pt2pt wg

FP16

Virtual MPI Forum Meeting
Jan 31, 2018

Atsushi Hori
RIKEN AICS

New Ticket #74

- Stop introducing new types
  MPI standard only defines *fundamental* data types such as MPI_INT, MPI_FLOAT, and so on
  - decouple type definition from standard
  - to give implementors the freedom of choice when to add new basic data types
- Instead,
  - Standard defines the *naming rule* of those basic data type names
  - Implementors can add new basic data types having the names following the naming tule

Related Tickets

- #65 16-bit floating-point support for C/C++
  - ➡ MPI standard will not define FP16, but implementor can decide to support or not
- #66 define language-agnostic, IEEE types
  - ➡ Implementors can decide to implement new types
- #69 Definition and Explanation of External32
  - ➡ Left unchanged because of backward compatibility

Definition

- *Basic data types*
  - primitive data types
    - can be appeared in type maps
  - Two kinds
    - *Fundamental data types*
    - *Additional data types*
- *Fundamental data types*
  - Defined in the standard
- *Additional data types*
  - Added by an implementor

Modification to the standard

- Fundamental data type definition
- Data type naming rule
- Addition of fundamental data type names
  - Old basic data type names become synonyms (and deprecated ?)
- MPI_type_equivalent() function
  - to check if two data types are equivalent or not in an implementation

Impact to users

- Old types are left as synonyms and no impact to current users

Details (1)

- Fundamental Data Types
  - C MPI_CHAR, MPI_INT, MPI_LONG, …
  - Fortran MPI_CHARACTER, MPI_INTEGER, …
  - MPI MPI_BYTE, MPI_PACKED, MPI_AINT, …
- *Implementation-defined* Data Types
  - *(Optional) Data Type Naming Rule* (later)
  - An implementation may add basic data type(s) if the implementation and associated compiler support it (them).
  - The current data type names (<= 3.2) not compliant with the rule are thought to be synonyms. (deprecated in the future ?)

Details (2)

- Advice to users (informal text)
  - Excepting external32, MPI standard only defines the type names, regardless to the sizes of datatypes, data representation format, if the data type is computed as is or any format conversion(s) is (are) accompanied with basics computation(s) [4 calculus, bit ops, etc.]. An implementation may add basic data types supported by the accompanied compiler. The data type names added by an implementation follow the data type naming rule in the standard so that users can identify the data type of the underlying programming language form the data type name used in MPI function calls. If any, documentation on these additional data types should be provided by the implementor.
  - Actual storage size of a data type depends on an implementation, and accompanied compiler (see also MPI_Type_size(x)). For example, MPI_LONG_INT has the size of 4 bytes or 8 bytes depending on the compiler and/or processor. In the usage of MPI-IO with "external32," however, data type sizes and formats are explicitly defined for compatibility (see also MPI-IO chapter XYZ).
- Advice to implementors (informal text)
  - If some or all global reduction operations are offloaded to a device other than the host processor [i.e., network device], then the data sizes and formats of all data types must be equivalent to the ones of the processor.

New Data Type Function

- int MPI_Type_equivalent(type0, type1, *flag)
  - *flag* is set, if *type0* and *type1* are having the same binary format and length. Otherwise *flag* is reset.
  - If derived data types are given, then the type maps of both types must be equivalent. Here, “equivalent type maps” means all basic data type and offset pairs in two type maps are the “equivalent” defined by MPI_Type_equivalent().
  - ex)
    - MPI_Type_equivalent(MPI_SHORT_FLOAT, MPI_FLOAT, flag)
    - MPI_Type_equivalent(MPI_REAL, MPI_FLOAT, flag)

Type Naming Rule (1/3)

- MPI_<CAT>_<TYPENAME>[_<EXTRA>]
  - examples
    - MPI_C_LONG_LONG
    - MPI_CXX_FLOAT
    - MPI_F_DOUBLE
    - MPI_F_REAL_2, …
- MPI-defined types
  - MPI_MPI_BYTE, MPI_MPI_PACKED, …
- Language agnostic types (if somebody wants to add)
  - MPI_IEEE_BINARY32
  - MPI_ISO_DECIMAL64, …

Type Naming Rule (2/3)

- MPI_<KIND>_<TYPENAME>[_<EXTRA>]
- KIND
  - Language "C", "CXX", "F", …
  - MPI-defined "MPI"
  - Standard "ISO", "IEEE", "ANSI", …
- Type Name (defined by language, shortest one)
  - C/C++ "int", "long long", …
  - Fortran "INTEGER", "REAL", …
  - ISO/IEEE "binary16", "decimal32", ..
- Extra
  - Fortran "REAL(4)", "COMPLEX(8)", …

Type Naming Rule (3/3)

- MIN_LOC, MAX_LOC (just in case)
  - <TYPENAME0>_AND_<TYPENAME1> or
  - <TYPENAME0>_AT_<TYPENAME1>

  - MPI_C_FLOAT_AND_MPI_C_INT
  - MPI_C_FLOAT_AT_MPI_C_INT

  - Voting ?

