MPI Performance Tuning Tutorial

Karl Feind

Parallel Communication Engineering

SGI

Overview

- Automatic MPI Optimizations
- Tunable MPI Optimizations
- Tips for Optimizing

Automatic Optimizations

- Optimized MPI send/recv
- Optimized Collectives
- NUMA Placement
- MPI One-Sided

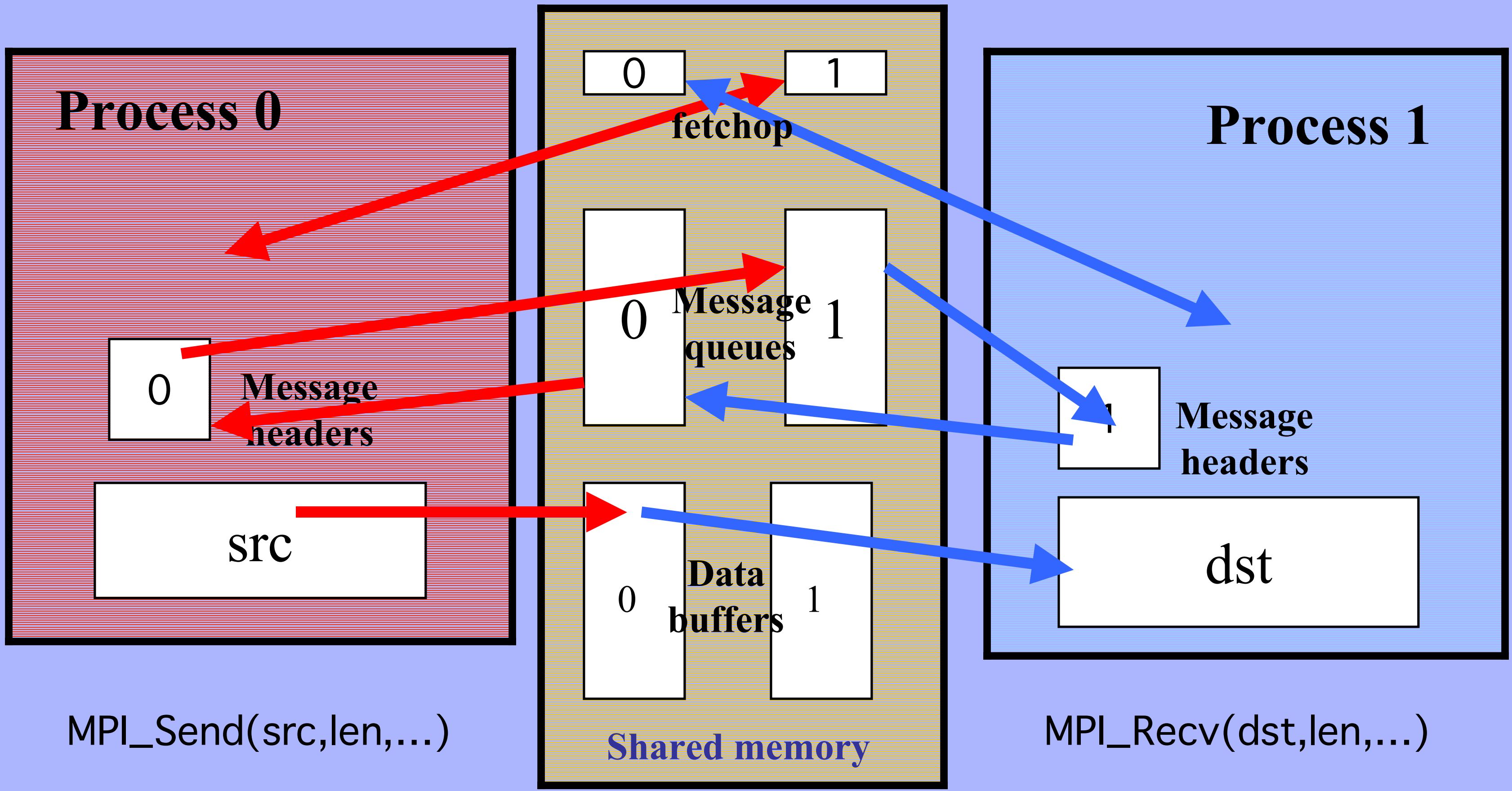
Automatic Optimizations

- Optimized MPI send/recv
  - MPI_Send() / MPI_Recv()
  - MPI_Isend() / MPI_Irecv()
  - MPI_Rsend() = MPI_Send()

Automatic Optimizations

- Optimized MPI send/recv
  - Low latency
  - High Bandwidth
  - High repeat rate
  - Fast path for less than 64 bytes

MPI Message exchange (on host)



MPI_Send(src,len,...)

