5.9.2 Predefined Reduction Operations

The following predefined operations are supplied for MPI_REDUCE and related functions MPI_ALLREDUCE, MPI_REDUCE_SCATTER, MPI_SCAN, and MPI_EXSCAN. These operations are invoked by placing the following in op.

Name	Meaning
MPI_MAX	maximum
MPI_MIN	minimum
MPI_SUM	sum
MPI_PROD	product
MPI_LAND	logical and
MPI_BAND	bit-wise and
MPI_LOR	logical or
MPI_BOR	bit-wise or
MPI_LXOR	logical exclusive or (xor)
MPI_BXOR	bit-wise exclusive or (xor)
MPI_MAXLOC	max value and location
MPI_MINLOC	min value and location

The two operations MPI_MINLOC and MPI_MAXLOC are discussed separately in Section 5.9.4. For the other predefined operations, we enumerate below the allowed combinations of op and datatype arguments. First, define groups of MPI basic datatypes in the following way.

C integer:	MPI_INT, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG, MPI_LONG_LONG_INT, MPI_LONG_LONG (as synonym), MPI_UNSIGNED_LONG_LONG, MPI_SIGNED_CHAR, MPI_UNSIGNED_CHAR, ticket18. MPI_INT8_T, MPI_INT16_T, ticket18. MPI_INT32_T, MPI_INT64_T, MPI_UINT8_T, MPI_UINT16_T, MPI_UINT32_T, MPI_UINT64_T
Fortran integer:	MPI_INTEGER, MPI_AINT, MPI_OFFSET, ticket18. and handles returned from ticket64. MPI_TYPE_CREATE_F90_INTEGER, and if available: MPI_INTEGER1, MPI_INTEGER2, MPI_INTEGER4, MPI_INTEGER8, MPI_INTEGER16
Floating point:	MPI_FLOAT, MPI_DOUBLE, MPI_REAL, MPI_DOUBLE_PRECISION MPI_LONG_DOUBLE ticket64. and handles returned from MPI_TYPE_CREATE_F90_REAL,

		and if available: MPI_REAL2,
		MPI_REAL4, MPI_REAL8, MPI_REAL16
ticket18.	Logical:	MPI_LOGICAL, MPI_C_BOOL
ticket18.	Complex:	MPI_COMPLEX,
ticket18.		MPI_C_FLOAT_COMPLEX,
		MPI_C_DOUBLE_COMPLEX,
ticket64.		MPI_C_LONG_DOUBLE_COMPLEX,
		and handles returned from
		MPI_TYPE_CREATE_F90_COMPLEX,
		and if available: MPI_DOUBLE_COMPLEX,
		MPI_COMPLEX4, MPI_COMPLEX8,
		MPI_COMPLEX16, MPI_COMPLEX32
	Byte:	MPI_BYTE

Now, the valid datatypes for each option is specified below.

Op	Allowed Types
MPI_MAX, MPI_MIN	C integer, Fortran integer, Floating point
MPI_SUM, MPI_PROD	C integer, Fortran integer, Floating point, Complex
MPI_LAND, MPI_LOR, MPI_LXOR	C integer, Logical
MPI_BAND, MPI_BOR, MPI_BXOR	C integer, Fortran integer, Byte

The following examples use intracommunicators.

Example 5.15 A routine that computes the dot product of two vectors that are distributed across a group of processes and returns the answer at node zero.

```
SUBROUTINE PAR_BLAS1(m, a, b, c, comm)
REAL a(m), b(m)       ! local slice of array
REAL c                ! result (at node zero)
REAL sum
INTEGER m, comm, i, ierr

! local sum
sum = 0.0
DO i = 1, m
   sum = sum + a(i)*b(i)
END DO

! global sum
CALL MPI_REDUCE(sum, c, 1, MPI_REAL, MPI_SUM, 0, comm, ierr)
RETURN
```

Example 5.16 A routine that computes the product of a vector and an array that are distributed across a group of processes and returns the answer at node zero.

```
SUBROUTINE PAR_BLAS2(m, n, a, b, c, comm)
REAL a(m), b(m,n)    ! local slice of array
```

