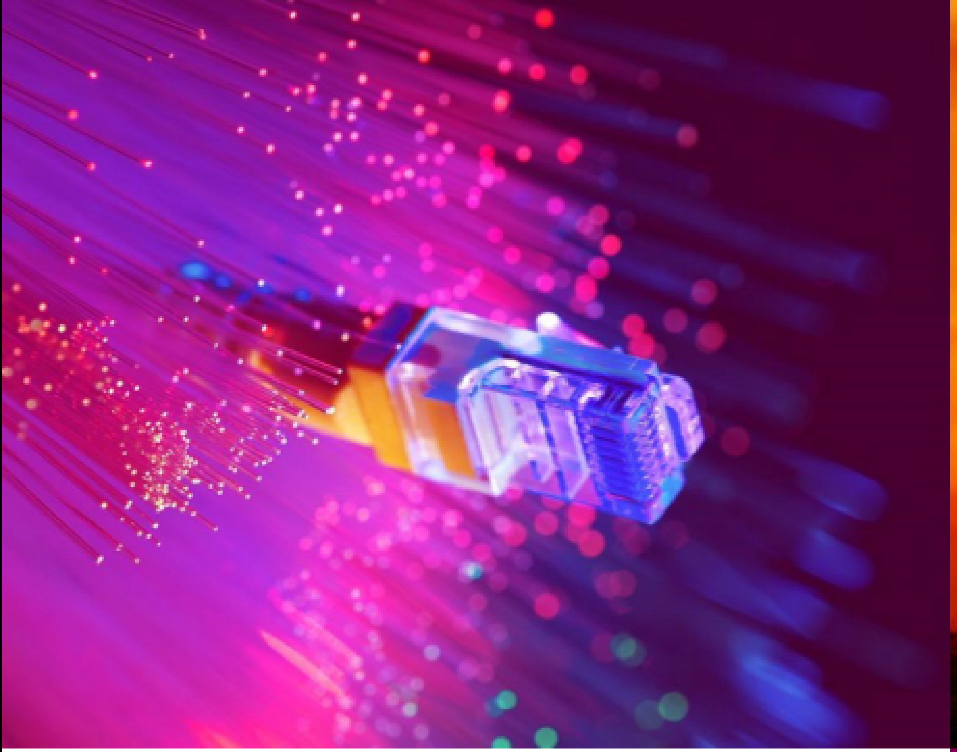

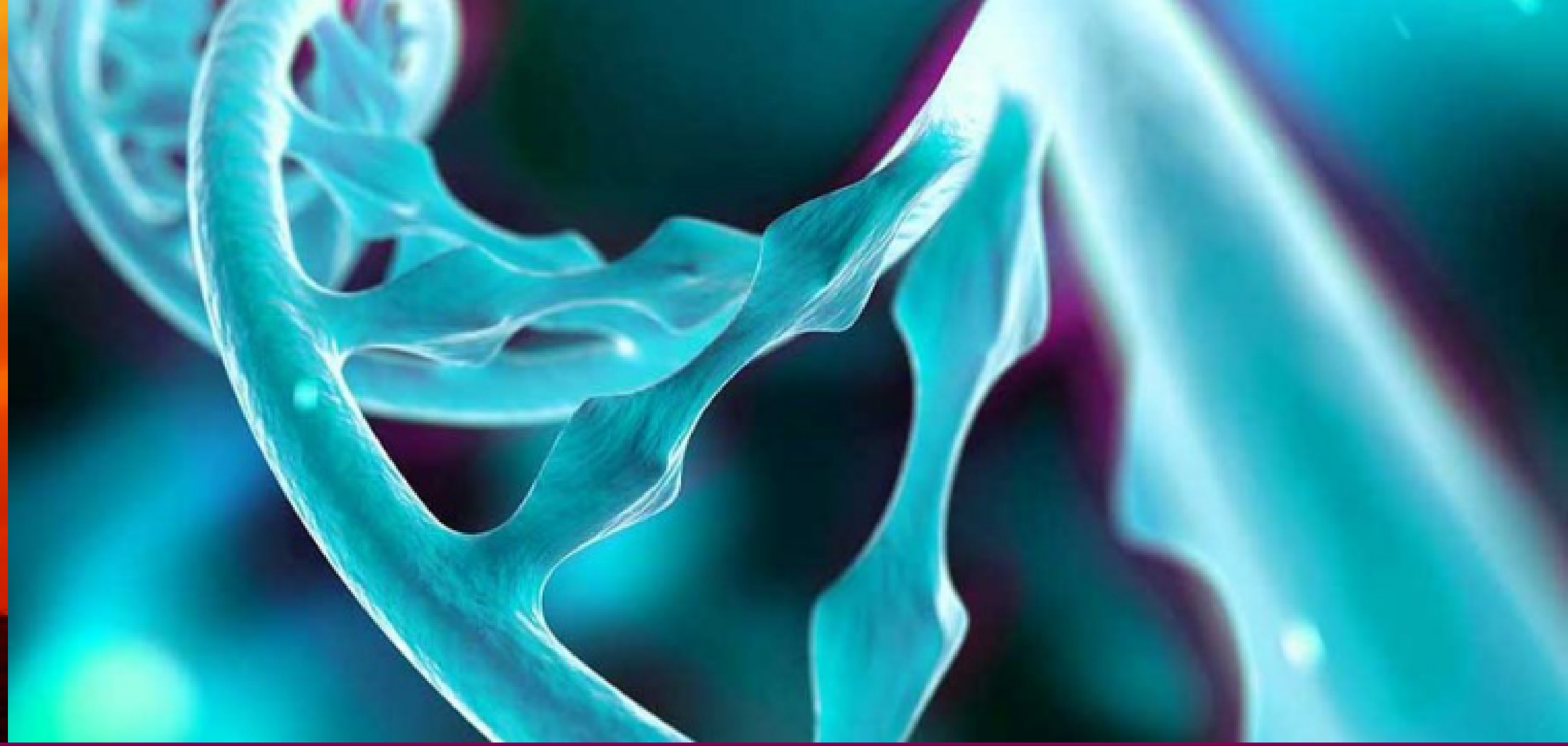

MPI reference

CSC Summer School in High-Performance Computing 2022

CSC – Finnish expertise in ICT for research, education and public administration

C interfaces

C interfaces for the “first six” MPI operations

```
int MPI_Init(int *argc, char **argv)

int MPI_Init_thread(int *argc, char **argv, int required, int *provided)

int MPI_Comm_size(MPI_Comm comm, int *size)

int MPI_Comm_rank(MPI_Comm comm, int *rank)

int MPI_Barrier(MPI_Comm comm)

int MPI_Finalize()
```

C interfaces for the basic point-to-point operations

```
int MPI_Send(void *buffer, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm, MPI_Status *status)

int MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int
                 void *recvbuf, int recvcount, MPI_Datatype recvtype, int source, in
                 MPI_Comm comm, MPI_Status *status)

int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)
```

MPI datatypes for C

MPI type	C type
MPI_CHAR	signed char
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_BYTE	

C interfaces for collective operations

```
int MPI_Bcast(void* buffer, int count, MPI_datatype datatype, int root, MPI_Comm com

int MPI_Scatter(void* sendbuf, int sendcount, MPI_datatype sendtype,
                void* recvbuf, int recvcount, MPI_datatype recvtype, int root, MPI_C

int MPI_Scatterv(void* sendbuf, int *sendcounts, int *displs, MPI_datatype sendtype,
                 void* recvbuf, int recvcount, MPI_datatype recvtype, int root, MPI_

int MPI_Gather(void* sendbuf, int sendcount, MPI_datatype sendtype,
               void* recvbuf, int recvcount, MPI_datatype recvtype, int root, MPI_Co

int MPI_Gatherv(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf, int *recvcnts, int *displs, MPI_Datatype recvtype,
                int root, MPI_Comm comm)
```

