MPItrampoline: Choose your MPI implementation at run time

Erik Schnetter, Perimeter Institute

EasyBuild tech talks V

2021-12-20

Please interrupt me for questions at any time

Part 1: Why MPItrampoline?

What is MPI (“Message Passing Interface”)?

- MPI is a **source-level standard** for distributed computing
  - Basically send/receive messages
  - mpi-forum.org

- Many implementations:
  - MPICH (open source)
  - OpenMPI (open source)

  - Cray MPI
  - IBM Spectrum MPI
  - Intel MPI
  - Microsoft MPI
  - …

Using MPI

- Run many copies of a program on different nodes
- Send/receive messages
- Collective operations: barrier, broadcast, reduce, …
- Read/write from/to memory of another process
- Parallel I/O

- CSP, “Communicating Sequential Processes”
    - Very 1990s, somewhat object-oriented programming model, with global state
- Portable and ubiquitous in HPC (High-Performance Computing)

High-Performance Computing

- Performance has two sides:
  - Bandwidth (bytes per second)
  - Latency (minimum wait time)
- Bandwidth scales easily (cloud computing). Latency doesn’t.

- 10 Gbit Ethernet (TCP): 10…100 µs
- InfiniBand: 1…10 µs

- MPI libraries offers efficient access to efficient network interfaces

How to install software

- Unix philosophy (outdated): build from source
  - Slow, fragile, requires substantial expertise, basically not reproducible
- Real world: download binaries
  - Red Hat, Ubuntu, Nix, Anaconda (Python), Yggdrasil (Julia)
  - Docker images
    - VM (Virtual Machine) images
      - disk images
- What about MPI?

Everything Unravels

- MPI standard is a source-level standard
- MPI libraries cannot be installed as binaries
  - MPI libraries access system hardware; need to be linked against system-specific libraries, configured for particular hardware (like a device driver)
- ⟶ All software that uses MPI must be installed from source everywhere
- ⟶ Decades of package management advances down the drain
- ⟶ EasyBuild, Spack cannot offer binaries on HPC systems
- ⟶ HPC suffers from very high **incidental complexity**

