Understanding MPI on Cray XC30

Basic information about Cray's MPI implementation

New features in MPI 3.0

- **Improved collective communication**
  - Scalable sparse collectives on process topologies
    - MPI_(I)Neighbor_allgather(v) & alltoall(v/w)
  - Nonblocking collectives
    - MPI_Ibcast, MPI_Iscatter,...
    - Analogous to non-blocking point-to-point operations
- **Rewritten one-sided communication**
- **Support for SMP nodes**
  - Topology-aware communicators (MPI_Comm_split_type)
  - Shared-memory RMA window (MPI_Win_allocate_shared)
- **Improved Fortran bindings**
  - ”Use mpi_f08”
  - Support for subarrays
  - Some optional arguments (ierror...)
  - CRAY MPI will support this later this year

Status of MPI-3 implementation

- **The MPT 6.0 and later versions are MPI-3 compliant**
  - E.g., one-sided communication finally truly one-sided
  - Currently missing is
- **High-performance versions of non-blocking communication are already now implemented**
  - When using them, you should do

```
export MPICH_NEMESIS_ASYNC_PROGRESS=1
export MPICH_MAX_THREAD_SAFETY=multiple
export MPICH_GNI_USE_UNASSIGNED_CPUS=enabled
```

  - See the MPI 3.0 standard (e.g. at www.mpi-forum.org) for the exact bindings

Newer changes to the Cray MPI implementation

- **MPT 7.0 ABI compatibility change.**
  - An ABI change in the MPT 7.0 release requires all MPI user code compiled with MPT 6.x to be recompiled. Scientific libraries, Performance Tools, Cray Compiling Environment and third party libraries compatible with the new ABI are included in the June and later PE release.
  - As there are several libs dependent on mpi Cray has a module taking care of the dependencies. Simply load module cray-mpich-compat/v7 or cray-mpich-compat/v6 to get the correct libs
- **Cray MPI will always use huge pages**
  - This is independent on loading craype-hugepagesX and is controlled by MPICH_GNI_HUGEPAGE_SIZE (default is 2M)
- **MPI Fine-grained multi-threading support has been added**

Newer changes to the Cray MPI implementation

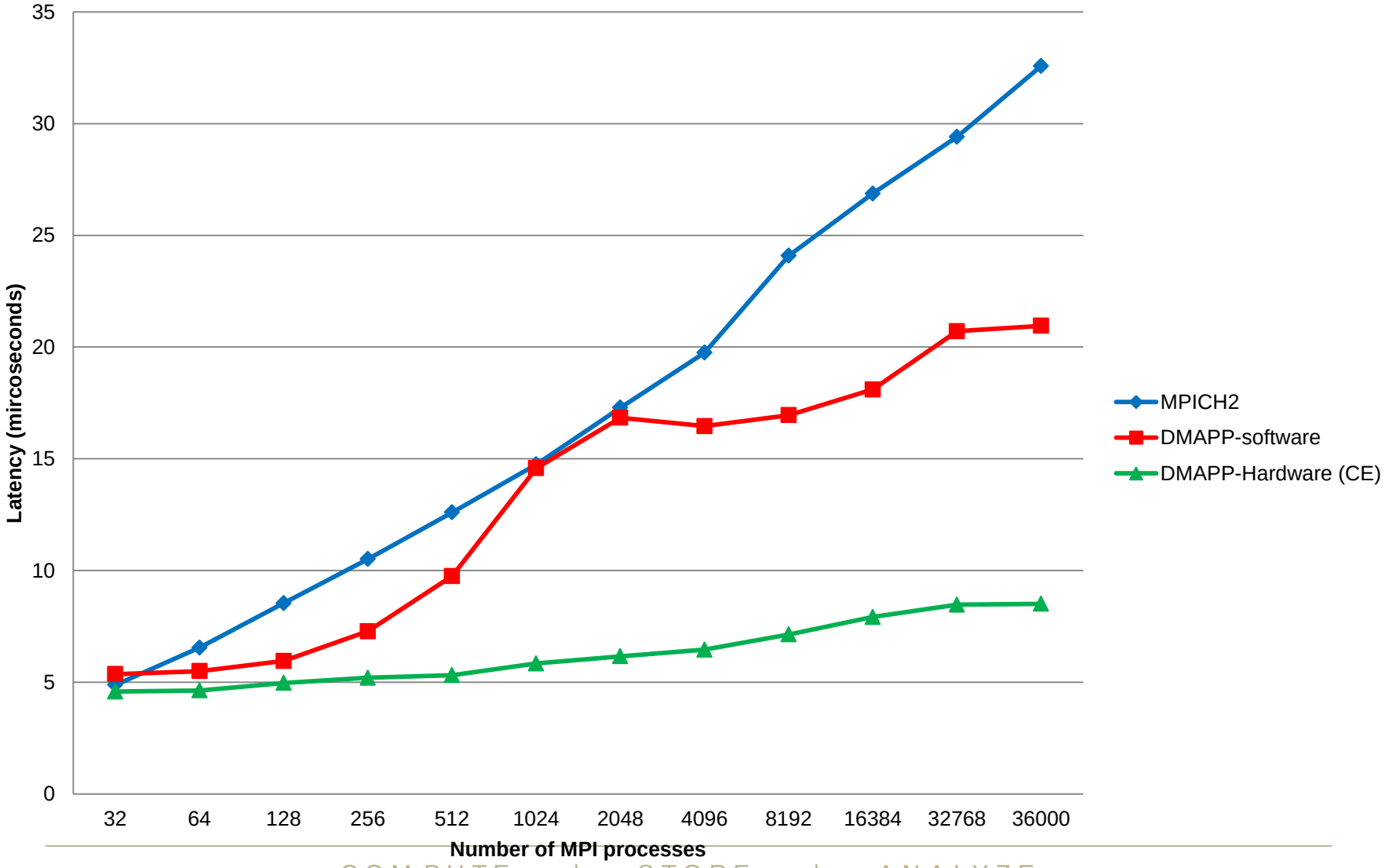
- **By setting MPICH_MPIIO_STATS=2 you get a profile over your MPI-IO usage**
  - See ‘man mpi’ for more details
- **The new MPICH ABI in mpt v7 is compatible with other mpich implementations like Intel**
  - http://docs.cray.com/books/S-2544-703//S-2544-703.pdf
  - This allows to change the MPI lib used for dynamic linked mpi applications