C interfaces for collective operations

```
int MPI_Reduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype,
               MPI_Op op, int root, MPI_Comm comm)

int MPI_Allreduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype,
                  MPI_Op op, MPI_Comm comm)

int MPI_Allgather(void* sendbuf, int sendcount, MPI_datatype sendtype,
                  void* recvbuf, int recvcount, MPI_datatype recvtype, MPI_Comm comm)

int MPI_Reduce_scatter(void* sendbuf, void* recvbuf, int* recvcounts, MPI_Datatype datatype,
                       MPI_Op op, MPI_Comm comm)

int MPI_Alltoall(void* sendbuf, int sendcount, MPI_datatype sendtype,
                 void* recvbuf, int recvcount, MPI_datatype recvtype, MPI_Comm comm)

int MPI_Alltoallv(void* sendbuf, int *sendcounts, int *sdispls, MPI_Datatype sendtype,
                  void* recvbuf, int *recvcounts, int *rdispls, MPI_Datatype recvtype,
                  MPI_Comm comm)
```

Available reduction operations

Operation	Meaning
`MPI_MAX`	Max value
`MPI_MIN`	Min value
`MPI_SUM`	Sum
`MPI_PROD`	Product
`MPI_MAXLOC`	Max value + location
`MPI_MINLOC`	Min value + location

Operation	Meaning
`MPI_LAND`	Logical AND
`MPI_BAND`	Bytewise AND
`MPI_LOR`	Logical OR
`MPI_BOR`	Bytewise OR
`MPI_LXOR`	Logical XOR
`MPI_BXOR`	Bytewise XOR

C interfaces for user-defined communicators

```
int MPI_Comm_split (MPI_Comm comm, int color, int key, MPI_Comm newcomm)

int MPI_Comm_compare (MPI_Comm comm1, MPI_Comm comm2, int result)

int MPI_Comm_dup ( MPI_Comm comm, MPI_Comm newcomm )

int MPI_Comm_free ( MPI_Comm comm )
```

C interfaces for non-blocking operations

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

int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag,
              MPI_Comm comm, MPI_Request *request )

int MPI_Wait(MPI_Request *request, MPI_Status *status)

int MPI_Waitall(int count, MPI_Request *array_of_requests, MPI_Status *array_of_stat
```

C interfaces for Cartesian process topologies

```
int MPI_Cart_create(MPI_Comm old_comm, int ndims, int *dims, int *periods, int reord
                    MPI_Comm *comm_cart)

int MPI_Dims_create(int ntasks, int ndims, int *dims);

int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdim, int *coords)

int MPI_Cart_rank(MPI_Comm comm, int *coords, int rank)

int MPI_Cart_shift( MPI_Comm comm, int direction, int displ, int *low, int *high )
```

C interfaces for persistent communication

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

int MPI_Recv_init(void *buf, int count, MPI_Datatype datatype, int source, int tag,
              MPI_Comm comm, MPI_Request *request )

int MPI_Start(MPI_Request *request)

int MPI_Startall(int count, MPI_Request *array_of_requests);
```

C interfaces for neighborhood collectives

```
int MPI_Neighbor_allgather(void* sendbuf, int sendcount, MPI_Datatype sendtype,
                           void* recvbuf, int recvcount, MPI_datatype recvtype, MPI_Comm comm);

int MPI_Neighbor_allgatherv(void* sendbuf, int sendcount, MPI_Datatype sendtype,
                            void* recvbuf, int* recvcounts, int* displs, MPI_datatype recvtype,
                            MPI_Comm comm);

int MPI_Neighbor_alltoall(void* sendbuf, int sendcount, MPI_Datatype sendtype,
                          void* recvbuf, int recvcount, MPI_datatype recvtype,
                          MPI_Comm comm);

int MPI_Neighbor_alltoallv(void* sendbuf, int* sendcounts, int* senddispls, MPI_Datatype sendtype,
                           void* recvbuf, int* recvcounts, int* recvdispls, MPI_datatype recvtype,
                           MPI_Comm comm);

int MPI_Neighbor_alltoallw(void* sendbuf, int* sendcounts, int* senddispls, MPI_Datatype* sendtypes,
                           void* recvbuf, int* recvcounts, int* recvdispls, MPI_datatype* recvtypes,
                           MPI_Comm comm);
```

