MPI_REDUCE_LOCAL(inbuf, inoutbuf, count, datatype, op)

IN	inbuf	input buffer (choice)
INOUT	inoutbuf	combined input and output buffer (choice)
IN	count	number of elements in inbuf and inoutbuf buffers (non-negative integer)
IN	datatype	data type of elements of inbuf and inoutbuf buffers (handle)
IN	op	operation (handle)

```
int MPI_Reduce_local(void* inbuf, void* inoutbuf, int count,
             MPI_Datatype datatype, MPI_Op op)

MPI_REDUCE_LOCAL(INBUF, INOUBUF, COUNT, DATATYPE, OP, IERROR)
    <type> INBUF(*), INOUTBUF(*)
    INTEGER COUNT, DATATYPE, OP, IERROR

{void MPI::Op::Reduce_local(const void* inbuf, void* inoutbuf, int count,
             const MPI::Datatype& datatype) const (binding deprecated, see
             Section 15.2) }
```

ticket150.

ticket150.

The function applies the operation given by op element-wise to the elements of inbuf and inoutbuf with the result stored element-wise in inoutbuf, as explained for user-defined operations in Section 5.9.5. Both inbuf and inoutbuf (input as well as result) have the same number of elements given by count and the same datatype given by datatype. The MPI_IN_PLACE option is not allowed.

Reduction operations can be queried for their commutativity.

MPI_OP_COMMUTATIVE(op, commute)

IN	op	operation (handle)
OUT	commute	true if op is commutative, false otherwise (logical)

```
int MPI_Op_commutative(MPI_Op op, int *commute)

MPI_OP_COMMUTATIVE(OP, COMMUTE, IERROR)
    LOGICAL COMMUTE
    INTEGER OP, IERROR

{bool MPI::Op::Is_commutative() const (binding deprecated, see Section 15.2) }
```

ticket150.

ticket150.

5.10 Reduce-Scatter

ticket27.

[MPI includes a variant of the reduce operations where the result is scattered to all processes in a group on return.]MPI includes variants of the reduce operations where the result is scattered to all processes in a group on return. One variant scatters equal-sized blocks to all processes, while another variant scatters blocks that may vary in size for each process.

ticket27.

ticket27.

5.10.1 MPI_REDUCE_SCATTER_BLOCK

MPI_REDUCE_SCATTER_BLOCK(sendbuf, recvbuf, recvcount, datatype, op, comm)

IN	sendbuf	starting address of send buffer (choice)
OUT	recvbuf	starting address of receive buffer (choice)
IN	recvcount	element count per block (non-negative integer)
IN	datatype	data type of elements of send and receive buffers (handle)
IN	op	operation (handle)
IN	comm	communicator (handle)

```
int MPI_Reduce_scatter_block(void* sendbuf, void* recvbuf, int recvcount,
             MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

MPI_REDUCE_SCATTER_BLOCK(SENDBUF, RECVBUF, RECVCOUNT, DATATYPE, OP, COMM,
             IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER RECVCOUNT, DATATYPE, OP, COMM, IERROR
```

ticket150.

```
{void MPI::Comm::Reduce_scatter_block(const void* sendbuf, void* recvbuf,
             int recvcount, const MPI::Datatype& datatype,
```

ticket150.

```
             const MPI::Op& op) const = 0 (binding deprecated, see Section 15.2) }
```

If comm is an intracommunicator, MPI_REDUCE_SCATTER_BLOCK first performs a global, element-wise reduction on vectors of count = n*recvcount elements in the send buffers defined by sendbuf, count and datatype, using the operation op, where n is the number of processes in the group of comm. The routine is called by all group members using the same arguments for recvcount, datatype, op and comm. The resulting vector is treated as n consecutive blocks of recvcount elements that are scattered to the processes of the group. The i-th block is sent to process i and stored in the receive buffer defined by recvbuf, recvcount, and datatype.