MPI Features / Functionality for XE/XC

- **MPI on XC behaves essentially the same as MPI on XE**
  - GNI interface is the same for XE and XC
    - Did not change name when going to Aries, but what used to be the “Gemini Network interface” became the “Generic network interface” ;)
  - MPICH code base is nearly the same
  - Messaging Paths (VSHORT, EAGER, RENDEZVOUS) are Identical

- **Enhanced Features for XC**
  - Modified MPI Asynchronous Progress Engine Threads
    - Threads can be placed on unused Intel hyper thread cores
  - XC Hardware Collective Engine (CE)
    - XC supports hardware-offload of Barrier & Allreduce collectives
    - Invoke these via MPICH_USE_DMAPP_COLL and MPICH_DMAPP_HW_CE enviroment variables
    - Must also link libdmapp into your application (see ‘man mpi’)

CE Test : 8-byte MPI_Allreduce Latency (16p/node)
Comparing MPICH2 software, DMAPP software, Aries Collective Engine
Cray XC system (CSCS daint) - 03/21/13



MPI Gemini/Aries Software Implementation

- **MPI device for Gemini/Aries based on**
  - User level Generic Network Interface (uGNI)
  - Distributed Memory Applications (DMAPP) library
  - Documentation for both APIs can be found under docs.cray.com

- **Data can be transferred in two ways :**
  - Fast Memory Access : FMA
    - In general used for small transfers
    - FMA transfers are lower latency
  - Block Transfer Engine : BTE
    - BTE transfers take longer to start but can transfer large amount of data without CPU involvement (DMA offload engine)

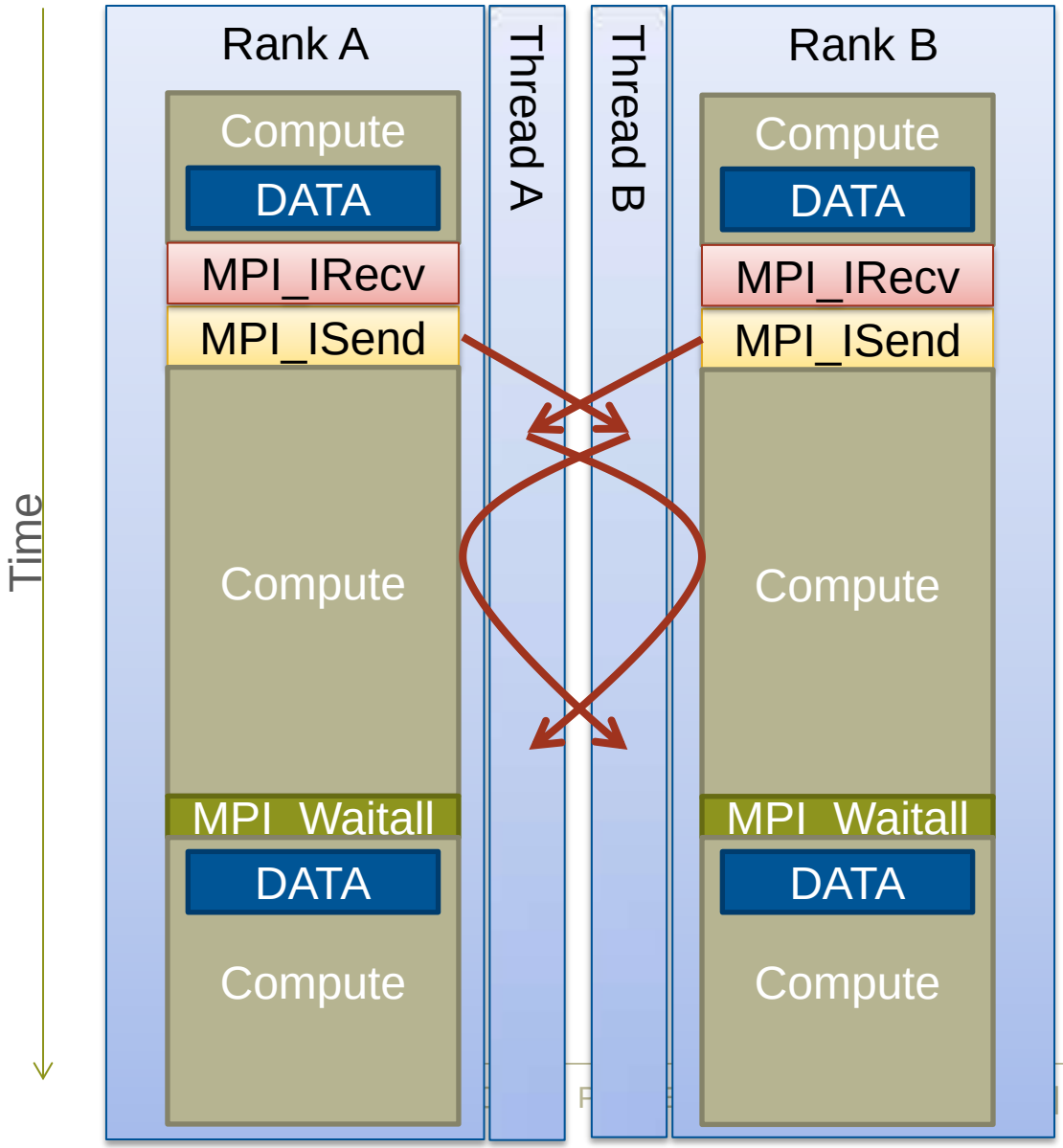
Which is Better

- **FMA PROS**
  - Lowest latency (<1.4 usec)
  - All data has been read by the time dmapp returns
  - More than one transfer active at the same time
- **FMA CONS**
  - CPU involved in the transfer
  - Performance can vary depending on die used
- **BTE PROS**
  - Transfer done by Gemini/Aries NIC, asynchronous with CPU
  - Seems to get better P2P bandwidth in more cases
- **BTE CONS**
  - Higher latency
    - Transfers are queued if Aries is busy
  - Only 4 BTE transaction can be active at a time (4 virtual channels)

MPICH and Cray MPT

- **Cray MPI uses MPICH distribution from Argonne**
  - Provides a good, robust and feature rich MPI
  - Cray provides enhancements on top of this:
    - low level communication libraries
    - Point to point tuning
    - Collective tuning
    - Shared memory device is built on top of Cray XPMEM
- **Many layers are straight from MPICH**
  - Error messages can be from MPICH or Cray Libraries.

