MPI.jl 1.0

(almost)

Simon Byrne
Lead Software Engineer, CliMA project
California Institute of Technology
clima.caltech.edu

MPI.jl

Julia bindings for the Message Passing Interface

- Started by Lucas Wilcox in July 2012, since had 60 contributors
- Inspired by mpi4py, similar to C API + some Julia niceties
  - Handles buffer length and datatypes
  - Generates custom datatypes and operators
  - Julia exceptions instead of error codes
- Exposes most commonly used functions, but still only a small part
- For more info, see JuliaCon 2020 talk + proceedings paper

Recent changes

What has changed since 2020?

- Consistency and interface cleanup
  - Make use of mutation exclamation suffix consistent
  - Add keyword args for many ops (tag, rank, root)
- More coverage of API (PRs welcome!)
- Bugfixes / workarounds for different implementations and hardware
- Better docs and examples (more welcome)
- Imminent release of 0.20, which will (hopefully) be pre-release for 1.0

GPU support

- Allows direct MPI communication between GPU devices (without buffering through main memory)
  - Uses same functions, passing device pointers
  - Requires GPU-aware implementation
- MPI.jl supports
  - CUDA.jl `CuArrays` (added by Seyoon Ko)
  - AMDGPU.jl `ROCArrays` (added by Ludovic Räss)
- Rough edges
  - No standardized mechanism to query features
  - Not all functions are GPU-supported
  - CUDA.jl uses per-thread streams: need to be manually synchronized before communication.
  - Reusing same MPI functions has technical limitations

MPIPreferences.jl

Select your MPI binary

- Every cluster has their own MPI implementation (or more than one!)
  - MPI ABI (types + constants) are *not* standardized
- Previously we used `Pkg.build` + environment variables
  - Difficult to switch between implementations (requires full rebuild)
  - Tied to MPI.jl version: an update could break everything
- MPIPreferences.jl is a lightweight package for selecting MPI implementations, uses new Preferences.jl functionality
  - Pre-built jll (MPICH, Open MPI, Microsoft MPI, MPItrampoline)
  - System MPI, with ABI detection
  - Allows for system-wide settings by modifying JULIA_LOAD_PATH
  - Plays nice with precompile cache
- Downstream binaries are a work-in-progress (insert grumbling about HDF5)

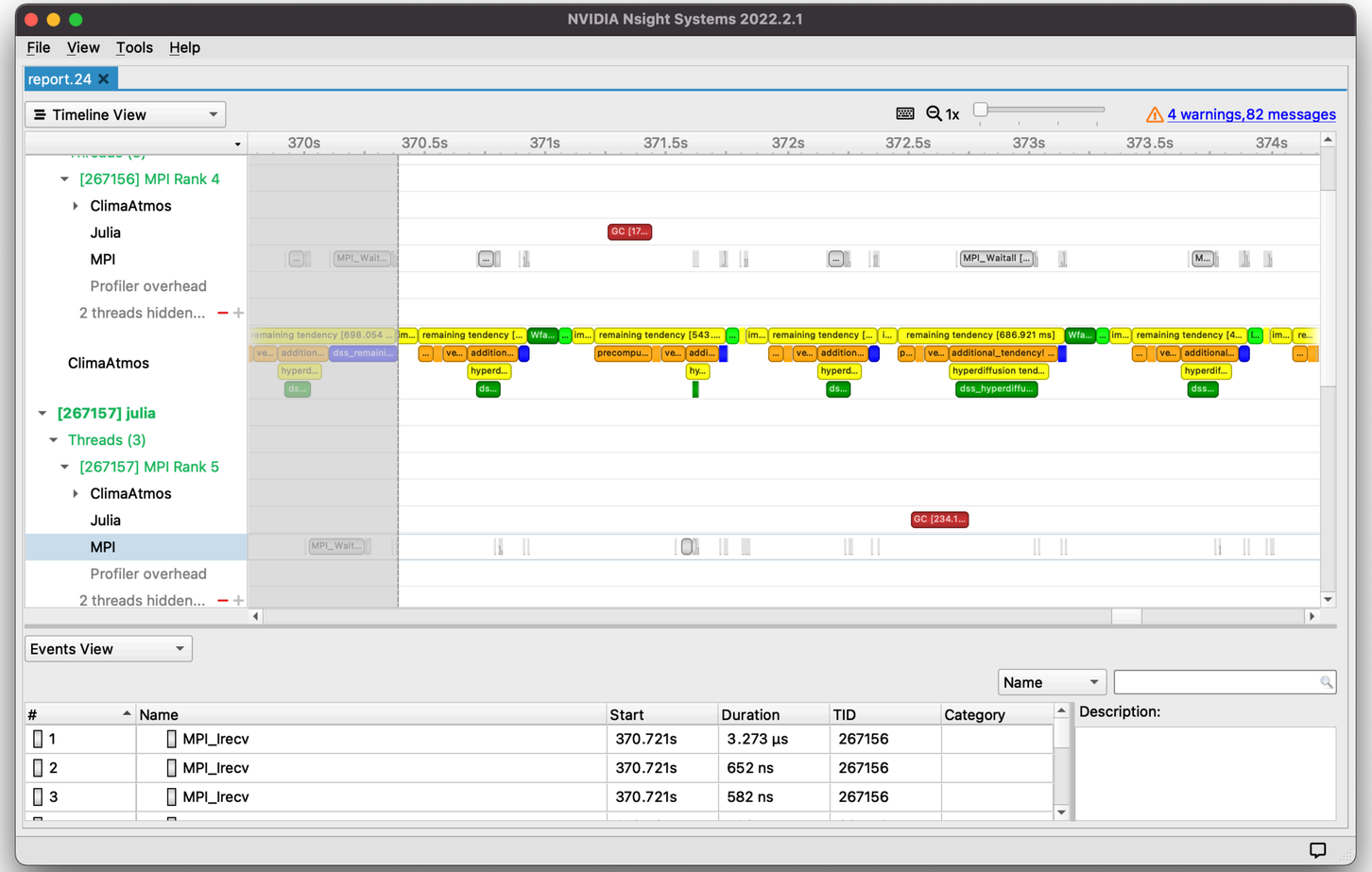
```
julia> using MPIPreferences

julia> MPIPreferences.use_system_binary()
┌ Info: MPI implementation
│   libmpi = "libmpi"
│   version_string = "Open MPI v4.1.4, package
│   impl = "OpenMPI"
│   version = v"4.1.4"
└   abi = "OpenMPI"
```

