Improving the Performance of the PSDNS Pseudo-Spectral Turbulence Application on Blue Waters using Coarray Fortran and Task Placement

CUG 2013

R. Fiedler, N. Wichmann, & S. Whalen, (Cray), and D. Pekurovsky (SDSC)

Outline

- **Blue Waters petascale acceptance benchmark**
- **Background**
  - Cray gemini network
  - Bisection bandwidth
- **Algorithm & performance model**
- **Memory allocation optimization**
- **Communication optimizations**
  - Minimize off-node communication
  - Choose best layout of tasks on system
  - Use Coarray Fortran for All-to-All
- **Summary and future work**

Blue Waters petascale acceptance benchmark

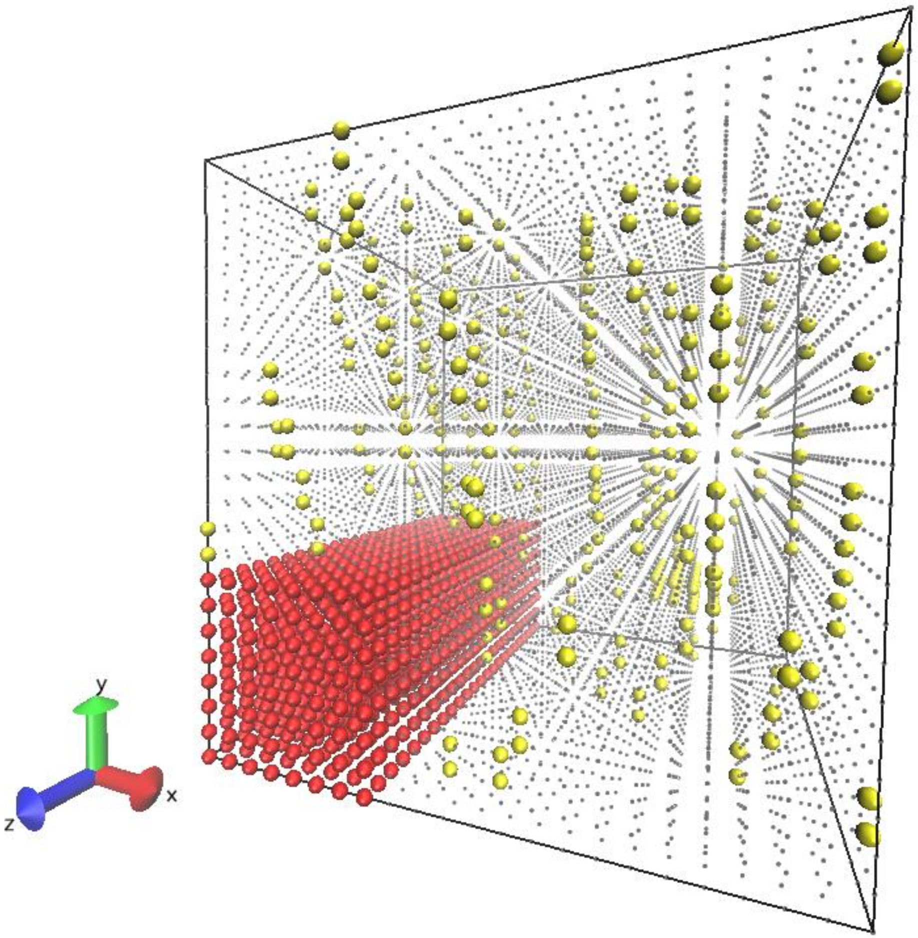
Direct Numerical Simulation of isotropic turbulence

- **Pseudo-spectral method (3D FFTs)**
- **Domain has 12288^3 grid points**
- **4th order Runge-Kutta time stepping, 10000 steps**
- **Double precision, 50 output dumps (74 TB each)**
- **Original plan: run on 2*12288 nodes (1 socket)**
- **Selected application**
  - PSDNS (D. Donzis, P. K. Yeung, D. Pekurovsky)
- **Initial assessment on smaller problem & system:**
  - Performance as of 3/2012 was well below model prediction