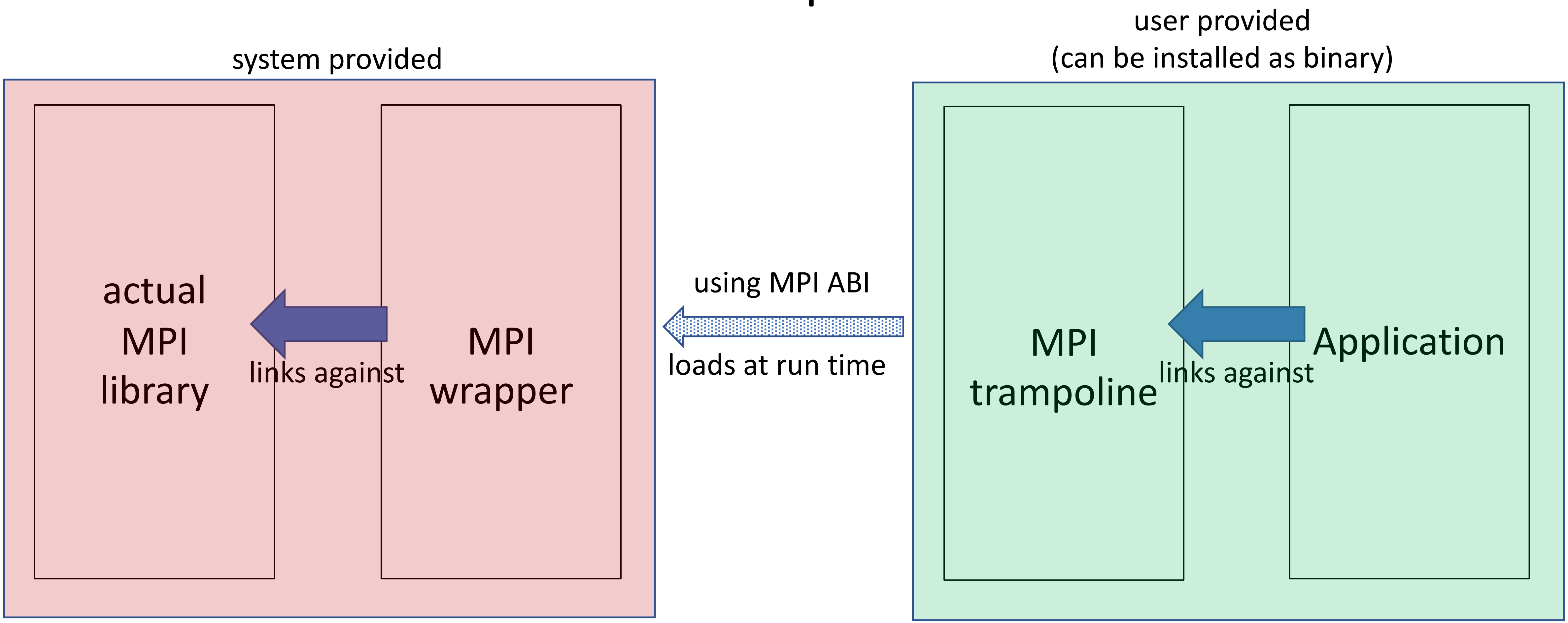
MPI ABI

- MPI ABI: Make different MPI implementations binary compatible

- Approach:
  - The system-specific parts of MPI are installed by an administrator, similar to a device driver
  - The application links to a generic MPI interface
  - They interact via a well-defined ABI (Application Binary Interface)

- Works for libc and OS kernel, for CUDA and CUDA driver, etc.

MPItrampoline

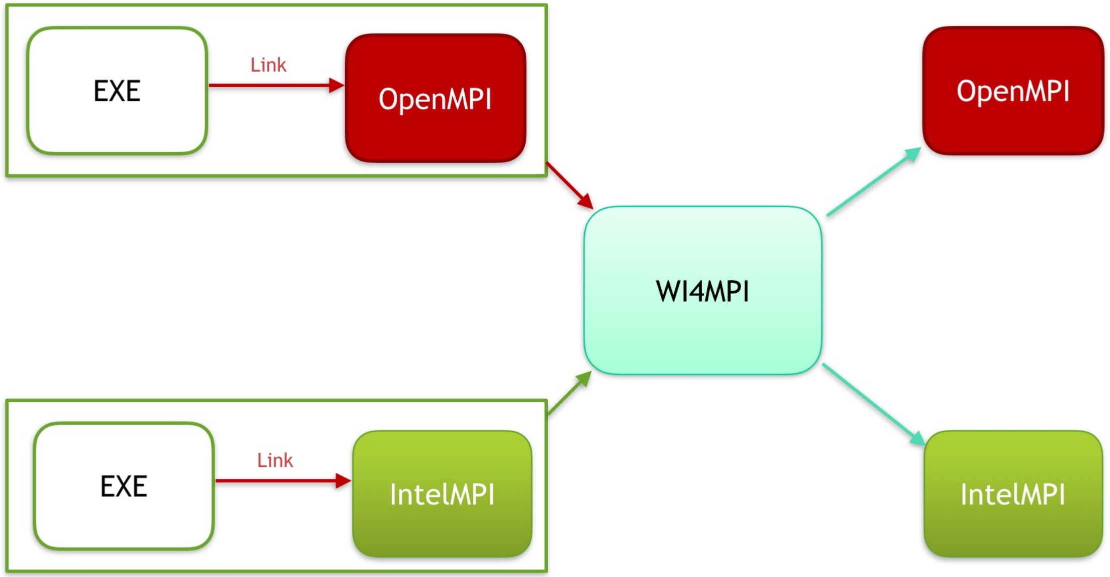


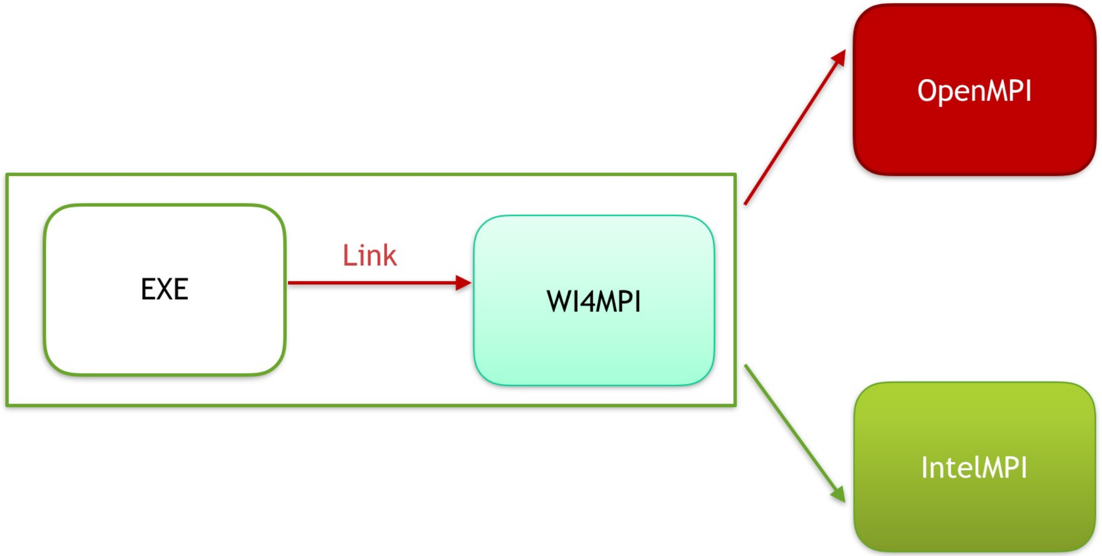
Provides compatibility layer around real MPI library

