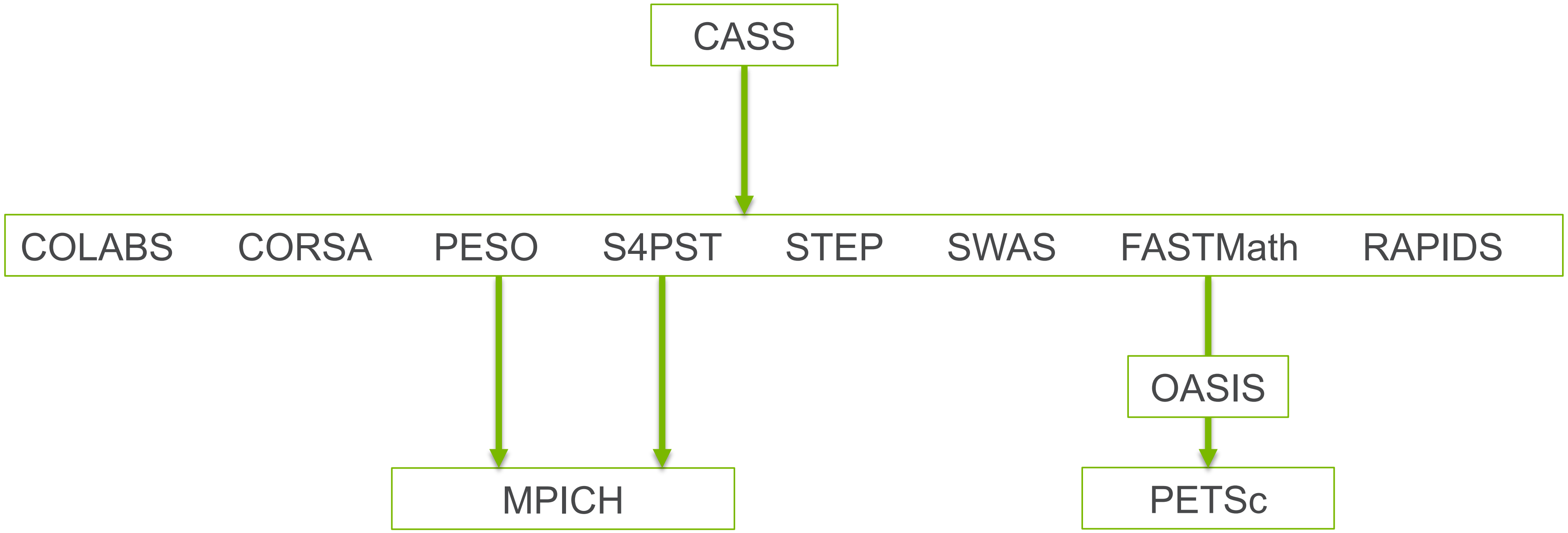
MPICH BoF at Consortium for the Advancement of Scientific Software (CASS) Community BoF Days, June 11-13, 2024, Virtual

THE COLLABORATION BETWEEN PETSC AND MPICH

Junchao Zhang
(jczhang@anl.gov)

Mathematics and Computer Science Division
Argonne National Laboratory
June. 12, 2024

PETSc and MPICH under CASS


$$\begin{tikzcd}
 & & & \text{CASS} \arrow[d] & & & & \\
\text{COLABS} & \text{CORSA} & \text{PESO} \arrow[dd] & \text{S4PST} \arrow[dd] & \text{STEP} & \text{SWAS} & \text{FASTMath} \arrow[d, dash] & \text{RAPIDS} \\
 & & & & & & \text{OASIS} \arrow[d] & \\
 & & \text{MPICH} & \text{MPICH} & & & \text{PETSc} &
\end{tikzcd}$$

