

Backwards Compatibility Issues

Dave Solt/HP, Quincey Koziol/HDF Group, Jeff Squyres/Cisco

List of issues

- ▶ Source and binary forward / backward compatibility
- ▶ Language bindings
- ▶ MPI_Count



Forward or Backward Compatibility?

- ▶ We've really been talking about Forward
 - ▶ Taking existing MPI-1/2 codes and moving them to MPI-3
- ▶ Survey data points (with the usual disclaimers)
 - ▶ Only 16% think recompiling to MPI-3 is bad: **Good!**
 - ▶ Only 11% willing to change source code to MPI-3: **Boo!**
- ▶ Do we care:
 - ▶ Running MPI-3 codes in MPI-1/2 implementations
 - ▶ **I don't think so**
 - ▶ Integrating MPI-3 features in existing MPI-1/2 codes
 - ▶ **I DO think so**
 - ▶ Run MPI-1/2 codes without source code changes in MPI-3
 - ▶ **YES (per survey data)**



Forward Compatibility

- ▶ **Source code**

- ▶ Mix MPI-1/2 and MPI-3 features in the same code
- ▶ Using an MPI-3 function in a legacy code: this obviously works
- ▶ Better examples
 - ▶ Converting MPI-1/2 INTEGER Fortran handles to MPI-3 typed Fortran handles
 - ▶ Using an assertion that changes the behavior of legacy codes
- ▶ Consequences must be documented

- ▶ **Binary code**

- ▶ Intel has spoken strongly about forward binary compatibility
- ▶ We have not dived into these details yet



Language Bindings

- ▶ F03 bindings are coming
- ▶ So what to do about F77 and F90?
 - ▶ Do we also create F77 bindings for new MPI-3 functions?
 - ▶ **Yes**
 - ▶ Do we also create F90 bindings for new MPI-3 functions?
 - ▶ Hmm...
- ▶ What to do about C++?
 - ▶ Do we create C++ bindings for new MPI-3 functions?
 - ▶ **No** -- C++ has officially been deprecated
 - ▶ No new functions, types, constants, etc.
 - ▶ Survey data shows a surprisingly large C++ contingent
 - ▶ Don't know yet if the survey responses are accurate or not
 - ▶ But it is a different question whether to remove C++ or not

MPI_Count

- ▶ **First widely disruptive change since MPI-1**
 - ▶ ...there could (will?) be more over time
- ▶ **Encompasses several issues**
 - ▶ Widespread changes to existing MPI API functions
 - ▶ Potential for train wreck with legacy MPI applications
- ▶ **We could look at just addressing the MPI_Count issue**
 - ▶ Or we could try to address this more broadly



MPI_Count Motivation – initial requests

- ▶ HP received several requests to scale applications in a way that required sending messages greater than 2GB
 - ▶ Did not want to change their Fortran source
 - ▶ compiled with `-l8` (compiler flags to promote ALL integers to be 64-bytes)
 - ▶ Also wanted Scalapack that would scale similarly
 - ▶ requested that the MPI implementation “handle” this
- ▶ Although cores are getting more plentiful and relatively less powerful, I think we can expect hybrid programming models to increase the need for large off-node message counts
 - ▶ Meaning: we should do something about large counts



MPI_Count Motivation – HP’s partial response

▶ Fortran:

- ▶ Produced an `-l8` compatible mode
- ▶ Determined at run time if application was compiled with `-l8` (and/or `-R8`)
- ▶ Correctly interpret arguments as either 32 or 64-bit integers
- ▶ Dynamically defines MPI data types
- ▶ Internally MPI library always uses 64-bit integers for all count related arguments.

▶ C code:

- ▶ HP-MPI produced new C interfaces (`MPI_SendL`) to allow modification of Scalapack and other libraries.



