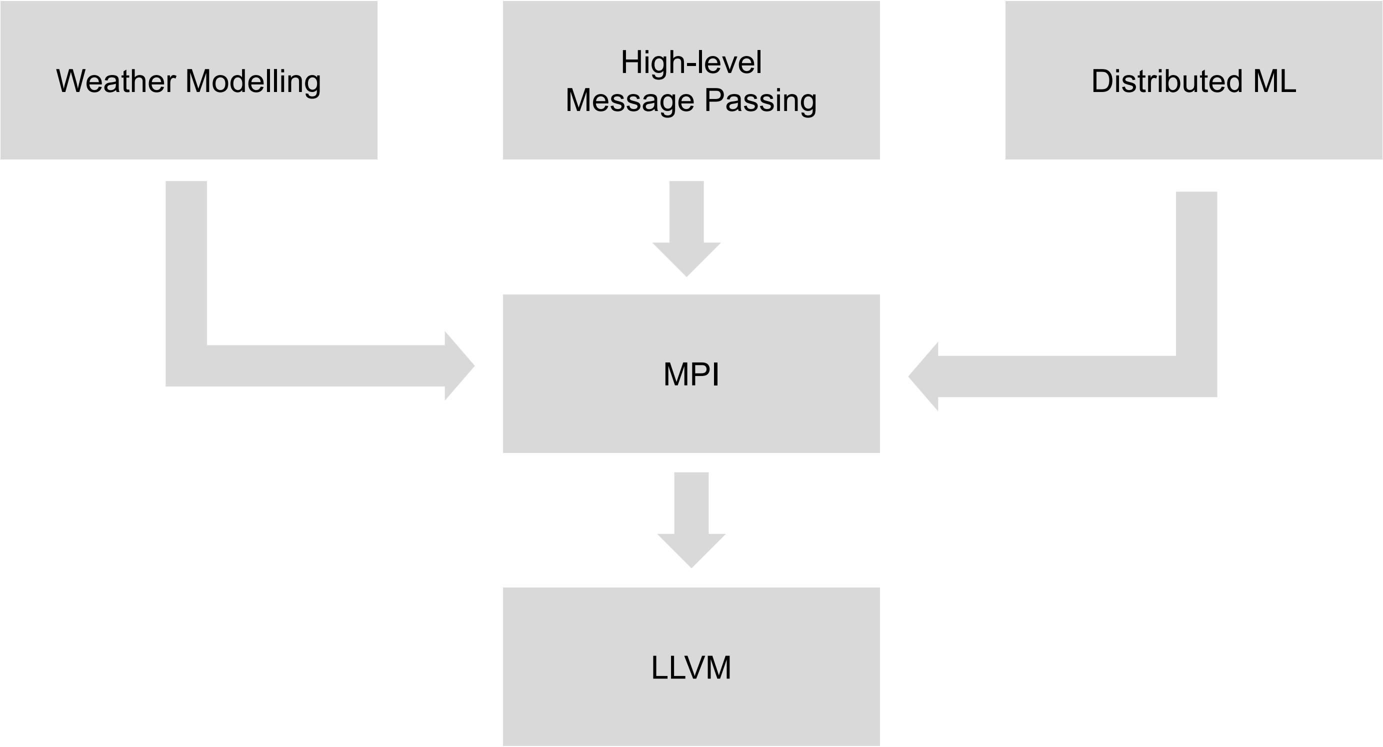
```
func.func @test(%ref: memref<12x12x12xf32>) {
  %view = memref.subview %ref[2,10,0][8,8,4][1,1,1]
  : memref<12x12x12xf32> to
  memref<8x8x4xf32, strided<[144, 12, 1], offset: 408>

  %cst0 = arith.constant 0 : i32

  mpi.send(%view, %cst0, %cst0)
}
```

```
void test(struct memref_f32_rank_2 ref) {
    MPI_Datatype strided_vec;

    // create MPI_Datatype for strided memref
    MPI_Type_vector(
        /*count =*/       8*8,
        /*blocklength =*/ 4,
        /*stride =*/      144,
        /*oldtype =*/     MPI_FLOAT,
        /*newtype =*/     &strided_vec
     );

    // get offset pointer (base + 408 * sizeof(float))
    void * offset_ptr = (void *) (
        ((float *) ref.aligned) + ref.offset
    );

    MPI_Send(
        offset_ptr, 1, strided_vec, 0, 0, MPI_COMM_WORLD
    );
}
```

MPI ABI

- Two big targets: MPICH (Intel-style) and OpenMPI
  - **MPICH**: Handles are of type `int` and have compile-time known values for constants
  - **OpenMPI**: Handles are external opaque struct pointers
- Stable ABI:
  - Handles are opaque struct pointers with compile time known values for constants
- We have Prototypes showing we can both lower our design to MPICH and OpenMPI

MPI Dialect Role



Weather Modelling

High-level
Message Passing

Distributed ML

MPI

LLVM