MPI_Recv(dst,len,...)

MPI Latencies (Point-to-Point)

Latency for an 8 byte message(msec)

protocol	O2K 250 MHz R10K	O3K 400 MHz R12K
shared memory	6.9	4.4
HIPPI 800 OS bypass	130	-
GSN/ST	18.6	12.7
Myrinet 2000(GM)	-	18*
TCP	320	180

*actual results obtained on O300 500 MHz R14K

MPI Bandwidths (Point-to-Point)

Bandwidths in MB/sec

protocol	O2K 400 MHz R12K	O3K 400 MHz R12K
shared memory	80	174
HIPPI 800 OS bypass	70	-
GSN/ST	130	204
Myrinet 2000(GM)	-	170*
TCP	~40	~40

*actual results obtained on O300 500 MHz R14K

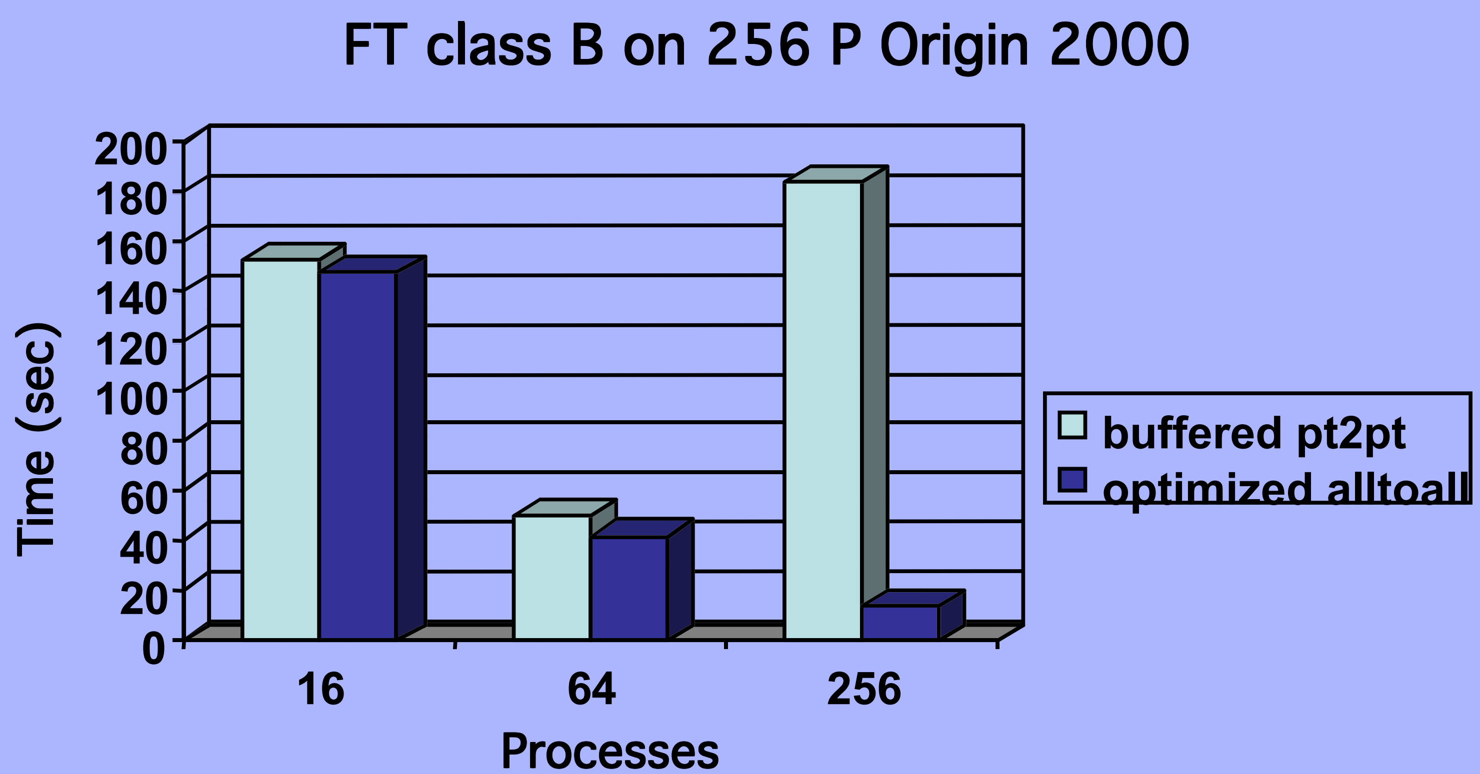
Automatic Optimizations

Optimized collective operations

routine	Shared memory optimization	Cluster optimization
MPI_Barrier	**yes**	**yes**
MPI_Alltoall	**yes**	**yes**
MPI_Bcast	**no**	**yes**
MPI_Allreduce	**no**	**yes**