Progress threads help overlap



Cray's MPT library can spawn additional threads that allow progress of messages while computation occurs in the background.

Thread performs message matching and initiates the transfer.

Data has already arrived by the time Waitall is called, so overlap between compute and communication.

MPI - Async Progress Engine Support

- Used to improve communication/computation overlap
  - Each MPI rank starts a “helper thread” during MPI_Init
- Helper threads progress MPI engine while application computes
- Only inter-node messages that use Rendezvous Path are progressed (relies on BTE for data motion)
- To enable on XC when using 1 stream per core:
  - `export MPICH_NEMESIS_ASYNC_PROGRESS=1`
  - `export MPICH_MAX_THREAD_SAFETY=multiple`
  - `export MPICH_GNI_USE_UNASSIGNED_CPUS=enabled`
  - Run application: `aprun –n XX  a.out`
- To enable on XE or XC when using 2 streams per core recommend running with the corespec option:
  - `export MPICH_NEMESIS_ASYNC_PROGRESS=1`
  - `export MPICH_MAX_THREAD_SAFETY=multiple`
  - Run application with corespec: `aprun –n XX  -r [1-2]  a.out`
- 10% or more performance improvements with some apps

Other Techniques - Collectives

- `MPICH_COLL_OPT_OFF=<collective name>` switches off the Cray optimized algorithm for a given collective and uses the original MPICH algorithm
  - E.g. `MPICH_COLL_OPT_OFF=mpi_allgather`
- The algorithm selection for all-to-all routines (allgather(v), alltoall(v)) is based on the number of ranks on the calling communicator and the message sizes.
- This can be adjusted with `MPICH_XXXX_VSHORT_MSG` environment variable, where XXXX=collective, e.g. ALLGATHER.

Why use Huge Pages

- **Gemini/Aries performs better with HUGE pages than with 4K pages. They can map more pages using fewer resources meaning communications may be faster.**

- **Huge Pages will also affect TLB performance:**
  - Your code may run with fewer TLB misses (hence faster)
  - However, your code may load extra data and so run slower
  - Only way to know is by experimentation.

- **Use modules to change default page sizes (`man intro_hugepages`):**
  - `e.g. module load craype-hugepages#`
    - `craype-hugepages128K (not on XC30)`
    - `craype-hugepages512K (not on XC30)`
    - **`craype-hugepages2M`**
    - **`craype-hugepages8M`**
    - `craype-hugepages16M`

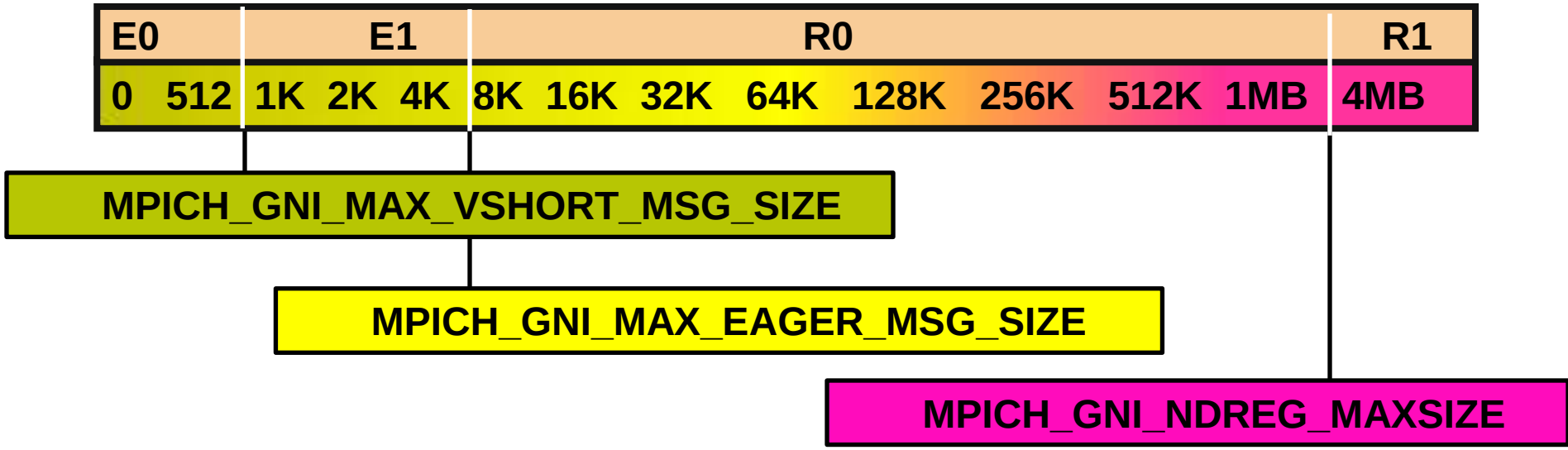
Most commonly successfully on Cray XE/XC

More detail.

- In reality, there are two different Eager and Rendezvous protocols in use with Cray's MPI.
- The following slides show more detail on the implementation and possible techniques for tuning them.