C interfaces for datatype routines

```
int MPI_Type_commit(MPI_Datatype *type)

int MPI_Type_free(MPI_Datatype *type)

int MPI_Type_contiguous(int count, MPI_Datatype oldtype, MPI_Datatype *newtype)

int MPI_Type_vector(int count, int block, int stride, MPI_Datatype oldtype,
                    MPI_Datatype *newtype)

int MPI_Type_indexed(int count, int blocks[], int displs[], MPI_Datatype oldtype,
                     MPI_Datatype *newtype)

int MPI_Type_create_subarray(int ndims, int array_of_sizes[], int array_of_subsizes[],
                             int array_of_starts[], int order, MPI_Datatype oldtype,
                             MPI_Datatype *newtype )

int MPI_Type_create_struct(int count, const int array_of_blocklengths[],
                           const MPI_Aint array_of_displacements[],
                           const MPI_Datatype array_of_types[], MPI_Datatype *newtype)
```

C interfaces for one-sided routines

```c
int MPI_Win_create(void *base, MPI_Aint size, int disp_unit, MPI_Info info,
                   MPI_Comm comm, MPI_Win *win)

int MPI_Win_fence(int assert, MPI_Win win)

int MPI_Put(const void *origin_addr, int origin_count, MPI_Datatype origin_datatype,
                  int target_rank, MPI_Aint target_disp, int target_count,
                  MPI_Datatype target_datatype, MPI_Win win)

int MPI_Get(void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int t
            MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MP

int MPI_Accumulate(const void *origin_addr, int origin_count, MPI_Datatype origin_da
                   int target_rank, MPI_Aint target_disp, int target_count,
                   MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)
```

C interfaces to MPI I/O routines

```
int MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File

int MPI_File_close(MPI_File *fh)

int MPI_File_seek(MPI_File fh, MPI_Offset offset, int whence)

int MPI_File_read(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Stat

int MPI_File_read_at(MPI_File fh, MPI_Offset offset, void *buf, int count,
                     MPI_Datatype datatype, MPI_Status *status)

int MPI_File_write(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Sta

int MPI_File_write_at(MPI_File fh, MPI_Offset offset, void *buf, int count,
                      MPI_Datatype datatype, MPI_Status *status)
```

C interfaces to MPI I/O routines

```
int MPI_File_set_view(MPI_File fh, MPI_Offset disp, MPI_Datatype etype,
                      MPI_Datatype filetype, char *datarep, MPI_Info info)

int MPI_File_read_all(MPI_File fh, void *buf, int count, MPI_Datatype datatype,
                      MPI_Status *status)

int MPI_File_read_at_all(MPI_File fh, MPI_Offset offset, void *buf, int count,
                         MPI_Datatype datatype, MPI_Status *status)

int MPI_File_write_all(MPI_File fh, void *buf, int count, MPI_Datatype datatype,
                       MPI_Status *status)

int MPI_File_write_at_all(MPI_File fh, MPI_Offset offset, void *buf, int count,
                          MPI_Datatype datatype, MPI_Status *status)
```

C interfaces for environmental inquiries

```
int MPI_Get_processor_name(char *name, int *resultlen)
```

Fortran interfaces

Fortran interfaces for the “first six” MPI operations

```
mpi_init(ierror)
  integer :: ierror

mpi_init_thread(required, provided, ierror)
  integer :: required, provided, ierror

mpi_comm_size(comm, size, ierror)
mpi_comm_rank(comm, rank, ierror)
  type(mpi_comm) :: comm
  integer :: size, rank, ierror

mpi_barrier(comm, ierror)
  type(mpi_comm) :: comm
  integer :: ierror

mpi_finalize(ierror)
  integer :: ierror
```

Fortran interfaces for the basic point-to-point operations

```
mpi_send(buffer, count, datatype, dest, tag, comm, ierror)
  <type> :: buf(*)
  integer :: count, dest, tag, ierror
  type(mpi_datatype) :: datatype
  type(mpi_comm) :: comm

mpi_recv(buf, count, datatype, source, tag, comm, status, ierror)
  <type> :: buf(*)
  integer :: count, source, tag, ierror
  type(mpi_datatype) :: datatype
  type(mpi_comm) :: comm
  type(mpi_status) :: status
```

Fortran interfaces for the basic point-to-point operations

```
mpi_sendrecv(sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, &
             recvtype, source, recvtag, comm, status, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcount, recvcount, dest, source, sendtag, recvtag, ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm
  type(mpi_status) :: status

mpi_get_count(status, datatype, count, ierror)
  integer :: count, ierror
  type(mpi_datatype) :: datatype
  type(mpi_status) :: status
```