Background

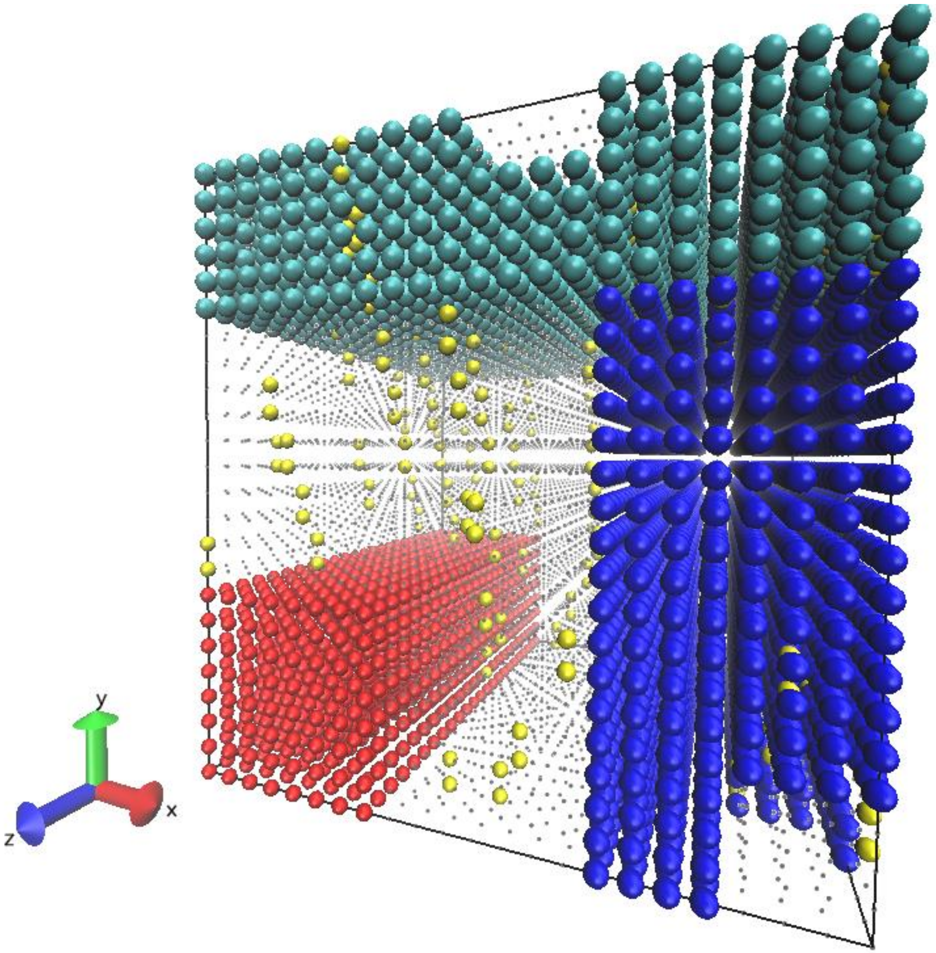
Blue Waters Interconnect

- Topology is 23x24x24 gemini routers
- 2 nodes per gemini, 2 geminis along y per blade
- 8x8x24 XK geminis (red)
- Service blades randomly distributed (yellow)
- x and z-links have 2X bandwidth of y-links between blades
  - 2 nodes on same gemini don't use interconnect to exchange messages
- Routing algorithm is x, then y, then z