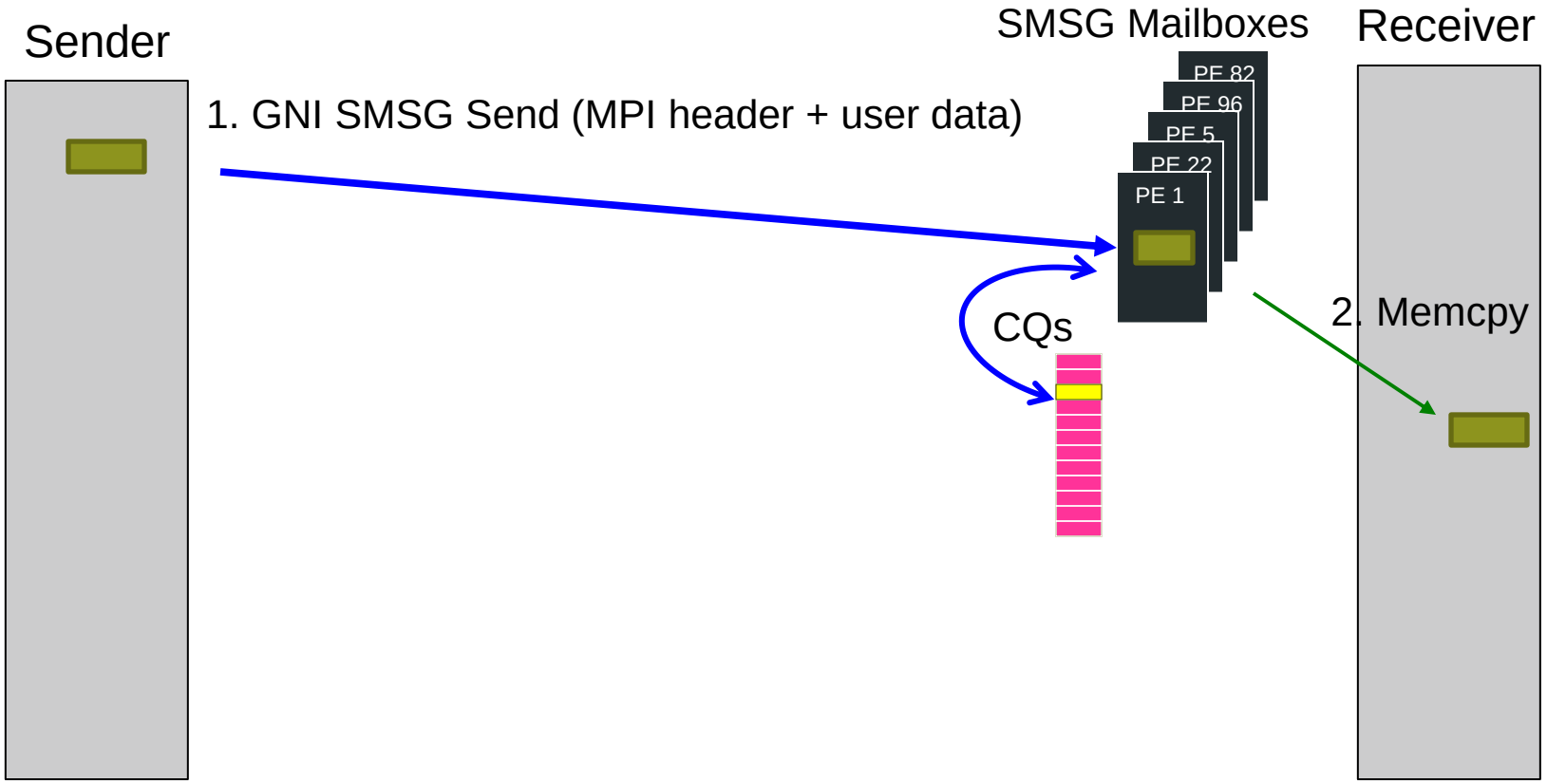
Day in the Life of an MPI Message

- **Four Main Pathways through the MPICH2 GNI NetMod**
  - Two EAGER paths (E0 and E1)
    - For a message that can fit in a GNI SMSG mailbox (E0)
    - For a message that can't fit into a mailbox but is less than MPICH_GNI_MAX_EAGER_MSG_SIZE in length (E1)
  - Two RENDEZVOUS (aka LMT) paths : R0 (RDMA get) and R1 (RDMA put)
- **Selected Pathway is based on Message Size**



Day in the Life of Message type E0

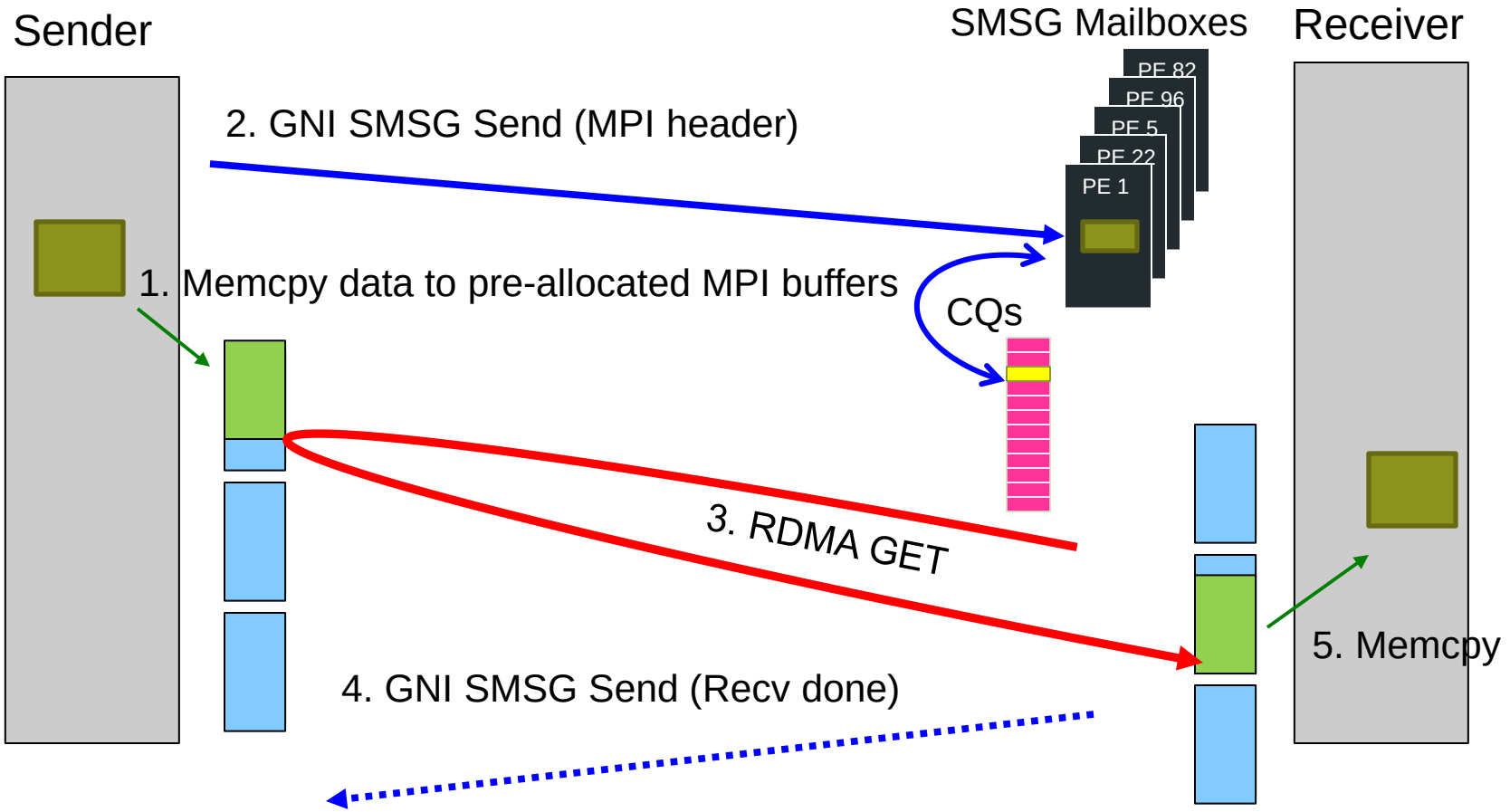
EAGER messages that fit in the GNI SMSG Mailbox