MPI datatypes for Fortran

MPI type	Fortran type
MPI_CHARACTER	character
MPI_INTEGER	integer
MPI_REAL	real32
MPI_DOUBLE_PRECISION	real64
MPI_COMPLEX	complex
MPI_DOUBLE_COMPLEX	double complex
MPI_LOGICAL	logical
MPI_BYTE	

Fortran interfaces for collective operations

```
mpi_bcast(buffer, count, datatype, root, comm, ierror)
  <type> :: buffer(*)
  integer :: count, root, ierror
  type(mpi_datatype) :: datatype
  type(mpi_comm) :: comm

mpi_scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm,
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcount, recvcount,  root, ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm

mpi_scatterv(sendbuf, sendcounts, displs, sendtype, recvbuf, recvcount, recvtype, &
             root, comm, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcounts(*), displs(*), recvcount, ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm
```

Fortran interfaces for collective operations

```
mpi_gather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, i
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcount, recvcount,  root, ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm

mpi_gatherv(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, &
            root, comm, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcount, recvcounts(*), displs(*), ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm

mpi_reduce(sendbuf, recvbuf, count, datatype, op, root, comm, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: count, root, ierror
  type(mpi_datatype) :: datatype
```

Fortran interfaces for collective operations

```
mpi_allreduce(sendbuf, recvbuf, count, datatype, op, comm, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: count, ierror
  type(mpi_datatype) :: datatype
  type(mpi_op) :: op
  type(mpi_comm) :: comm

mpi_allgather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierr
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcount, recvcount, ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm

mpi_reduce_scatter(sendbuf, recvbuf, recvcounts, datatype, op, comm, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: recvcounts(*), ierror
  type(mpi_datatype) :: datatype
```

Fortran interfaces for collective operations

```
mpi_alltoall(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierro
  type:: sendbuf(*), recvbuf(*)
  integer :: sendcount, recvcount, ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm

mpi_alltoallv(sendbuf,sendcounts, sdispls, sendtype, recvbuf, recvcounts, rdispls, r
              comm, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcounts(*), recvcounts(*), sdispls(*), rdispls(*), ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm
```

Available reduction operations

Operation	Meaning
MPI_MAX	Max value
MPI_MIN	Min value
MPI_SUM	Sum
MPI_PROD	Product
MPI_MAXLOC	Max value + location
MPI_MINLOC	Min value + location

Operation	Meaning
MPI_LAND	Logical AND
MPI_BAND	Bytewise AND
MPI_LOR	Logical OR
MPI_BOR	Bytewise OR
MPI_LXOR	Logical XOR
MPI_BXOR	Bytewise XOR

Fortran interfaces for user-defined communicators

```
mpi_comm_split(comm, color, key, newcomm, ierror)
  integer :: color, key, ierror
  type(mpi_comm) :: comm, newcomm

mpi_comm_compare(comm1, comm2, result, ierror)
  integer :: result, ierror
  type(mpi_comm) :: comm1, comm2

mpi_comm_dup(comm, newcomm, ierror)
  integer :: ierror
  type(mpi_comm) :: comm, newcomm

mpi_comm_free(comm, ierror)
  integer :: ierror
  type(mpi_comm) :: comm
```

Fortran interfaces for non-blocking operations

```
mpi_isend(buf, count, datatype, dest, tag, comm, request,ierror)
  <type> :: buf(*)
  integer :: count, dest, tag, ierror
  type(mpi_datatype) :: datatype
  type(mpi_request) :: request
  type(mpi_comm) :: comm

mpi_irecv(buf, count, datatype, source, tag, comm, request,ierror)
  <type> :: buf(*)
  integer :: count, source, tag, ierror
  type(mpi_datatype) :: datatype
  type(mpi_request) :: request
  type(mpi_comm) :: comm

mpi_wait(request, status, ierror)
  integer :: ierror
  type(mpi_request) :: request
  type(mpi_status) :: status

mpi_waitall(count, array_of_requests, array_of_statuses, ierror)
  integer :: count, ierror
  type(mpi_request) :: array_of_requests(:)
  type(mpi_status) :: array_of_statuses(:)
```