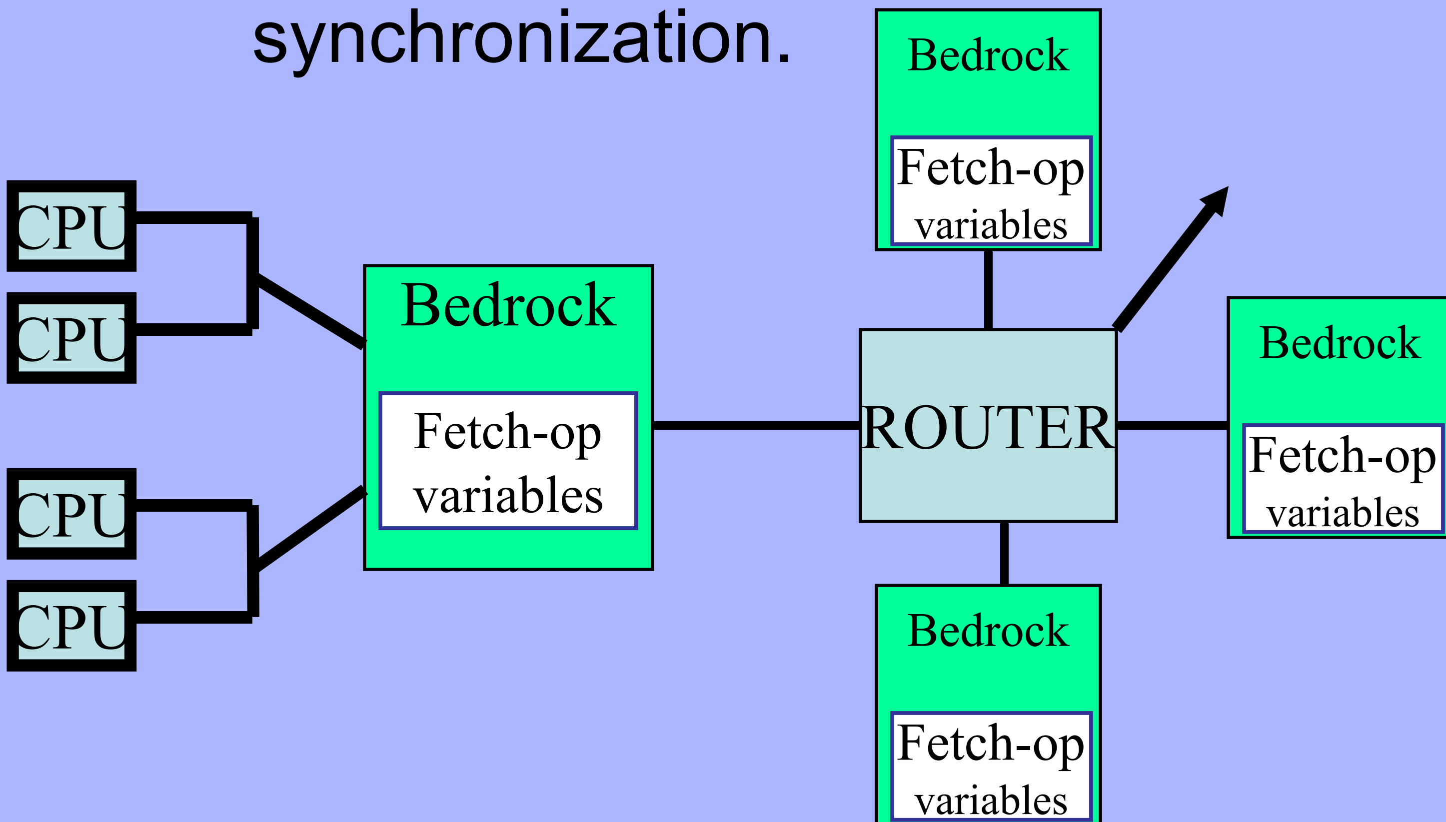
NAS Parallel FT Execution Time

Example of Collective Optimization

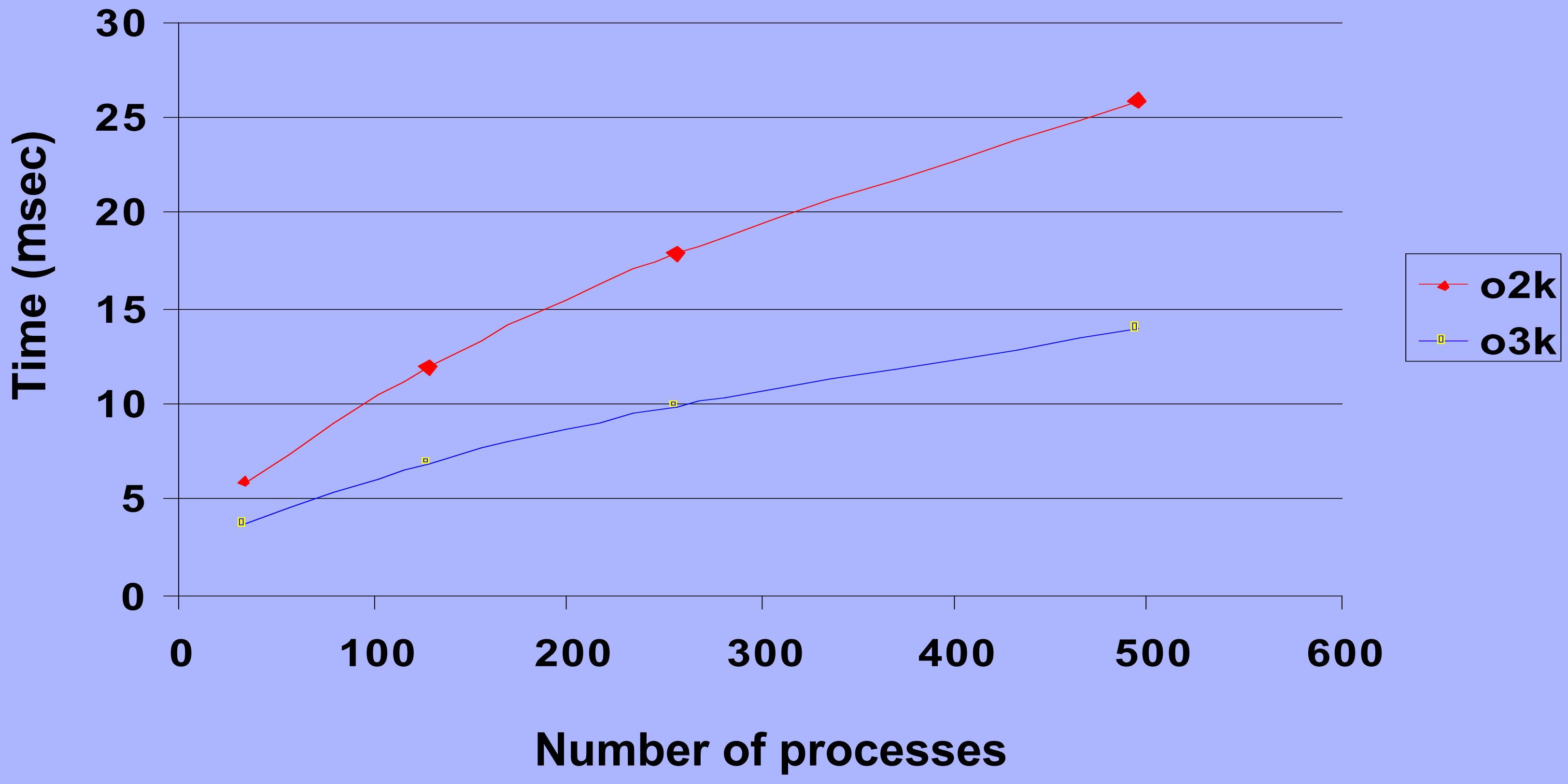


MPI Barrier Implementation

- MPI Barrier Implementation
  - fetch-op-variables on Bedrock provide fast synchronization.