- GNI SMSG Mailbox size changes with number of ranks in job
- If user data is 16 bytes or less, it is copied into the MPI header

Day in the Life of Message type E1

EAGER messages that don't fit in the GNI SMSG Mailbox



- **User data is copied into internal MPI buffers on both send and receive side**

MPICH_GNI_NUM_BUFS
default 64 buffers, each 32K

EAGER Message Protocol

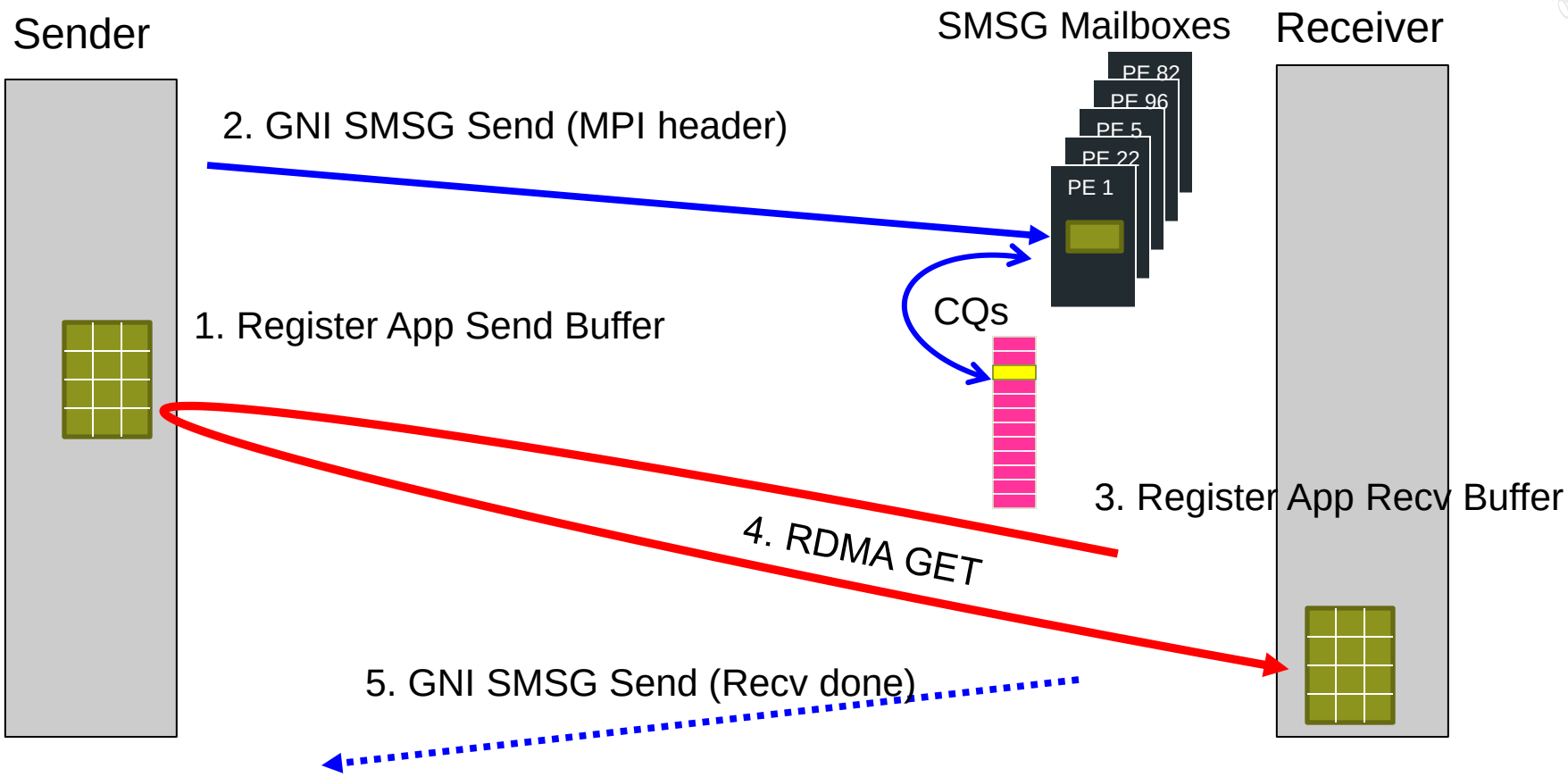
Default mailbox size varies with number of ranks in the job

- **Protocol for messages that can fit into a GNI SMSG mailbox**
- **The default varies with job size, although this can be tuned by the user to some extent**

Ranks in Job	Max user data (MPT 5.3)	MPT 5.4 and later
< = 512 ranks	984 bytes	8152 bytes
> 512 and <= 1024	984 bytes	2008 bytes
> 1024 and < 16384	472 bytes	472 bytes
> 16384 ranks	216 bytes	216 bytes