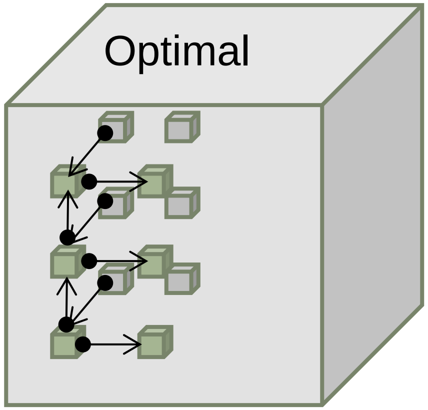
Background

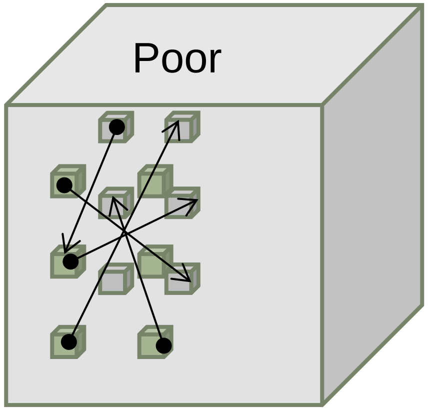
- Routing takes shortest path
- If using > 1/2 of geminis in any dimension, traffic may wrap around the torus through geminis not assigned to job
- Jobs share interconnect for application communication, IO
- Run times affected by task placement, other running jobs



Task Placement and Interference

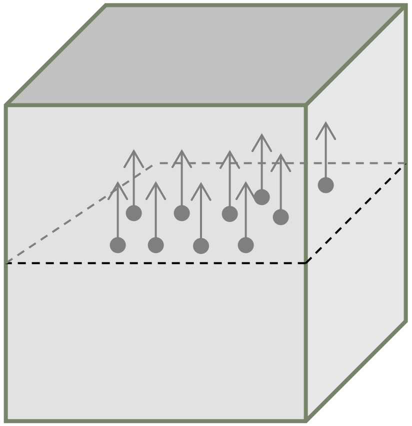
- **Applications that perform more communication are more sensitive to placement and interference**
  - Applications with All-to-All communication patterns compete more with other jobs
- **Applications with only nearest-neighbor communication in their virtual topology, if poorly placed, actually perform pairwise communication between randomly located nodes**
  - Thus, analysis below of bisection bandwidth for All-to-All is relevant to many types of applications



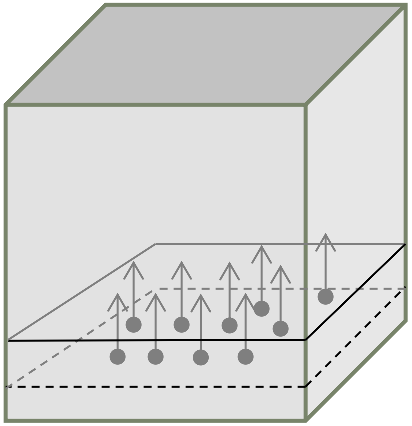


Bisection Bandwidth

- **Bisection bandwidth of nodes in use determines run time for All-to-All**
- **Bisection bandwidth is defined as lowest bandwidth through any bisecting plane**
  - BW topology is 23x24x24 geminis
  - Bisection bandwidth through cross section:
    - Normal to x: 24*24*x-link-bw*2 for torus
    - Normal to y: 23*24*y-link-bw*2 for torus
    - Normal to z: 23*24*z-link-bw*2 for tours
  - Y-link bandwidth ~ 1/2 x-link or z-link bandwidth
  - Bisection bandwidth normal to y ~ 23*24*x-link-bw, limits All-to-All