Example 16.3 Example using assignment operator. In this example, MPI::Intracomm::Dup() is *not* called for foo_comm. The object foo_comm is simply an alias for MPI::COMM_WORLD. But bar_comm is created with a call to MPI::Intracomm::Dup() and is therefore a different communicator than foo_comm (and thus different from MPI::COMM_WORLD). baz_comm becomes an alias for bar_comm. If one of bar_comm or baz_comm is freed with MPI_COMM_FREE it will be set to MPI::COMM_NULL. The state of the other handle will be undefined — it will be invalid, but not necessarily set to MPI::COMM_NULL.

```
MPI::Intracomm foo_comm, bar_comm, baz_comm;

foo_comm = MPI::COMM_WORLD;
bar_comm = MPI::COMM_WORLD.Dup();
baz_comm = bar_comm;
```

Comparison The comparison operators are prototyped as follows: ticket150.

{bool MPI::<CLASS>::operator==(const MPI::<CLASS>& data) const *(binding deprecated, see Section 15.2)* } ticket150.

ticket150.

{bool MPI::<CLASS>::operator!=(const MPI::<CLASS>& data) const *(binding deprecated, see Section 15.2)* } ticket150.

The member function operator==() returns true only when the handles reference the same internal MPI object, false otherwise. operator!=() returns the boolean complement of operator==(). However, since the Status class is not a handle to an underlying MPI object, it does not make sense to compare Status instances. Therefore, the operator==() and operator!=() functions are not defined on the Status class.

Constants Constants are singleton objects and are declared const. Note that not all globally defined MPI objects are constant. For example, MPI::COMM_WORLD and MPI::COMM_SELF are not const.

16.1.6 C++ Datatypes

Table 16.1 lists all of the C++ predefined MPI datatypes and their corresponding C and C++ datatypes, Table 16.2 lists all of the Fortran predefined MPI datatypes and their corresponding Fortran 77 datatypes. Table 16.3 lists the C++ names for all other MPI datatypes.

MPI::BYTE and MPI::PACKED conform to the same restrictions as MPI_BYTE and MPI_PACKED, listed in Sections 3.2.2 on page 29 and Sections 4.2 on page 125, respectively.

The following table defines groups of MPI predefined datatypes:

C integer:	MPI::INT, MPI::LONG, MPI::SHORT, MPI::UNSIGNED_SHORT, MPI::UNSIGNED, MPI::UNSIGNED_LONG, MPI::_LONG_LONG, MPI::UNSIGNED_LONG_LONG, MPI::SIGNED_CHAR, MPI::UNSIGNED_CHAR
Fortran integer:	MPI::INTEGER ticket64.
	and handles returned from

MPI datatype	C datatype	C++ datatype
MPI::CHAR	char	char
MPI::SHORT	signed short	signed short
MPI::INT	signed int	signed int
MPI::LONG	signed long	signed long
MPI::LONG_LONG	signed long long	signed long long
MPI::SIGNED_CHAR	signed char	signed char
MPI::UNSIGNED_CHAR	unsigned char	unsigned char
MPI::UNSIGNED_SHORT	unsigned short	unsigned short
MPI::UNSIGNED	unsigned int	unsigned int
MPI::UNSIGNED_LONG	unsigned long	unsigned long int
MPI::UNSIGNED_LONG_LONG	unsigned long long	unsigned long long
MPI::FLOAT	float	float
MPI::DOUBLE	double	double
MPI::LONG_DOUBLE	long double	long double
MPI::BOOL		bool
MPI::COMPLEX		Complex<float>
MPI::DOUBLE_COMPLEX		Complex<double>
MPI::LONG_DOUBLE_COMPLEX		Complex<long double>
MPI::WCHAR	wchar_t	wchar_t
MPI::BYTE		
MPI::PACKED		

Table 16.1: C++ names for the MPI C and C++ predefined datatypes, and their corresponding C/C++ datatypes.

MPI datatype	Fortran datatype
MPI::INTEGER	INTEGER
MPI::REAL	REAL
MPI::DOUBLE_PRECISION	DOUBLE PRECISION
MPI::F_COMPLEX	COMPLEX
MPI::LOGICAL	LOGICAL
MPI::CHARACTER	CHARACTER(1)
MPI::BYTE	
MPI::PACKED	

Table 16.2: C++ names for the MPI Fortran predefined datatypes, and their corresponding Fortran 77 datatypes.