Day in the Life of Message type R0

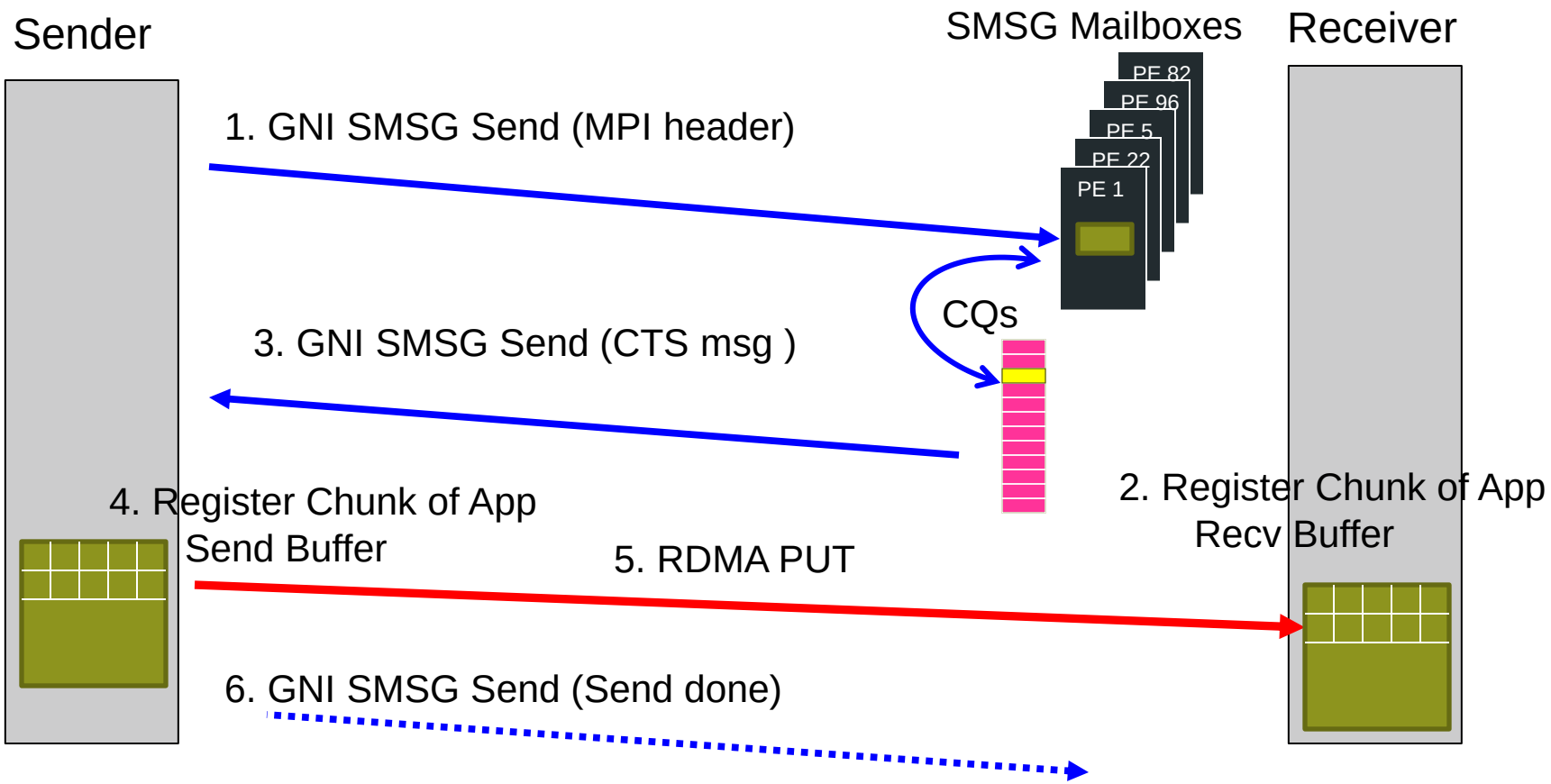
Rendezvous messages using RDMA Get



- **No extra data copies**
- **Best chance of overlapping communication with computation**