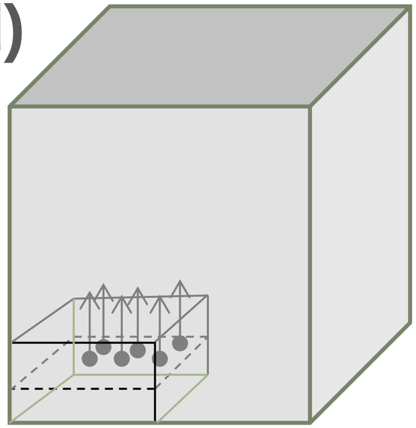
Bisection Bandwidth



- **Consider subset of nodes: 23x6x24**
- **Contains ¼ of all nodes**
- **Bisection bandwidth through cross section:**
  - Normal to x: 6*24*x-link-bw*2 for torus ~ 12x24*x-link-bw
  - Normal to y: 23*24*y-link-bw ~ 23x12*x-link-bw
  - Normal to z: 23*6*z-link-bw*2 for tours = 23x12 x-link-bw
- **Bisection bandwidth normal to y ~ EQUALS that of other directions**
- **Bisection bandwidth for this subset is ~1/2 of bisection bandwidth for full system**
- **Gives highest possible bandwidth per node for All-to-All communication for > 2000 nodes**

Bisection Bandwidth

- **23x6x24 gemini subsection best for ~ 6k nodes**
  - 23x4x24 best for ~ 4k nodes
- **Consider smaller node counts, e.g., 11x6x12 so no wrapping around torus (shortest route is used)**
  - 1584 nodes, ~1/16 of all nodes in system
- **Bisection bandwidth through cross section:**
  - Normal to x: 6*12*x-link-bw ~ 12*6*x-link-bw
  - Normal to y: 11*12*y-link-bw ~ 11*6*x-link-bw
  - Normal to z: 11*6*z-link-bw = 11*6 x-link-bw

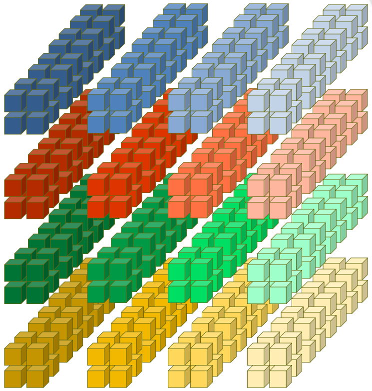


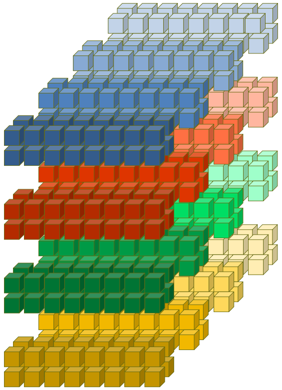
- **Bisection bandwidth normal to y ~ EQUALS that of other directions**
- **Bisection bandwidth for subset ~ 1/8 of bisection bandwidth for full system**
  - This shape also gives maximum bandwidth per node

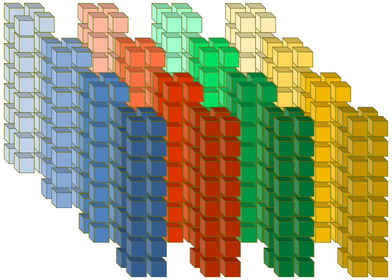
PSDNS Algorithm & Performance Model

CFD Using Pseudo-Spectral Method

- **Uses 3D FFTs of fluid variables to compute spatial derivatives**
- **Implementation uses 2D pencil decomposition**
- **For 3D FFT, must transpose full 3D arrays twice:**
  - Begin with partitions spanning domain in x
  - 1D FFTs along x
  - Transpose within xy planes so each partition spans domain in y
  - 1D FFTs along y
  - Transpose within yz planes so each partition spans domain in z
  - 1D FFTs along z
- **After some calculations requiring no communication, inverse 3D FFTs are performed in similar fashion**
  - Dozens of forward and inverse 3D FFTs per time step
- **Transposes comprise 50-75% of run time**
  - Compute time includes local field variable updates, packing/unpacking communication buffers, 1D FFTs