MPICH and PETSc at Argonne

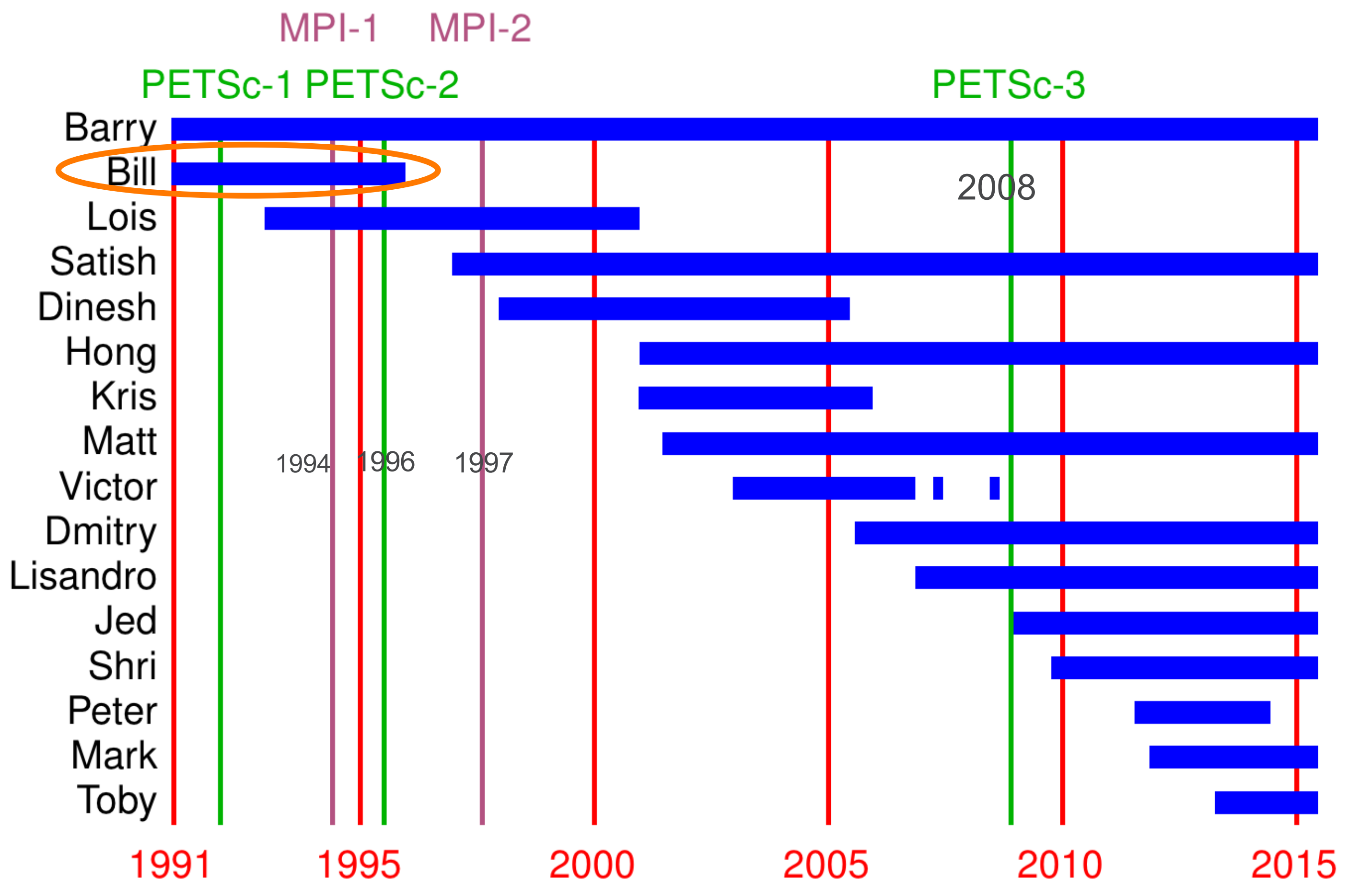
- MPICH: the most widely used MPI implementation and is the implementation of choice for the world's fastest machines

- PETSc: a scalable numerical library for linear and non-linear solvers and more
  - Has C, Fortran, Python, Rust bindings
  - Runs on Linux, Mac and Windows
  - Widely used in academia and industry in dozens of disciplines

Outline

- The PETSc/MPICH collaboration in the MPI-1.0~2.0 era and PETSc’s adoption of new MPI features
- The PETSc/MPICH collaboration in the MPI-2.0~4.0 era and PETSc’s adoption of new MPI features
- The PETSc/MPICH collaboration in recent years
- Conclusion

MPI-1.0~2.0 era



- MPICH was originally developed during the MPI standards process starting in 1992 to provide feedback to the MPI Forum on implementation and usability issues.
- Bill Gropp was deeply involved in both projects

PETSc’s most successful use of MPI-1.0 features

- MPI communicators and attributes
  - PETSc inner communicator to separate PETSc library messages from callers
  - Sub-communicator in multigrid solvers
- Persistent MPI_Send/Recv
  - Repeated, split-phased sparse neighborhood communication in Krylov solvers
- Various MPI collectives
  - MPI_Allreduce() for VecNorm(); two-sided discovery from one-sided
- MPI datatypes
  - Note derived data types are less used, since we mainly deal with sparse data

"MPI changed everything, by providing an extensive API for message passing and collectives that allowed portable distributed memory scientific libraries to no longer need to be programmed to the lowest common denominator of message passing systems. … The MPI communicator concept made distributed parallel scientific libraries practical in two ways, it eliminated the tag collision problem and (by the use of subcommunicators) allowed applications to simply utilize scientific libraries to perform needed computations on subsets of processes, for example with 'divide and conquer' algorithms."