> *Advice to implementors.* The MPI_REDUCE_SCATTER_BLOCK routine is functionally equivalent to: an MPI_REDUCE collective operation with count equal to recvcount*n, followed by an MPI_SCATTER with sendcount equal to recvcount. However, a direct implementation may run faster. (*End of advice to implementors.*)

The "in place" option for intracommunictors is specified by passing MPI_IN_PLACE in the sendbuf argument on *all* processes. In this case, the input data is taken from the receive buffer.

If comm is an intercommunicator, then the result of the reduction of the data provided by processes in one group (group A) is scattered among processes in the other group (group B) and vice versa. Within each group, all processes provide the same value for the recvcount argument, and provide input vectors of count = n*recvcount elements stored in the send buffers, where n is the size of the group. The number of elements count must be the same for the two groups. The resulting vector from the other group is scattered in blocks of recvcount elements among the processes in the group.

Rationale. The last restriction is needed so that the length of the send buffer of one group can be determined by the local recvcount argument of the other group. Otherwise, a communication is needed to figure out how many elements are reduced. (*End of rationale.*)

ticket27.

5.10.2 MPI_REDUCE_SCATTER

MPI_REDUCE_SCATTER extends the functionality of MPI_REDUCE_SCATTER_BLOCK such that the scattered blocks can vary in size. Block sizes are determined by the recvcounts array, such that the i-th block contains recvcounts[i] elements.

MPI_REDUCE_SCATTER(sendbuf, recvbuf, recvcounts, datatype, op, comm)

IN	sendbuf	starting address of send buffer (choice)
OUT	recvbuf	starting address of receive buffer (choice)
IN	recvcounts	non-negative integer array (of length group size) specifying the number of elements [in]of the result distributed to each process. [Array must be identical on all calling processes.]
IN	datatype	data type of elements of [input buffer]send and receive buffers (handle)
IN	op	operation (handle)
IN	comm	communicator (handle)

ticket93.
ticket124.
ticket124.
ticket124.

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

MPI_REDUCE_SCATTER(SENDBUF, RECVBUF, RECVCOUNTS, DATATYPE, OP, COMM,
              IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER RECVCOUNTS(*), DATATYPE, OP, COMM, IERROR