Single-Task Optimizations

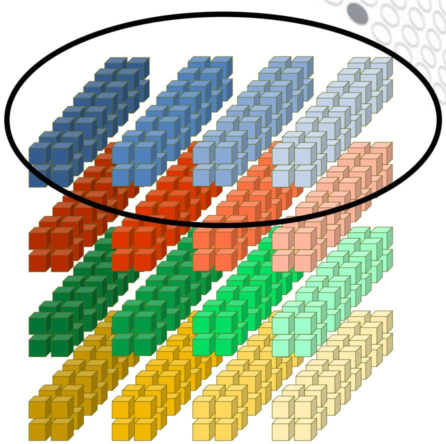
Improving “Compute” Time

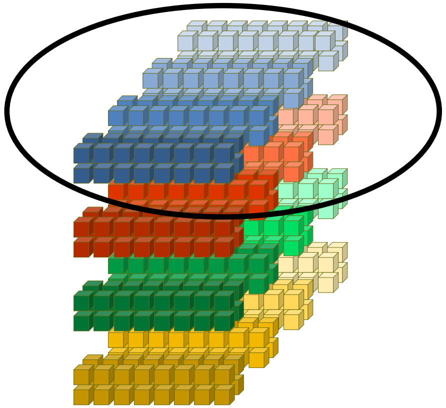
- PSDNS allocates/deallocates buffer arrays for communication every time it performs All-to-All operations
- For PGI (maybe GNU) compiler, a 10-20% improvement in run time was obtained by setting environment variables:
  - MALLOC_MMAP_MAX_=0
  - MALLOC_TRIM_THRESHOLD_=512MiB
- Cray compiler by default manages memory better, so setting these variables does not help
- Avoiding repeated allocation/deallocation of the same arrays may reduce overhead for many applications

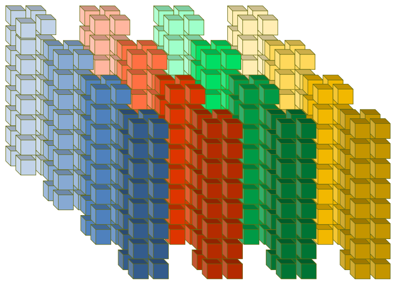
Communication Optimizations

Minimize off-node communication

- **Transposes require All-to-All communication within each row (column) of pencils**
  - Multiple concurrent All-to-Alls on all rows (columns), not global All-to-All
- **Eliminate inter-nodal communication for xy transposes**
  - Place 1 or more full xy planes of domain per node
  - Each node has an entire row (16 or 32) of pencils
- **In benchmark runs with a 6k^3 grid on 3072 nodes, this strategy reduced the overall run time by up to 1.72X!**
- **Possible to place 1 row of pencils per gemini (node pair), but must ensure both nodes are available on all geminis used**







Communication Optimizations

Improving Transposes, II

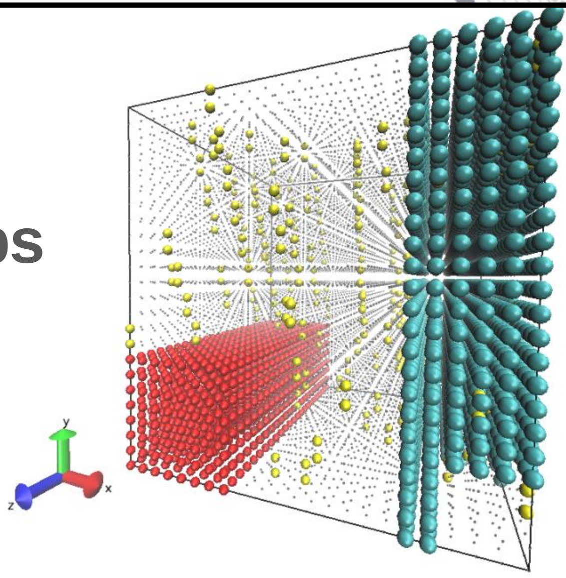
- **yz Transposes require off-node communication**
  - One process per node in each column communicator
  - Communication time depends on effective All-to-All bandwidth for nodes in job, plus any additional nodes relaying messages
- **Two approaches to increasing effective All-to-all bandwidth via placement**
  1. Request specific nodes & wait – works in shared batch mode
     - qsub -l hostlist=`cat node_list | sed -e 's/-/+/g' | sed -e 's/,/+/g'` job_script
  2. Run on a randomly distributed (spread out) set of nodes
     - Most useful on dedicated system (or node pool)
     - For a 6k^3 grid on 3072 nodes of ESS (~4500 nodes total), this strategy reduced the overall run time by ~21%