MPI datatype	Description
MPI::FLOAT_INT	C/C++ reduction type
MPI::DOUBLE_INT	C/C++ reduction type
MPI::LONG_INT	C/C++ reduction type
MPI::TWOINT	C/C++ reduction type
MPI::SHORT_INT	C/C++ reduction type
MPI::LONG_DOUBLE_INT	C/C++ reduction type
MPI::TWOREAL	Fortran reduction type
MPI::TWODOUBLE_PRECISION	Fortran reduction type
MPI::TWOINTEGER	Fortran reduction type
MPI::F_DOUBLE_COMPLEX	Optional Fortran type
MPI::INTEGER1	Explicit size type
MPI::INTEGER2	Explicit size type
MPI::INTEGER4	Explicit size type
MPI::INTEGER8	Explicit size type
MPI::REAL4	Explicit size type
MPI::REAL8	Explicit size type
MPI::REAL16	Explicit size type

Table 16.3: C++ names for other MPI datatypes. Implementations may also define other optional types (e.g., `MPI::INTEGER8`).

	MPI::Datatype::Create_f90_integer, and if available: MPI::INTEGER1, MPI::INTEGER2, MPI::INTEGER4, MPI::INTEGER8, MPI::INTEGER16
Floating point:	MPI::FLOAT, MPI::DOUBLE, MPI::REAL, MPI::DOUBLE_PRECISION, MPI::LONG_DOUBLE ticket64. and handles returned from MPI::Datatype::Create_f90_real, and if available: MPI::REAL2, MPI::REAL4, MPI::REAL8, MPI::REAL16
Logical:	MPI::LOGICAL, MPI::BOOL
Complex:	MPI::F_COMPLEX, MPI::COMPLEX, MPI::F_DOUBLE_COMPLEX, MPI::DOUBLE_COMPLEX, MPI::LONG_DOUBLE_COMPLEX ticket64. and handles returned from MPI::Datatype::Create_f90_complex, and if available: MPI::F_DOUBLE_COMPLEX, MPI::F_COMPLEX4, MPI::F_COMPLEX8, MPI::F_COMPLEX16, MPI::F_COMPLEX32
Byte:	MPI::BYTE

Valid datatypes for each reduction operation are specified below in terms of the groups defined above.

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

7. Section 3.7.3 on page 55.
   The advice to free an active request was removed in the Advice to users for
   ticket137. MPI_REQUEST_FREE.

8. Section 3.7.6 on page 67.
   ticket31. MPI_REQUEST_GET_STATUS changed to permit inactive or null requests as input.

9. Section 5.8 on page 161.
   "In place" option is added to MPI_ALLTOALL, MPI_ALLTOALLV, and
   ticket64. MPI_ALLTOALLW for intracommunicators.

10. Section 5.9.2 on page 169.
    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
    ticket18. list of valid datatypes in reduction operations.

11. Section 5.9.2 on page 169.
    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
    ticket24. MPI_C_LONG_DOUBLE_COMPLEX are considered Complex types.

12. Section 5.9.7 on page 180.
    The local routines MPI_REDUCE_LOCAL and MPI_OP_COMMUTATIVE have been
    ticket27. added.

13. Section 5.10.1 on page 182.
    The collective function MPI_REDUCE_SCATTER_BLOCK is added to the MPI stan-
    ticket94. dard.

14. Section 5.11.2 on page 185.
    ticket19. Added in place argument to MPI_EXSCAN.

15. Section 6.4.2 on page 204, and Section 6.6 on page 224.
    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
    ticket66. MPI_COMM_CREATE and MPI_COMM_SPLIT explicitly allow Intercomms.

16. Section 6.4.2 on page 204.
    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
    ticket33. processes in the same local group of comm must specify the same value for group.

17. Section 7.5.4 on page 268.
    New functions for a scalable distributed graph topology interface has been added. In this section, the functions MPI_DIST_GRAPH_CREATE_ADJACENT and