{void MPI::Comm::Reduce_scatter(const void* sendbuf, void* recvbuf,
              int recvcounts[], const MPI::Datatype& datatype,
              const MPI::Op& op) const = 0 (binding deprecated, see Section 15.2) }
```

ticket150.
ticket150.

If comm is an intracommunicator, MPI_REDUCE_SCATTER first [does an element-wise reduction on vector of count = $\sum_i$ recvcounts[i] elements in the send buffer defined by sendbuf, count and datatype. Next, the resulting vector of results is split into n disjoint segments, where n is the number of members in the group. Segment i contains recvcounts[i] elements. The i-th segment] performs a global, element-wise reduction on vectors of count = $\sum_{i=0}^{n-1}$ recvcounts[i] elements in the send buffers defined by sendbuf, count and datatype, using the operation op, where n is the number of processes in the group of comm. The routine is called by all group members using the same arguments for recvcounts, datatype, op and comm. The resulting vector is treated as n consecutive blocks where the number of elements of the i-th block is recvcounts[i]. The blocks are scattered to the

ticket124.

Annex B

Change-Log

This annex summarizes changes from the previous version of the MPI standard to the version presented by this document. [Only changes (i.e., clarifications and new features) are presented that may cause implementation effort in the MPI libraries.] Only significant changes (i.e., clarifications and new features) that might either require implementation effort in the MPI libraries or change the understanding of MPI from a user's perspective are presented. Editorial modifications, formatting, typo corrections and minor clarifications are not shown.

B.1 Changes from Version 2.1 to Version 2.2

1. Section 2.5.4 on page 14.
   It is now guaranteed that predefined named constant handles (as other constants) can be used in initialization expressions or assignments, i.e., also before the call to MPI_INIT.

2. Section 2.6 on page 16, Section 2.6.4 on page 19, and Section 16.1 on page 469.
   The C++ language bindings have been deprecated and will be removed in a future version of the MPI specification.

3. Section 3.2.2 on page 29.
   MPI_CHAR for printable characters is now defined for C type char (instead of signed char). This change should not have any impact on applications nor on MPI libraries (except some comment lines), because printable characters could and can be stored in any of the C types char, signed char, and unsigned char, and MPI_CHAR is not allowed for predefined reduction operations.

4. Section 3.2.2 on page 29.
   MPI_(U)INT{8,16,32,64}_T, MPI_AINT, MPI_OFFSET, MPI_C_BOOL, MPI_C_COMPLEX, MPI_C_FLOAT_COMPLEX, MPI_C_DOUBLE_COMPLEX, and MPI_C_LONG_DOUBLE_COMPLEX are now valid predefined MPI datatypes.

5. Section 3.4 on page 40, Section 3.7.2 on page 51, Section 3.9 on page 71, and Section 5.1 on page 133.
   The read access restriction on the send buffer for blocking, non blocking and collective API has been lifted. It is permitted to access for read the send buffer while the operation is in progress.

6. Section 3.7 on page 50.
   The Advice to users for IBSEND and IRSEND was slightly changed. ticket143.

7. Section 3.7.3 on page 54.
   The advice to free an active request was removed in the Advice to users for MPI_REQUEST_FREE. ticket137.

8. Section 3.7.6 on page 66.
   MPI_REQUEST_GET_STATUS changed to permit inactive or null requests as input. ticket31.

9. Section 5.8 on page 159.
   "In place" option is added to MPI_ALLTOALL, MPI_ALLTOALLV, and MPI_ALLTOALLW for intracommunicators. ticket64.

10. Section 5.9.2 on page 167.
    Predefined parameterized datatypes (e.g., returned by MPI_TYPE_CREATE_F90_REAL) and optional named predefined datatypes (e.g. MPI_REAL8) have been added to the list of valid datatypes in reduction operations. ticket18.

11. Section 5.9.2 on page 167.
    MPI_(U)INT{8,16,32,64}_T are all considered C integer types for the purposes of the predefined reduction operators. MPI_AINT and MPI_OFFSET are considered Fortran integer types. MPI_C_BOOL is considered a Logical type.
    MPI_C_COMPLEX, MPI_C_FLOAT_COMPLEX, MPI_C_DOUBLE_COMPLEX, and MPI_C_LONG_DOUBLE_COMPLEX are considered Complex types. ticket24.

12. Section 5.9.7 on page 178.
    The local routines MPI_REDUCE_LOCAL and MPI_OP_COMMUTATIVE have been added. ticket27.

13. Section 5.10.1 on page 180.
    The collective function MPI_REDUCE_SCATTER_BLOCK is added to the MPI standard. ticket94.

14. Section 5.11.2 on page 183.
    Added in place argument to MPI_EXSCAN. ticket19.

15. Section 6.4.2 on page 202, and Section 6.6 on page 222.
    Implementations that did not implement MPI_COMM_CREATE on intercommunicators will need to add that functionality. As the standard described the behavior of this operation on intercommunicators, it is believed that most implementations already provide this functionality. Note also that the C++ binding for both MPI_COMM_CREATE and MPI_COMM_SPLIT explicitly allow Intercomms. ticket66.

16. Section 6.4.2 on page 202.
    MPI_COMM_CREATE is extended to allow several disjoint subgroups as input if comm is an intracommunicator. If comm is an intercommunicator it was clarified that all processes in the same local group of comm must specify the same value for group. ticket33.

17. Section ?? on page ??.
    New functions for a scalable distributed graph topology interface has been added. In this section, the functions MPI_DIST_GRAPH_CREATE_ADJACENT and