Communication Optimizations

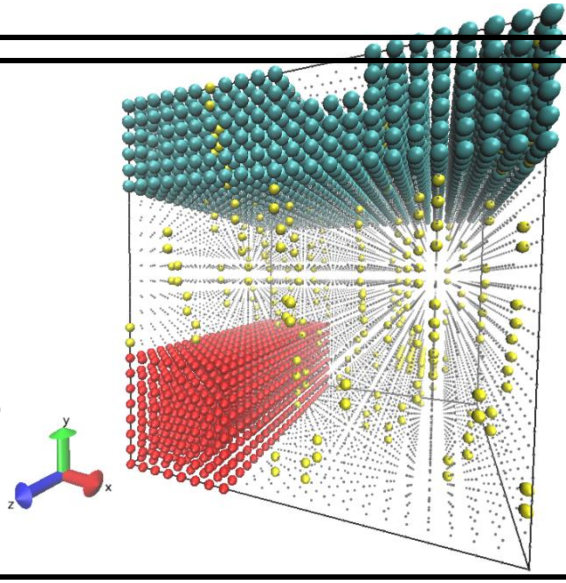
Sensitivity to Placement

- **6144 XE nodes, 8 non-IO steps, 2 IO steps**

- **6k-node job in 6x24x24 XE Region**
  - Ave max time per non-IO step: 35.3 s
  - Ave max time per IO step: 67.9 s



- **6k-node job in 23x6x24 XE region**
  - Ave max time per non-IO step: 21.5 s
  - Ave max time per IO step: 48.0 s
  - Step on slab normal to x takes 1.64X (1.41X for IO step) longer than on slab normal to y



Communication Optimizations

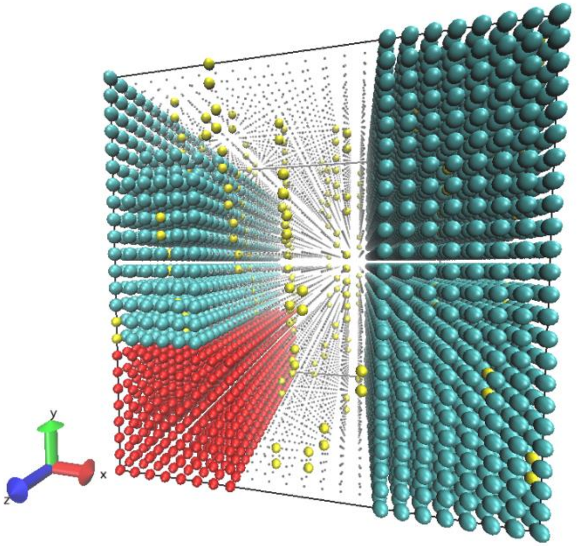
Ensuring both nodes on each gemini are up