Fortran interfaces for Cartesian process topologies

```
mpi_cart_create(old_comm, ndims, dims, periods, reorder, comm_cart, ierror)
  integer :: ndims, dims(:), ierror
  type(mpi_comm) :: old_comm, comm_cart
  logical :: reorder, periods(:)

mpi_dims_create(ntasks, ndims, dims, ierror)
  integer :: ntasks, ndims, dims(:), ierror

mpi_cart_coords(comm, rank, maxdim, coords, ierror)
  integer :: rank, maxdim, coords(:), ierror
  type(mpi_comm) :: comm

mpi_cart_rank(comm, coords, rank, ierror)
  integer :: coords(:), rank, ierror
  type(mpi_comm) :: comm

mpi_cart_shift(comm, direction, displ, low, high, ierror)
  integer :: direction, displ, low, high, ierror
  type(mpi_comm) :: comm
```

Fortran interfaces for persistent communication

```
mpi_send_init(buf, count, datatype, dest, tag, comm, request,ierror)
  <type> :: buf(*)
  integer :: count, dest, tag, ierror
  type(mpi_datatype) :: datatype
  type(mpi_request) :: request
  type(mpi_comm) :: comm

mpi_recv_init(buf, count, datatype, source, tag, comm, request,ierror)
  <type> :: buf(*)
  integer :: count, source, tag, ierror
  type(mpi_datatype) :: datatype
  type(mpi_request) :: request
  type(mpi_comm) :: comm

mpi_start(request, ierror)
  integer :: ierror
  type(mpi_request) :: request

mpi_startall(count, array_of_requests, ierror)
  integer :: count, ierror
  type(mpi_request) :: array_of_requests(:)
```

Fortran interfaces for neighborhood collectives

```
mpi_neighbor_allgather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, c
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcount, recvcount, ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm

mpi_neighbor_allgatherv(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, r
                        comm, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcount, recvcounts(:), displs(:), ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm

mpi_neighbor_alltoall(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, co
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcount, recvcount, ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm
```

Fortran interfaces for neighborhood collectives

```
mpi_neighbor_alltoallv(sendbuf, sendcounts, sendtype, senddispls, &
                       recvbuf, recvcounts, recvdispls, recvtype, comm, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcounts(:), recvcounts(:), senddispls(:), recvdispls(:), ierror
  type(mpi_datatype) :: sendtype, recvtype
  type(mpi_comm) :: comm

mpi_neighbor_alltoallw(sendbuf, sendcounts, sendtypes, senddispls, &
                       recvbuf, recvcounts, recvdispls, recvtypes, comm, ierror)
  <type> :: sendbuf(*), recvbuf(*)
  integer :: sendcounts(:), recvcounts(:), senddispls(:), recvdispls(:), ierror
  type(mpi_datatype) :: sendtypes(:), recvtypes(:)
  type(mpi_comm) :: comm
```

Fortran interfaces for datatype routines

```
mpi_type_commit(type, ierror)
  type(mpi_datatype) :: type
  integer :: ierror

mpi_type_free(type, ierror)
  type(mpi_datatype) :: type
  integer :: ierror

mpi_type_contiguous(count, oldtype, newtype, ierror)
  integer :: count, ierror
  type(mpi_datatype) :: oldtype, newtype

mpi_type_vector(count, block, stride, oldtype, newtype, ierror)
  integer :: count, block, stride, ierror
  type(mpi_datatype) :: oldtype, newtype
```

Fortran interfaces for datatype routines

```
mpi_type_indexed(count, blocks, displs, oldtype, newtype, ierror)
  integer :: count, ierror
  integer, dimension(count) :: blocks, displs
  type(mpi_datatype) :: oldtype, newtype

mpi_type_create_subarray(ndims, sizes, subsizes, starts, order, oldtype, newtype, ie
  integer :: ndims, order, ierror
  integer, dimension(ndims) :: sizes, subsizes, starts
  type(mpi_datatype) :: oldtype, newtype

mpi_type_create_struct(count, blocklengths, displacements, types, newtype, ierror)
  integer :: count, blocklengths(count), ierror
  type(mpi_datatype) :: types(count), newtype
  integer(kind=mpi_address_kind) :: displacements(count)
```