-- Barry Smith

https://www.hpcwire.com/2017/05/01/mpi-25-years-old/

PETSc’s adoption of new MPI-2.0 features

MPI-IO

✅ MPI Fortran-90 binding

❌ MPI one-sided (RMA) & dynamic process

– Not even tried in the next decade

❌ MPI + multithreading

– PETSc added support for both OpenMP and Pthreads and found the code was never faster than pure MPI and cumbersome to use hence we have removed it

"The PETSc team has no problems with proposals to replace the pure MPI programming model with a different programming model but only with an alternative that is demonstrably better, …

At least for the PETSc package, the concept of being thread-safe is not simple. It has major ramifications about its performance and how it would be used; it is not a simple matter of throwing a few locks around and then everything is honky-dory."

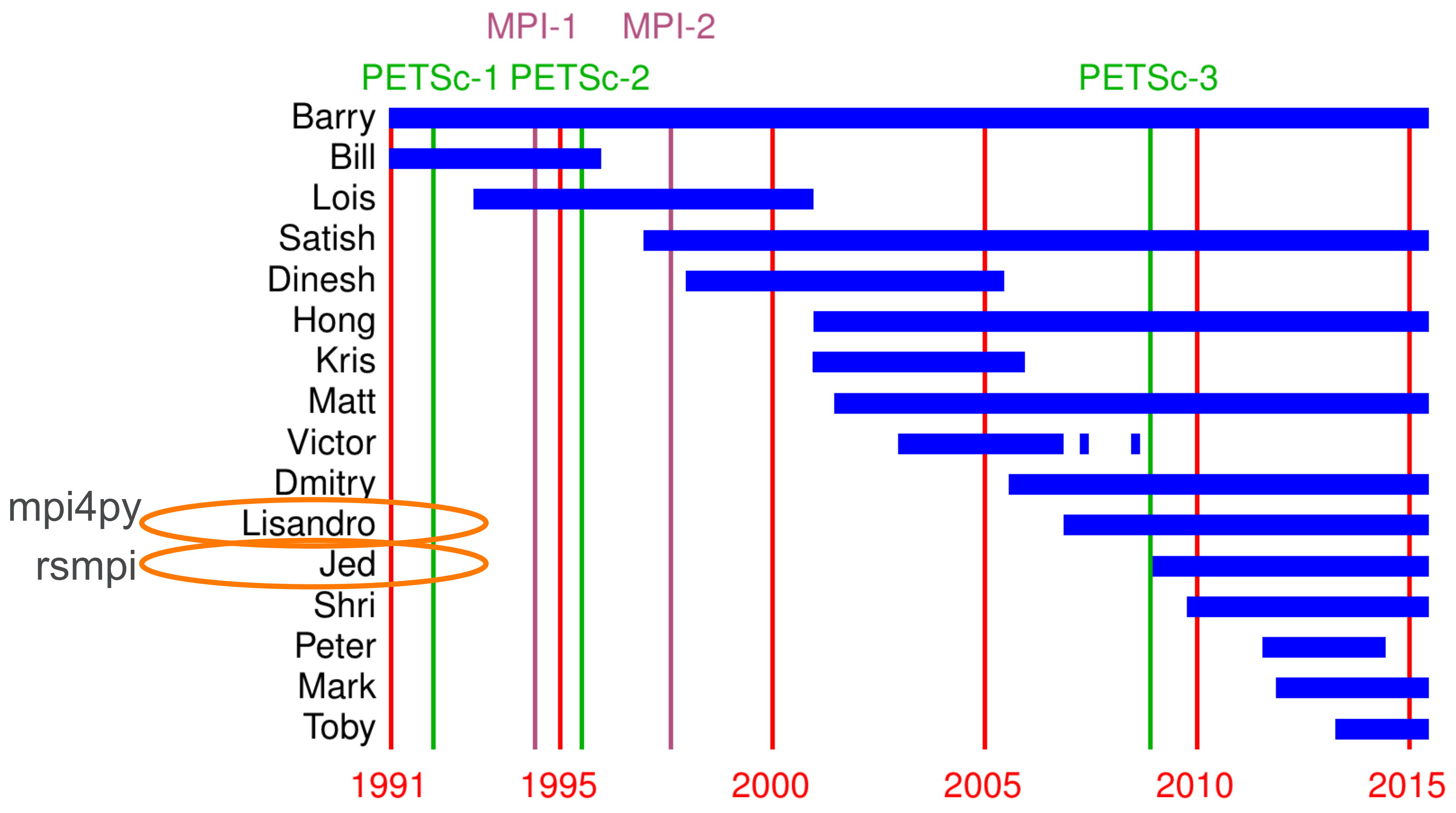
-- Barry Smith

https://www.mcs.anl.gov/petsc/petsc-3.15/docs/miscellaneous/threads.html

MPI-2.0~4.0 era

- As both projects became mature, the close collaboration was almost lost
- PETSc occasionally tried new features introduced in the MPI standard
  - MPI-3.0 process-shared memory to improve intra-node communication
    - Not easier than two-sided for sparse-neighborhood & no obvious performance benefit
  - MPI-3.0 revised one-sided in PetscSF implementation
    - `-sf_type window -sf_window_flavor <create|dynamic| allocate> -sf_window_sync <fence|active|lock>`
    - Yet to show an advantage over two-sided
  - MPI (persistent) neighborhood collectives
    - `-sf_type neighbor –sf_neighbor_persistent <bool>`
  - MPI_Iallreduce() in pipelined CG solver (`–ksp_type pipecg`)
  - MPI_Ibarrier/Iprobe() with `–build_twosided ibarrier*`
    - The ibarrer alg. [hoefler2010] performs better at large scale than the allreduce alg.
    - Less reliable than allreduce, frequently run into errors with Intel MPI
  - MPI large count (`--with-64-bit-indices`)

PETSc developers' contribution to the MPI community -- MPI for Python and Rust maintainers



The enhanced PETSc/MPICH collaboration in recent years

- PETSc CI job coverage with MPICH on GPUs
  - PETSc CI helped MPICH identify its excessive GPU memory usage
  - MPICH helped PETSc discovery a serious GPU stream sync bug
- PETSc is experimenting with the MPICH GPU stream extension
  - `-sf_use_gpu_aware_mpi <bool>` (not steam-aware)
  - `-sf_use_stream_aware_mpi <bool>` (experimental)
- PETSc is experimenting with the MPI-5.0 ABI implemented in MPICH