Day in the Life of Message type R1

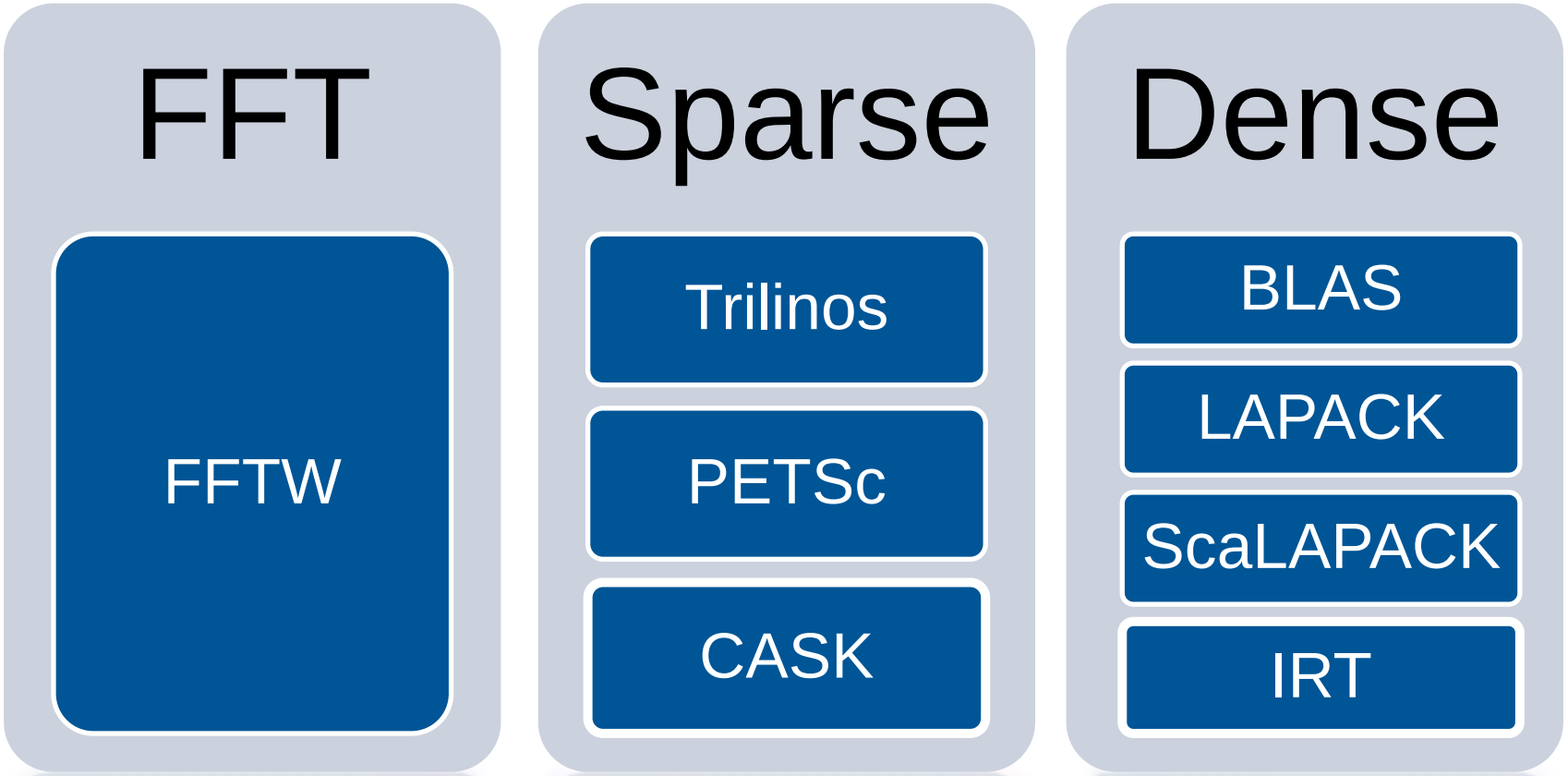
Rendezvous messages using RDMA Put



- ***Repeat steps 2-6 until all sender data is transferred***
- **Chunksize is MPI_GNI_MAX_NDREG_SIZE (default of 4MB)**

Cray Scientific Libraries

Scientific libraries on XC – functional view



What makes Cray libraries special

1. **Node performance**
   - Highly tuned routines at the low-level (ex. BLAS)
2. **Network performance**
   - Optimized for network performance
   - Overlap between communication and computation
   - Use the best available low-level mechanism
   - Use adaptive parallel algorithms
3. **Highly adaptive software**
   - Use auto-tuning and adaptation to give the user the known best (or very good) codes at runtime
4. **Productivity features**
   - Simple interfaces into complex software

LibSci usage

- **LibSci**
  - The drivers should do it all for you - no need to explicitly link (or load)
  - Threading is used within LibSci
  - If you call within a parallel region, single thread used
  - For threads, set OMP_NUM_THREADS
    - NOTE : With PBSpro v12 the default is :

      OMP_NUM_THREADS='All Cores on a node'

      Even if HLRS doesn't use this scheduler, you might run into this on other systems.
- **FFTW**
  - `module load fftw` (there are also wisdom files available)
- **PETSc**
  - `module load petsc` (or module load petsc-complex)
  - Use as you would your normal PETSc build
- **Trilinos**
  - `module load trilinos`
- **Cray Adaptive Sparse Kernels (CASK)**
  - You get optimizations for free

Check you got the right library!

- Add options to the linker to make sure you have the correct library loaded.
- `-Wl` adds a command to the linker from the driver
- You can ask for the linker to tell you where an object was resolved from using the –y option.
  - E.g. `–Wl,-ydgemm_` (notice the '_' at the end of the name)

```
.//main.o: reference to dgemm_
/opt/xt-libsci/11.0.05.2/cray/73/mc12/lib/libsci_cray_mp.a(dgemm.o):
definition  of dgemm_
```

Note: do not explicitly link "-lsci". This will not be found from libsci 11+ and means a single core library for 10.x.

Threading

- **LibSci is compatible with OpenMP**
  - Control the number of threads to be used in your program using **OMP_NUM_THREADS**
  - e.g., in job script **`export OMP_NUM_THREADS=16`**
  - Then run with **`aprun -n1 -d16`**
- **What behavior you get from the library depends on your code**
  1. No threading in code
     - The BLAS call will use OMP_NUM_THREADS threads
  2. Threaded code, outside parallel regions
     - The BLAS call will use OMP_NUM_THREADS threads
  3. Threaded code, inside parallel regions
     - The BLAS call will use a single thread

Threaded LAPACK

- Threaded LAPACK works exactly the same as threaded BLAS
- Anywhere LAPACK uses BLAS, those BLAS can be threaded
- Some LAPACK routines are threaded at the higher level
- No special instructions

FFTW

- **Cray's main FFT library is FFTW from MIT**
  - Some additional optimizations for Cray hardware
- **Usage is simple**
  - Load the module
  - In the code, call an FFTW plan
- **Cray's FFTW provides wisdom files for these systems**
  - You can use the wisdom files to skip the plan stage
  - This can be a significant performance boost
- **FFTW 3.3.4.0 includes Cray optimizations for HSW**
- **CRAFFT is not longer supported**

Iterative Refinement Toolkit

- **Mixed precision can yield a big win on x86 machines.**
- **SSE (and AVX) units issue double the number of single precision operations per cycle.**
- **On CPU, single precision is always 2x as fast as double**
- **Accelerators sometimes have a bigger ratio**
  - Cell – 10x
  - Older NVIDIA cards – 7x
  - New NVIDIA cards (2x)
  - Newer AMD cards (> 2x)
- **IRT is a suite of tools to help exploit single precision**
  - A library for direct solvers
  - An automatic framework to use mixed precision under the covers

Iterative Refinement Toolkit - Library

- Various tools for solving linear systems in mixed precision
- Obtaining solutions accurate to double precision
  - For well conditioned problems
- Serial and Parallel versions of LU, Cholesky, and QR

Iterative Refinement Toolkit - Library

- 2 usage methods
  - **IRT Benchmark routines**
    - Uses IRT 'under-the-covers' without changing your code
      - Simply set an environment variable
      - Useful when you cannot alter source code
  - **Advanced IRT API**
    - If greater control of the iterative refinement process is required
    - Allows
      - condition number estimation
      - error bounds return
      - minimization of either forward or backward error
      - 'fall back' to full precision if the condition number is too high
      - max number of iterations can be altered by users