Looks like a real MPI library, but doesn’t do any work

WI4MPI (Wrapper Interface For MPI)

https://github.com/cea-hpc/wi4mpi





Using MPItrampoline

- Build application against MPItrampline as MPI library
  - Can be shipped as binary
- On target system, build MPIwrapper for every MPI library there
  - Ideally done by system administrator or experienced user
- At run time, set environment variable MPITRAMPOLINE_LIB to point to desired MPIwrapper

- Ready for production use, but no big user yet ("beta")
- Currently integrating MPItrampoline with Julia MPI bindings
- See github.com/eschnett/MPItrampoline

Julia MPI bindings

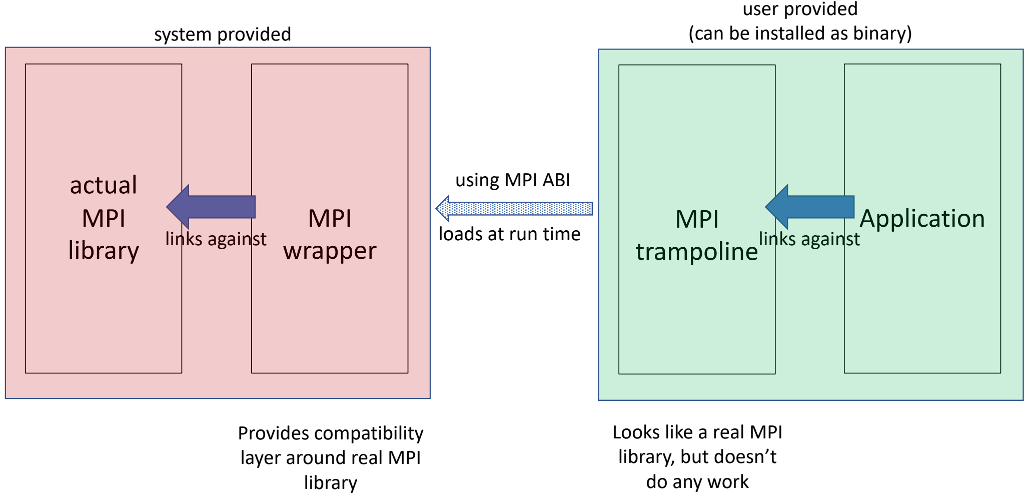
- Julia is a modern programming language, well suited for numerical applications
- Julia’s package manager (Yggdrasil) ships binaries for external dependencies (FFTW, HDF5, PETSc)
- **Important:** all code (both Julia code and all external dependencies) needs to use the **same** MPI implementation
  - Usually, caller passes MPI handles to libraries
- Goal: Use MPItrampoline
  - and a fallback MPICH for non-HPC systems

Part 2: Under the Hood

MPI ABI

- The MPI standard defines:
- Compile-time constants
  - #define MPI_MAX_ERROR_STRING 1024
- Types
  - typedef uintptr_t MPI_Comm;
- Load-time constants
  - MPI_Comm MPI_COMM_WORLD;
- Functions
  - int MPI_Comm_size(MPI_Comm comm, int *size);
- Callbacks
  - void (*)(void *invec, void *inoutvec, int *len, MPI_Datatype *datatype);