Fortran interfaces for one-sided routines

```
mpi_win_create(base, size, disp_unit, info, comm, win, ierror)
  <type> :: base(*)
  integer(kind=mpi_address_kind) :: size
  integer :: disp_unit, ierror
  type(mpi_info) :: info
  type(mpi_comm) :: comm
  type(mpi_win)  :: win

mpi_win_fence(assert, win, ierror)
  integer :: assert, ierror
  type(mpi_win)  :: win

mpi_put(origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, &
        target_datatype, win, ierror)
  <type> :: origin_addr(*)
  integer(kind=mpi_address_kind) :: target_disp
  integer :: origin_count, target_rank, target_count,  ierror
  type(mpi_datatype) :: origin_datatype, target_datatype
  type(mpi_win)  :: win
```

Fortran interfaces for one-sided routines

```
mpi_get(origin_addr, origin_count, origin_datatype, target_rank, target_disp, target
        target_datatype, win, ierror)
  <type> :: origin_addr(*)
  integer(kind=mpi_address_kind) :: target_disp
  integer :: origin_count, target_rank, target_count,  ierror
  type(mpi_datatype) :: origin_datatype, target_datatype
  type(mpi_win)  :: win

mpi_accumulate(origin_addr, origin_count, origin_datatype, target_rank, target_disp,
               target_count, target_datatype, op, win, ierror)
  <type> :: origin_addr(*)
  integer(kind=mpi_address_kind) :: target_disp
  integer :: origin_count, target_rank, target_count,  ierror
  type(mpi_datatype) :: origin_datatype, target_datatype
  type(mpi_op) :: op
  type(mpi_win)  :: win
```

Fortran interfaces for MPI I/O routines

```
mpi_file_open(comm, filename, amode, info, fh, ierror)
  integer :: amode, ierror
  character* :: filename
  type(mpi_info) :: info
  type(mpi_file) :: fh
  type(mpi_comm) :: comm

mpi_file_close(fh, ierror)
  integer :: ierror
  type(mpi_file) :: fh

mpi_file_seek(fh, offset, whence, ierror)
  integer(kind=MPI_OFFSET_KIND) :: offset
  integer :: whence, ierror
  type(mpi_file) :: fh
```

Fortran interfaces for MPI I/O routines

```
mpi_file_read(fh, buf, count, datatype, status, ierror)
mpi_file_write(fh, buf, count, datatype, status, ierror)
  <type> :: buf(*)
  integer :: count, ierror
  type(mpi_file) :: fh
  type(mpi_datatype) :: datatype
  type(mpi_status) :: status

mpi_file_read_at(fh, offset, buf, count, datatype, status, ierror)
mpi_file_write_at(fh, offset, buf, count, datatype, status, ierror)
  <type> :: buf(*)
  integer(kind=MPI_OFFSET_KIND) :: offset
  integer :: count, ierror
  type(mpi_file) :: fh
  type(mpi_datatype) :: datatype
  type(mpi_status) :: status
```

Fortran interfaces for MPI I/O routines

```
mpi_file_set_view(fh, disp, etype, filetype, datarep, info, ierror)
  integer :: ierror
  integer(kind=MPI_OFFSET_KIND) :: disp
  type(mpi_info) :: info
  character* :: datarep
  type(mpi_file) :: fh
  type(mpi_datatype) :: etype, datatype

mpi_file_read_all(fh, buf, count, datatype, status, ierror)
mpi_file_write_all(fh, buf, count, datatype, status, ierror)
  <type> :: buf(*)
  integer :: count, ierror
  type(mpi_file) :: fh
  type(mpi_datatype) :: datatype
  type(mpi_status) :: status
```

Fortran interfaces for MPI I/O routines

```
mpi_file_read_at_all(fh, offset, buf, count, datatype, status, ierror)
mpi_file_write_at_all(fh, offset, buf, count, datatype, status, ierror)
  <type> :: buf(*)
  integer(kind=MPI_OFFSET_KIND) :: offset
  integer :: count, ierror
  type(mpi_file) :: fh
  type(mpi_datatype) :: datatype
  type(mpi_status) :: status
```

Fortra interfaces for environmental inquiries

```
mpi_get_processor_name(name, resultlen, ierror)
  character(len=MPI_MAX_PROCESSOR_NAME) :: name
  integer :: resultlen, ierror
```