  - PETSc users might mess up the PETSc build time MPI (e.g., OpenMPI) with user code build time MPI (e.g., MPICH)
  - It is helpful to unify the MPI ABI
- PETSc inspired the MPIX_THREADCOMM extension in MPICH

The “PETSc + OpenMP” dilemma

- PETSc doesn’t support OpenMP because of the complexity and bad performance
- Some OpenMP-only codes want to call PETSc to leverage its tons of solvers
  - Also want PETSc to be run in parallel to make use of the CPU cores

The MPICH MPIX_Threadcomm Solution

```
Mat       A;
Vec       x, b;
int       nthreads = 4;
MPI_Comm comm;
PetscInitialize(&argc, &argv, NULL, NULL);
// user code building A, x, b etc
…
MPIX_Threadcomm_init(MPI_COMM_WORLD, nthreads, &comm);
#pragma omp parallel num_threads(nthreads)
{   Mat A2;
   Vec x2, b2;
   KSP ksp;
   MPIX_Threadcomm_start(comm); // comm's size is 4
   MatCreate(comm, &A2);
   MatCreateVecs(A2, &x2, &b2);
   // Assemble A2, b2 from the shared A, b
   KSPSolve(ksp, b2, x2);
   // Transfer the solution x2 to x
   MatDestroy(&A2);
   MPIX_Threadcomm_finish(comm)
 }
MPIX_Threadcomm_free(&comm);
PetscFinalize();
```

- Run the test as a regular OMP code:
  `OMP_NUM_THREADS=8 ./test -args`

- User's sequential code (might use OpenMP)
- PETSc is initialized on a single process
- Build sequential petsc objects such as matrices and vectors

- Build parallel petsc objects on the threadcomm *comm*
- *Somehow* transfer data from the shared sequential A, b to parallel A2, b2
- Other parts of the petsc code work as if they were run by `mpiexec -n 4 ./test`
- Caveats: petsc needs to be thread safe, e.g., in logging
- Future work: provide a new preconditioner type `PCOMP` to wrap around this stuff

Summary: MPI & MPICH’s use in PETSc

- In 2024, PETSc can still build with MPI-2.1 without (performance) issues!!
- PETSc users do not need MPI if they only use PETSc sequentially
  - `./configure --with-mpi=0`
  - petsc will use its fake single-process MPI (mpiuni) impl. to provide MPI APIs
  - Maybe MPICH could take it over as others also like it (?)
- MPICH is recommended by PETSc for users needing valgrind
- The *latest* MPICH can be downloaded and installed by PETSc (many users use that!)
  - `./configure --with-cc=gcc --with-cxx=g++ --with-cuda --download-mpich –download-mpich-device=<ch3:nemsis|..>`
- PETSc has 8000+ tests and 70+ CI jobs with many using MPICH for testing

Conclusion

- PETSc is an excellent testbed and a real world inspiring example for MPI and MPICH research
- The closer the two projects collaborate, the better they can serve their users