Use with MPI profilers

- MPI.jl uses global ccalls:

```
ccall(:MPI_XXX, Cint, …)
```

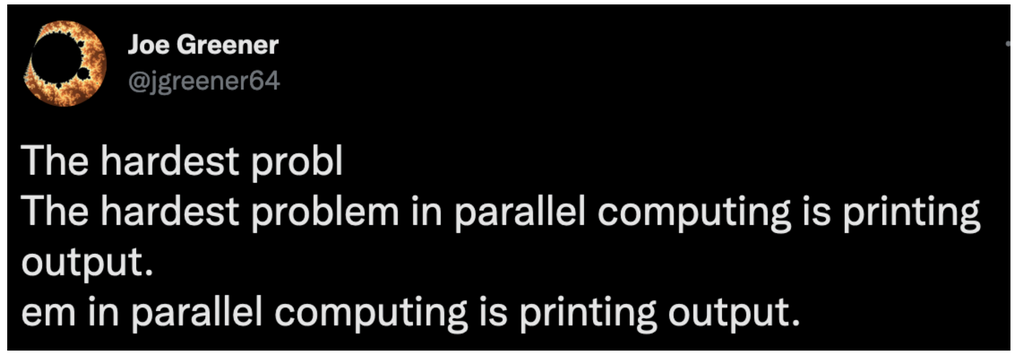
- Enables profilers & debuggers which use LD_PRELOAD to intercept MPI calls
- Nvidia Nsight Systems is free
  - Works without a GPU
  - NVTX.jl package for custom instrumentation, hooks for the Julia GC



Gripes

What I wish could be better

Printing



- MPI launchers merge stdout/stderr from each process into a single stream
  - Lines frequently get intermixed, making it difficult to parse output
  - Sometimes launchers have options to prefix printing with MPI rank
  - Each MPI implementation does this differently (of course)
  - No support from MPI IO interface
- Stack traces are especially bad
  - Julia seems to be worse than Python (mpi4py): perhaps Python buffers more?

Lack of interactivity

- REPL is one of Julia's nicest features
  - Can't run REPL from inside MPI launcher (buffering is incompatible)
  - Singleton MPI (i.e. running MPI outside launcher) + `MPI_Comm_connect` / `MPI_Comm_spawn` doesn't work in any MPI implementation
- Workarounds
  - **tmpi** (https://github.com/Azrael3000/tmpi): MPI launches tmux sessions, multiplexing the keyboard to all processes.
  - **MPIClusterManagers.jl**: Distributed.jl workers started via MPI launcher. Main process is not part of MPI session.

Integration with Julia tasks

`Base.wait` vs `MPI.Wait`

- Julia `wait(task)` will allow other scheduled tasks to run until `task` is completed.
- `MPI.Wait(request)` will wait in the ccall, blocking any other Julia tasks from running.
- MPI doesn't provide any callback or notification mechanism
- Alternatives
  - Use a spinlock

```
while !MPI.Test(request)
  yield()
end
```

  - Dedicate a hardware thread

```
Threads.@spawn MPI.Wait(request)
```