MPItrampoline

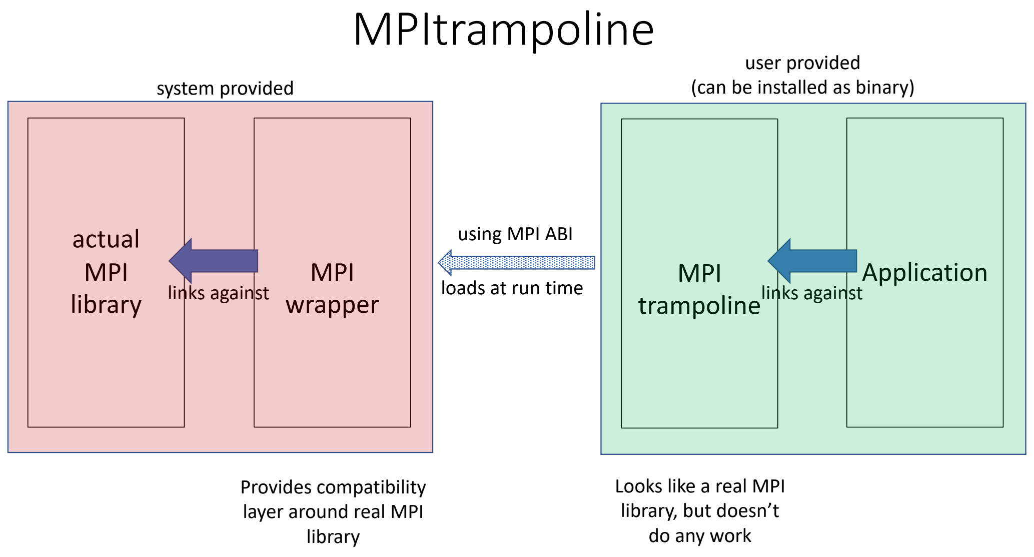


- MPItrampoline *queries* MPIwrapper about values of constants and pointers to functions
- MPIwrapper *translates* MPI ABI to actual MPI library

Startup

- At run time, when MPItrampoline is loaded, it needs to decide which MPIwrapper to load

- Environment variable MPITRAMPOLINE_LIB
- Can set a global variable
- Can hard-code a default value at build time
- Otherwise, cannot call MPI functions

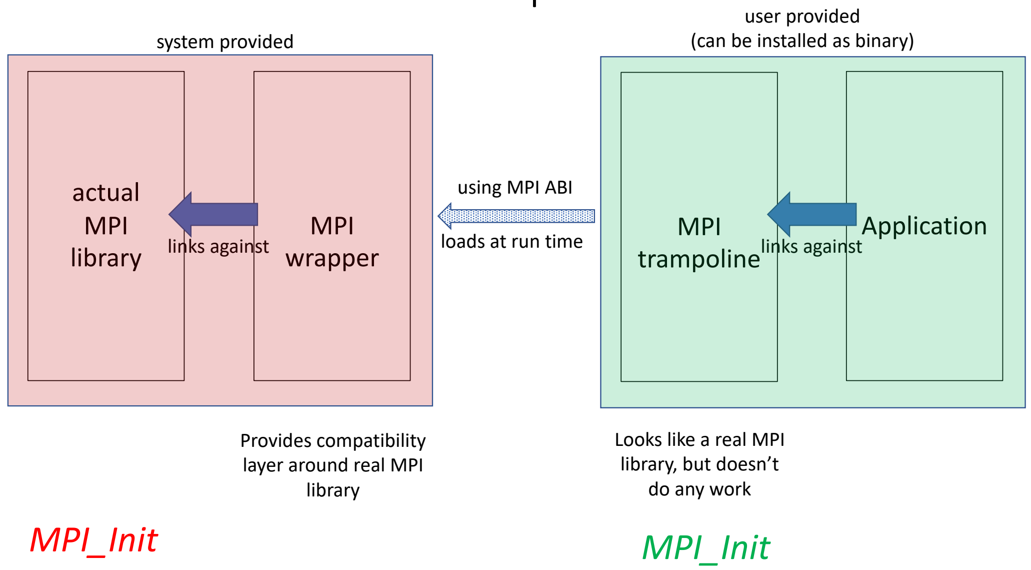


Shared Libraries and Plugins

- There are **two** functions *MPI_Init:*
  - In MPItrampoline
  - In the actual MPI library
- They are incompatible!

- Linker namespaces to the rescue (*dlmopen*)
  - Except they don't work in practice (on HPC systems)
  - And they aren't available on macOS, BSD
- Symbol Interposition
  - RTLD_DEEPBIND (Linux, (BSD?))
  - Two-level namespaces (macOS)

MPItrampoline



Efficiency

- MPItrampoline’s interface layer is very efficient:
    - MPI handles have 64 bits, might require conversion from/to 32 bits
    - Arrays of MPI handles might require copying (but no allocations)
    - MPI status has 3 extra fields (could be optimized)
    - Callback functions need to be wrapped

- MPI constants have the same values
- MPI functions are called via function pointers
- Nothing expensive happens inside MPItrampoline

- Sorry, no benchmark results yet

Current State

- C and Fortran 77 bindings complete, except for some callbacks
- ADIOS2, AMReX, Boost(*), FFTW, HDF5, PETSc, and many others build
- OpenMPI test suite passes (*)
- I am using it in production for the Einstein Toolkit einsteintoolkit.org, via Spack
- Working on using it for Julia’s external packages (with MPICH fallback)

- Let’s drag HPC package management kicking and screaming into the Century of the Fruitbat

Perimeter Institute for Theoretical Physics

Waterloo, Ontario, Canada