IRT library usage

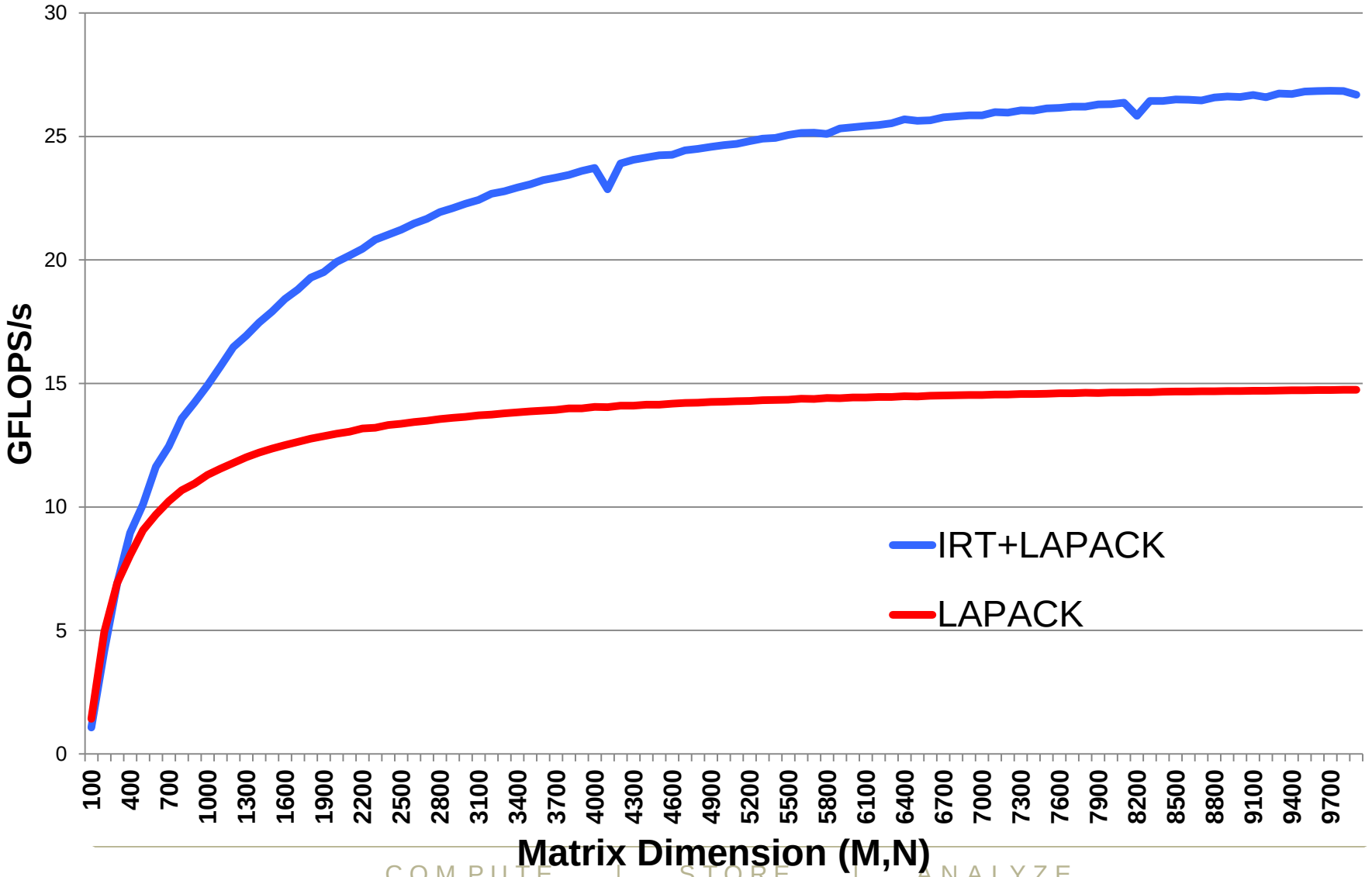
- **Benchmark API**

  `setenv IRT_USE_SOLVERS 1`

- **Advanced API**

  1. locate the factor and solve in your code (LAPACK or ScaLAPACK)
  2. Replace factor and solve with a call to IRT routine, e.g.
     - dgesv -> irt_lu_real_serial
     - pzgesv -> irt_lu_complex_parallel
  3. Set advanced arguments
     - Forward error convergence for most accurate solution
     - Condition number estimate
     - “fall-back” to full precision if condition number too high

IRT with LAPACK LU DGETRF
AMD Bulldozer 2.1 GHz
2threads :: September 2012



Third party Scientific Libraries (cray-tpsl)

- TPSL (Third Party Scientific Libraries) contains a collection of outside mathematical libraries that can be used with PETSc and Trilinos.
- This module will increase the flexibility of PETSc and Trilinos by providing users with multiple options for solving problems in dense and sparse linear algebra.
- The cray-tpsl module is automatically loaded when PETSc or Trilinos is loaded. The libraries included are MUMPs, SuperLU, SuperLU_dist, ParMetis, Hypre, Sundials, and Scotch.

cray-tpsl

MUMPS 4.9.2
MUMPS (MUltifrontal Massively Parallel sparse direct Solver) is a package of parallel, sparse, direct linear-system solvers based on a multifrontal algorithm. For further information, see http://graal.ens-lyon.fr/MUMPS/. MUMPS can now interface with SCOTCH as well.

SuperLU 4.3
SuperLU is a sequential version of SuperLU_dist and a sequential incomplete LU preconditioner that can accelerate the convergence of Krylov subspace iterative solvers. For further information, see http://crd.lbl.gov/~xiaoye/SuperLU/.

SuperLU_dist 3.1
SuperLU_dist is a package of parallel, sparse, direct linear-system solvers (available in Cray LibSci). For further information, see http://crd.lbl.gov/~xiaoye/SuperLU/.

ParMETIS 4.0.2
ParMETIS (Parallel Graph Partitioning and Fill-reducing Matrix Ordering) is a library of routines that partition unstructured graphs and meshes and compute fill-reducing orderings of sparse matrices. For further information, see http://glaros.dtc.umn.edu/gkhome/views/metis/.

cray-tpsl

ParMETIS 4.0.2

ParMETIS (Parallel Graph Partitioning and Fill-reducing Matrix Ordering) is a library of routines that partition unstructured graphs and meshes and compute fill-reducing orderings of sparse matrices. For further information, see http://glaros.dtc.umn.edu/gkhome/views/metis/.

HYPRE 2.8.0b

HYPRE is a library of high-performance preconditioners that use parallel multigrid methods for both structured and unstructured grid problems (not included with petsc-complex). For further information, see http://www.llnl.gov/CASC/linear_solvers/.

SUNDIALS 2.5.0

(SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) consists of 5 solvers: CVODE, CVODES, IDA, IDAS, and KINSOL. In addition, SUNDIALS provides a MATLAB interface to CVODES, IDAS, and KINSOL that is called sundialsTB. For further information, see https://computation.llnl.gov/casc/sundials/main.html.