A complete solution

- ▶ Do we want a more complete solution:
 - ▶ Works for all languages
 - ▶ Does not rely on compiler flags or non-standard APIs
- ▶ **Will** cause a train wreck for legacy codes
 - ▶ Maybe today, maybe 5 years from now
 - ▶ ...it's actually worse if it's 5 years from now!
- ▶ Framework for solutions
 - ▶ Do nothing
 - ▶ Extend
 - ▶ Replace
 - ▶ Abandon



#0 – Do Nothing (Fortran only solution)

▶ Proposal #1: Do nothing

- ▶ Let MPI's choose to support `-l8`
 - ▶ See Platform MPI for an example
- ▶ Fortran applications that need to use long counts much use `-l8`
- ▶ Pros:
 - ▶ No work for the Forum
- ▶ Cons:
 - ▶ Fortran only solution
 - ▶ Big hammer even for Fortran since it may double memory use



#1 – Extend: New API Functions

- ▶ Proposal #1: Create new interfaces that with 64-bit count
 - ▶ The count could simply be a long int, but preferably a new type
 - ▶ MPI_Count type to be defined by implementation
- ▶ Three options:
 1. MPI_Send → MPI_Send_count, or MPI_SendL
 2. MPI_Send → MPI_Send3
 3. MPI_Send → MPI_Send2 and MPI_Send3 + MPI_Send macro
 - ▶ Everyone hated #3 in Portland



#1 – Extend: New API Functions

- ▶ **Pros:**

- ▶ No backward compatibility issues
- ▶ Fixes the problem NOW

- ▶ **Cons:**

- ▶ Apps must be changed to access the new capability
- ▶ Explosion of interfaces



#2 – Replace: s/int/MPI_Count/ where relevant

- ▶ **Proposal #2: Change relevant functions to use MPI_Count**
 - ▶ Let implementations define the type of MPI_Count
 - ▶ C
 - ▶ Automatic type casting handles IN variables
 - ▶ No recompile needed if MPI_Count is an int and application passes int's.
 - ▶ C++ / F90 / F03
 - ▶ Function overloading handles it (...assuming we care about C++...)
 - ▶ No need to recompile unless you use MPI_Count.



#2 – Replace: s/int/MPI_Count/ where relevant

- ▶ Fortran 77
 - ▶ Must be “told” the size of the MPI_Count arguments
 - ▶ Need agreement between application and library.
 - ▶ Constants must be changed to declared parameters:

```
call MPI_Send(buf, 10, MPI_INTEGER, ...)
```

changes to

```
integer(kind=MPI_COUNT_KIND) ten  
parameter (ten = 10)  
call MPI_Send(buf, ten, MPI_INTEGER, ...)
```



#2 – Replace: s/int/MPI_Count/ where relevant

- ▶ **Pros:**

- ▶ Only impacts users who want 64-bit applications (vendors can ship 2 libraries)

- ▶ **Cons:**

- ▶ Existing C applications “work but are not MPI compliant”
- ▶ Fortran 77 application re-write is ugly.
- ▶ There are really 2 Fortran interfaces for every affected call:

- ▶ C:

- int MPI_Send(void* buf, MPI_Count count, MPI_Datatype datatype...)

- ▶ Fortran (if implementation is using 32-bit counts)

- MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)

- <type> BUF(*)

- INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR

- ▶ Fortran (if implementation is using 64-bit counts)

- MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)

- <type> BUF(*)

- INTEGER DATATYPE, DEST, TAG, COMM, IERROR

- ▶ INTEGER(KIND=MPI_COUNT_KIND) COUNT

#3 – Abandon: Leave Fortran 77 as INTEGER

- ▶ #3/Abandon must be mixed with #1/Extend or #2/Replace
- ▶ Much of the problem with Proposal #2/Extend is the “ugliness” of the Fortran 77 solution
 - ▶ ...so leave Fortran 77 only supporting INTEGER
- ▶ Move C, F03 to new MPI_Count type
 - ▶ Either (Abandon+Extend) or (Abandon+Replace)
- ▶ The road to large counts in Fortran is via F03
 - ▶ Still need to decide what to do with F90



Questions

- ▶ Fortran 77 seems to be “a big problem”
 - ▶ Can we just define the way to large counts as F03?
- ▶ Will using MPI_Count for C/C++ make users fearful?
 - ▶ Because they have to change their codes
- ▶ How would vendors feel about shipping 2 libraries?
 1. C MPI_Count == int
 2. C MPI_Count == uint64_t

