

# CS267 MPI

**William Saphir**

Lawrence Berkeley National Laboratory  
NERSC Division  
Phone: 510-486-4373  
wcsaphir@lbl.gov



## A note on software longevity

- Previous CS267/MPI class in 2000
  - Slides written with Framemaker
  - Framemaker has been around since 1986 and is still widely used
  - But I don't own a copy (hence these ppt additions)
- MPI has been around since 1994. Happy 10<sup>th</sup> anniversary!
  - MPI programs written in 1994 run fine today
  - MPI is available (for free) on all parallel computing platforms
  - The 2000 CS267 slides are still valid today (including URLs) except for machine examples (hardware changes, but programs are portable)



## What is MPI?

---

### Message Passing Interface:

#### Software standard for distributed memory parallel programs

- Library of routines: communication, utilities
- Parallel application model
  - MPMD
  - Explicit distributed memory
  - Emphasis on tightly coupled applications
  - Does not address runtime environment
- MPI is a specification, not a specific implementation

**MPI Forum:** voluntary organization representing industry, government labs, academia



## Other programming models

---

- OpenMP
  - Automatic parallelization (Fortran and C) on shared memory machines.
  - Sometimes used with MPI on CLUMPS.
- Threads
  - Explicit parallelization for shared memory machines.
  - Full address space visible to each thread
- High Performance Fortran (and other mostly dead languages)
  - Automatic parallelization for distributed memory machines
  - Single thread of control
- Co-Array Fortran/UPC
  - SPMD distributed shared memory model – but cache coherency not required
  - Communication through pointers to shared data
- One-sided communication (Get/Put)



## Update to 2000 slides

---

- Many features of MPI-1 are very rarely used (I'll skip these sections)
  - Communicators for anything other than coupled codes
  - Intercommunicators
  - Different send modes, pack/unpack
  - Datatypes (often cause more problems than they solve)
- Most features of MPI-2 are very rarely used
  - Dynamic process management
  - One-sided communication
  - C++ language bindings and I/O are most commonly used features of MPI-2



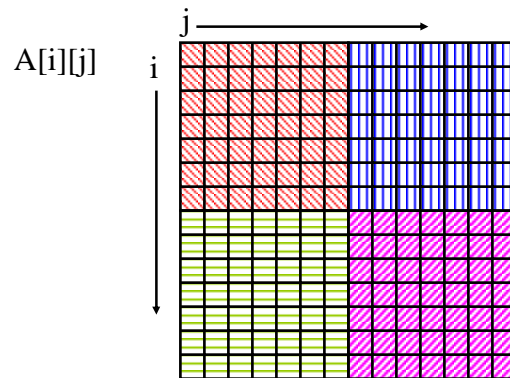
## Random MPI-2 notes (if time)

---



## Motivation for MPI-2 I/O

- Consider 2D array, row-major order, to be stored in single file, partitioned among 4 processors
- **Each processor writes many small non-contiguous blocks**



## C++

- **MPI needs a C++ interface**
  - Key question: closely related to C interface or full-blown OO?
  - Decision: C++ interface is close to C/Fortran interface.
- General principles:
  - MPI handles (`MPI_Comm`, etc.) become C++ **objects**.
  - MPI functions become **methods** on C++ classes.
  - Do what C++ programmers expect where possible but
  - Don't stray too far from MPI principles.



## More C++ principles

---

- Shallow copies
- Constructors create `MPI_XXX_NULL`. Destructors do not free.
  - User must generally use `create` and `free`
  - Reasons
    - Variables going in and out of scope could be collective operations
    - Automatic destruction violates shallow copy semantics



## C++ return values

---

- C routines are functions returning a rarely used error code
- Fortran routines return error code in argument
- C++ routines use C++ exception mechanism by default and don't return an error code
  - Frees up return value to return a value

```
request = comm.Isend(buffer, count, type, dest, tag);
mytype = MPI::DOUBLE.Create_contiguous(5);
```
- New `MPI::Exception` class for exceptions



## Which object?

---

- Choose "obvious" object
  - `MPI_COMM_RANK(comm, rank)`
    - `int Comm::Get_rank()`
  - `MPI_ISEND(buff, count, type, dest, tag, comm, request)`
    - `type.Isend(...)?`
    - `request.Isend(...)?`
    - `comm.Isend(...)?`
- Make static when no possible or clear object
  - `static int Request::Waitany(count, array_of_requests);`
  - `static Group Group::Difference(g1, g2);`



## C++ function names

---

- MPI-1 names were inconsistent
- MPI-2 names are much more consistent
  - `MPI_Class_action_subset`      `Class:Action_subset`
  - `MPI_FILE_OPEN`                `File::Open`
  - `MPI_COMM_SPAWN`               `Comm::Spawn`
- Standardized actions
  - `Create/Is/Get/Set`
- C++ names for MPI-1 functions use the new consistent rules
  - `MPI_COMM_RANK`                `Comm::Get_rank`
  - `MPI_INITIALIZED`               `MPI::Is_initialized`
  - `MPI_TOPO_TEST`                 `Comm::Get_topo`
  - `MPI_TYPE_CONTIGUOUS`         `Datatype::Create_contiguous`



## Class structure

---

```
namespace MPI {
  class Comm {...};
  class Intracomm : public Comm {...};
  ...
  class Datatype {...};
  class Exception {...}; /* new */
  class Request {...};
  class Prequest : public Request {...};
  class Status {...};
  class File {...}; /* MPI-2 */
  class Info {...}; /* MPI-2 */
  class Win {...}; /* MPI-2 */
};
```



## Comments on class structure

---

- **Namespace? What's that?**
  - Part of ANSI standard
  - May be implemented as non-instantiable class instead
- **Derived classes?**
  - Obvious", but leads to some unexpected strangeness
  - `MPI::Comm` is an abstract base class (non-instantiable) for `MPI::Intercomm`, `MPI::Intracomm`, etc.
  - This enforces typing in the right way
  - `Dup()` is defined on derived classes only
  - `clone()` is defined on `Comm` and returns a reference to new object



## Other C++ features

---

- C and C++ can coexist in same program
- C++ uses <mpi.h> also
- Automatic conversion between C and C++ objects
- C++ bindings can be easily layered on top of C bindings
- Assignment operator (=) overloaded to do shallow copies
- Almost everything is passed by reference



## Fortran 90 support

---

**Fortran 90 has many "modern" features.**

- User-defined types
- Function overloading
- Parameterized types
- Mechanisms for strict type checking (interface blocks)
- First class arrays

**Can MPI take advantage of these?** Mostly **no**.

Opinion: F90 tries to hide machine details from user to allow compilers to optimize. This was a mistake from the point of view of MPI. To get an MPI that works well with Fortran 90, need very close integration with compiler.



## MPI approach to F90

---

- Fortran 90 bindings based on F77 bindings
  - Validate current practice
- Carefully explain to users what can go wrong
  - Most of the time things work. Occasionally they don't
- Make use of F90 features where possible (rarely)
- Support F90 features where possible



## Fortran 90 vs. MPI

---

Fortran 90 and MPI are not completely compatible

- **MPI has choice arguments**
  - F90 argument checking is strict
  - Derived types require argument checking
- **MPI assumes flat address space**
  - F90 does not require sequence association



## Fortran 90 vs. MPI (2)

---

- **MPI needs to modify data asynchronously**
  - F90 may make temporary copies of arrays
- **MPI special constants are not special values**
  - Special constants cannot be defined within F90

**These issues are discussed in detail in the standard.**