Old and New Type Names (1/2)

C Kind		Ext32
Current Name	**Proposed New Name**	**Size**
MPI_CHAR	**MPI_C_CHAR**	1
MPI_SHORT	**MPI_C_SHORT**	2
MPI_INT	**MPI_C_INT**	4
MPI_LONG	**MPI_C_LONG**	4
MPI_LONG_LONG_INT	**MPI_C_LONG_LONG**	8
MPI_LONG_LONG	**MPI_C_LONG_LONG**	8
MPI_SIGNED_CHAR	**MPI_C_SINGED_CHAR**	1
M:PI_UNSIGNED_CHAR	**MPI_C_UNSIGNED_CHAR**	1
MPI_UNSIGNED_SHORT	**MPI_C_UNSIGNED_SHORT**	2
MPI_UNSIGNED	**MPI_C_UNSIGNED**	4
MPI_UNSIGNED_LONG	**MPI_C_UNSIGNED_LONG**	4
MPI_UNSIGNED_LONG_LONG	**MPI_C_UNSIGNED_LONG_LONG**	8
MPI_FLOAT	**MPI_C_FLOAT**	4
MPI_SHORT_FLOAT	MPI_C_SHORT_FLOAT	(2)
MPI_DOUBLE	**MPI_C_DOUBLE**	8
MPI_LONG_DOUBLE	**MPI_C_LONG_DOUBLE**	16
MPI_WCHAR	**MPI_C_WCHAR**	2
MPI_C_BOOL	**MPI_C_BOOL**	1
MPI_INT8_T	**MPI_C_INT8**	1
MPI_INT16_T	**MPI_C_INT16**	2
MPI_INT32_T	**MPI_C_INT32**	4
MPI_INT64_T	**MPI_C_INT64**	8
MPI_UINT8_T	**MPI_C_UINT8**	1
MPI_UINT16_T	**MPI_C_UINT16**	2
MPI_UINT32_T	**MPI_C_UINT32**	4
MPI_UINT64_T	**MPI_C_UINT64**	8
MPI_C_COMPLEX	**MPI_C_COMPLEX**	2*4
MPI_C_FLOAT_COMPLEX	**MPI_C_COMPLEX**	2*4
MPI_C_SHORT_COMPLEX	MPI_C_SHORT_COMPLEX	(2*2)
MPI_C_DOUBLE_COMPLEX	**MPI_C_DOUBLE_COMPLEX**	2*8
MPI_C_LONG_DOUBLE_COMPLEX	**MPI_C_LONG_DOUBLE_COMPLEX**	2*16

Fortran Kind		Ext32
Current Name	**Proposed New Name**	**Size**
MPI CHARACTER	**MPI F CHARACTER**	1
MPI LOGICAL	**MPI F LOGICAL**	4
MPI INTEGER	**MPI F INTEGER**	4
MPI REAL	**MPI F REAL**	4
MPI DOUBLE PRECISION	**MPI_F_DOUBLE_PRECISION**	8
MPI COMPLEX	**MPI F COMPLEX**	2*4
MPI DOUBLE COMPLEX	**MPI F DOUBLE COMPLEX**	2*8
MPI_INTEGER1	MPI_F_INTEGER_1	1
MPI INTEGER2	MPI F INTEGER 2	2
MPI INTEGER4	MPI F INTEGER 4	4
MPI_INTEGER8	MPI_F_INTEGER_8	8
MPI INTEGER16	MPI F INTEGER 16	16
MPI REAL2	MPI F REAL 2	2
MPI REAL4	MPI F REAL 4	4
MPI REAL8	MPI F REAL 8	8
MPI REAL16	MPI F REAL 16	16
MPI COMPLEX4	MPI F COMPLEX 4	2*2
MPI COMPLEX8	MPI F COMPLEX 8	2*4
MPI COMPLEX16	MPI F COMPLEX 16	2*8
MPI COMPLEX32	MPI F COMPLEX 32	2*16

- The new type names in blue are fundamental data types
- Ext32 size in brackets are not defined in the current standard (and future)

Old and New Type Names (2/2)

C++ Kind		Ext32
Current Name	**Proposed New Name**	**Size**
MPI_CXX_BOOL	**MPI_CXX_BOOL**	1
MPI_CXX_FLOAT_COMPLEX	**MPI_CXX_FLOAT_COMPLEX**	2*4
MPI_CXX_DOUBLE_COMPLEX	**MPI_CXX_DOUBLE_COMPLEX**	2*8
MPI_CXX_LONG_DOUBLE_COMPLEX	**MPI_CXX_LONG_DOUBLE_COMPLEX**	2*16

MPI Kind		Ext32
Current Name	**Proposed New Name**	**Size**
MPI_BYTE	**MPI_MPI_BYTE**	1
MPI_PACKED	**MPI_MPI_PACKED**	(1)
MPI_AINT	**MPI_MPI_AINT**	8
MPI_COUNT	**MPI_MPI_COUNT**	8
MPI_OFFSET	**MPI_MPI_OFFSET**	8