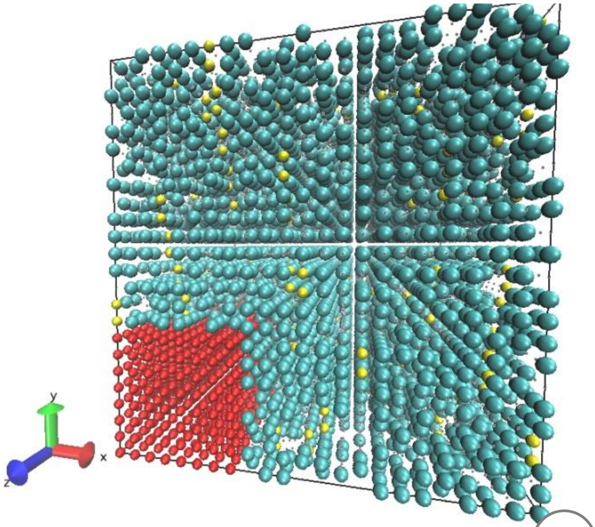
- **Request more nodes than needed (1% & up)**
  - Could use extra nodes for fault tolerance
- **At run time in batch script**
  - Get the list of nodes in reservation:

    aprun -B -D0x10000 /bin/true | head -1 > node_list

  - Node IDs on same gemini are consecutive even-odd integers
  - Randomization script can eliminate nodes with down partners:

    cat node_list | randomize.pl --block=2 > random_nodes
    aprun –l random_nodes …





- **Petascale benchmark on 12k nodes**
  - PSDNS on randomized nodes is 1.46X faster.

Communication Optimizations

Improving Transposes, III

- **Replace calls to MPI_AlltoAll with library routine in co-array Fortran (CAF)**
  - CAF has one-sided communication, lower latency, smaller headers
  - Library routine copies messages to/from 6 MB statically allocated co-array “bucket” on each image
  - Breaks messages into 512 B chunks
  - Pulls chunks from other images in different random order for each image
    - Reduces network congestion
  - Source code available on request
    - Tunable for specific application
    - Saves image-to-rank map & random orderings for row and column communicators
- **Reduces overall run time by ~33% on 4096 nodes**

Simplified CAF All-to-All Pseudo-Code

```
! My image is my_im

Do i=1,n_chunks  !  Number of 512 Byte chunks in messages
        i_start = 1 + (i-1)*512/8  ! 8 Bytes per word
        Do j=1,n_images  ! Number of images
                co_bucket(1:512/8, j) = sendbuf(i_start:i_start-1+512/8, j)
        End do ! images
        MPI barrier (communicator, ierr)
        Do j=1,n_images
                Set k = random_order ( j )
                recvbuf(i_start:i_start-1+512/8, k) =
                        co_bucket(1:512/8,my_im)[k] ! Pull from remote im.
        End do ! images
        Sync memory  ! Ensures compiled code gives correct results
        MPI barrier (communicator, ierr)
End do ! chunks
```

CAF in PSDNS

- Library expects mpi_byte data type
- Gets precision from PSDNS module (header file)
- Easily customized/generalized for other applications

```
#ifdef CAF
 call compi_alltoall(sendbuf,recvbuf,items,mpi_comm_col)
#else
 call mpi_alltoall(sendbuf,items,mpi_byte,
    &                recvbuf,items,mpi_byte,mpi_comm_col,ierr)
#endif
```

- compi_alltoallv also available, nearly as efficient

Summary and Future Work

- **Overall run time improvement on 12k nodes**

  1.1X for memory management (environment variables or switch to Cray compiler)

  1.4X for slab-on-node decomposition

  1.4X for randomizing node list, using node pairs with both partners available

  1.3X for CAF All-to-All library,

  2.8X overall (Conservative estimate, not directly measured)

- **Further PSDNS optimizations possible**
  - Eliminate extra copy to bucket in library by putting CAF directly in PSDNS
    - Coarray send buffers allocated just once
    - Test code shows only up to 5% improvement – bucket fits in L3 cache
  - Overlap communication for 1 vector component with computation for next component (2 out of 3 can be overlapped)
    - Try non-blocking MPI collectives
    - Need to use Block Transfer Engine, core specialization, 8 senders/node
    - Figure out best way to do this in CAF
- **Cray is improving MPI_AlltoAll (closer to CAF)**

CRAY

THE SUPERCOMPUTER COMPANY