Barrier Synchronization Time on O2K and O3K

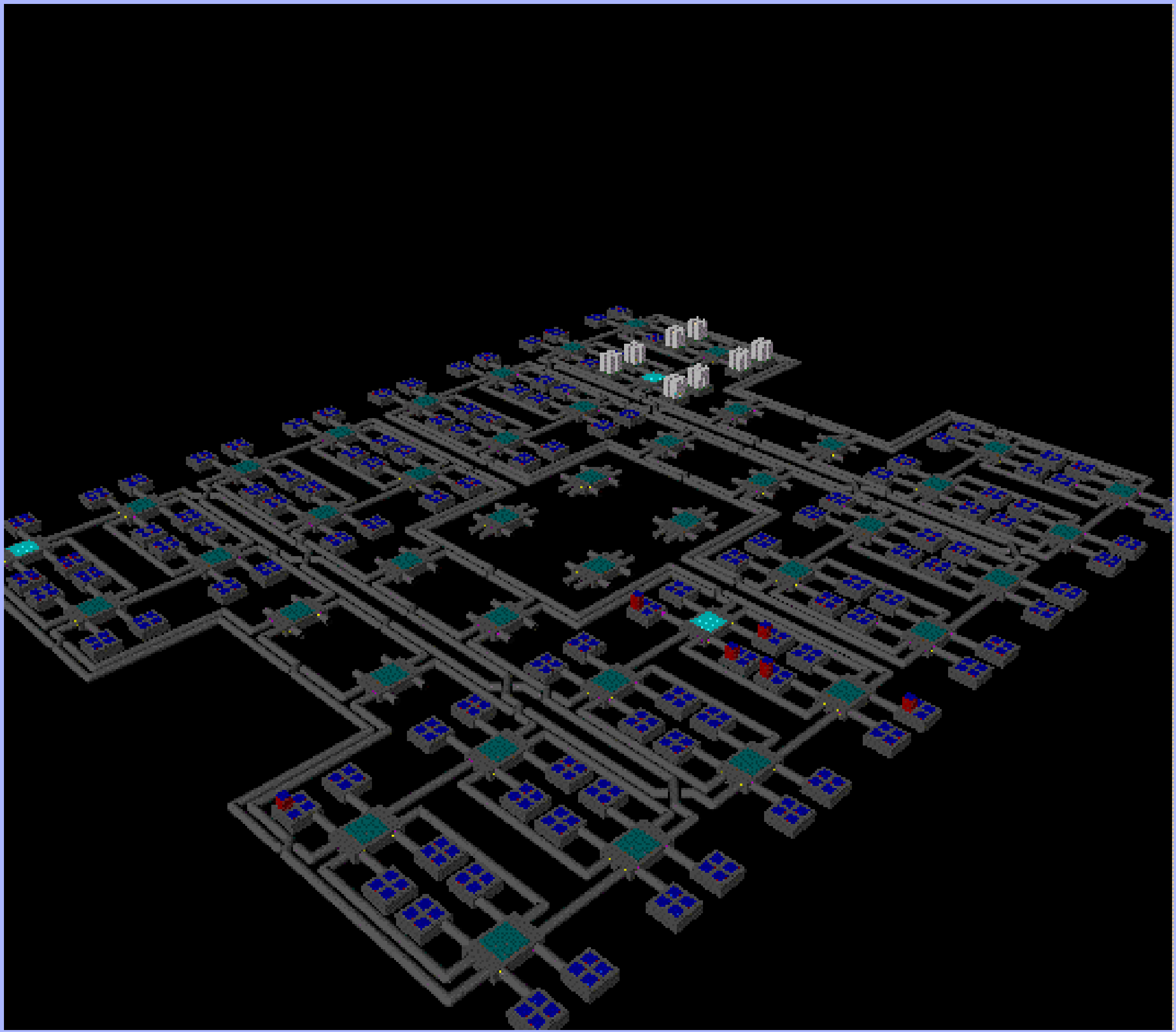


Automatic Optimizations

- Default NUMA Placement
  - IRIX 6.5.11 and later the default for MPI_DSM_TOPOLOGY is ***cpucluster***
  - Is ***cpuset*** and ***dplace*** aware
  - MPI does placement of key internal data structures

Automatic Optimizations

NUMA Placement (Performance Co-Pilot)

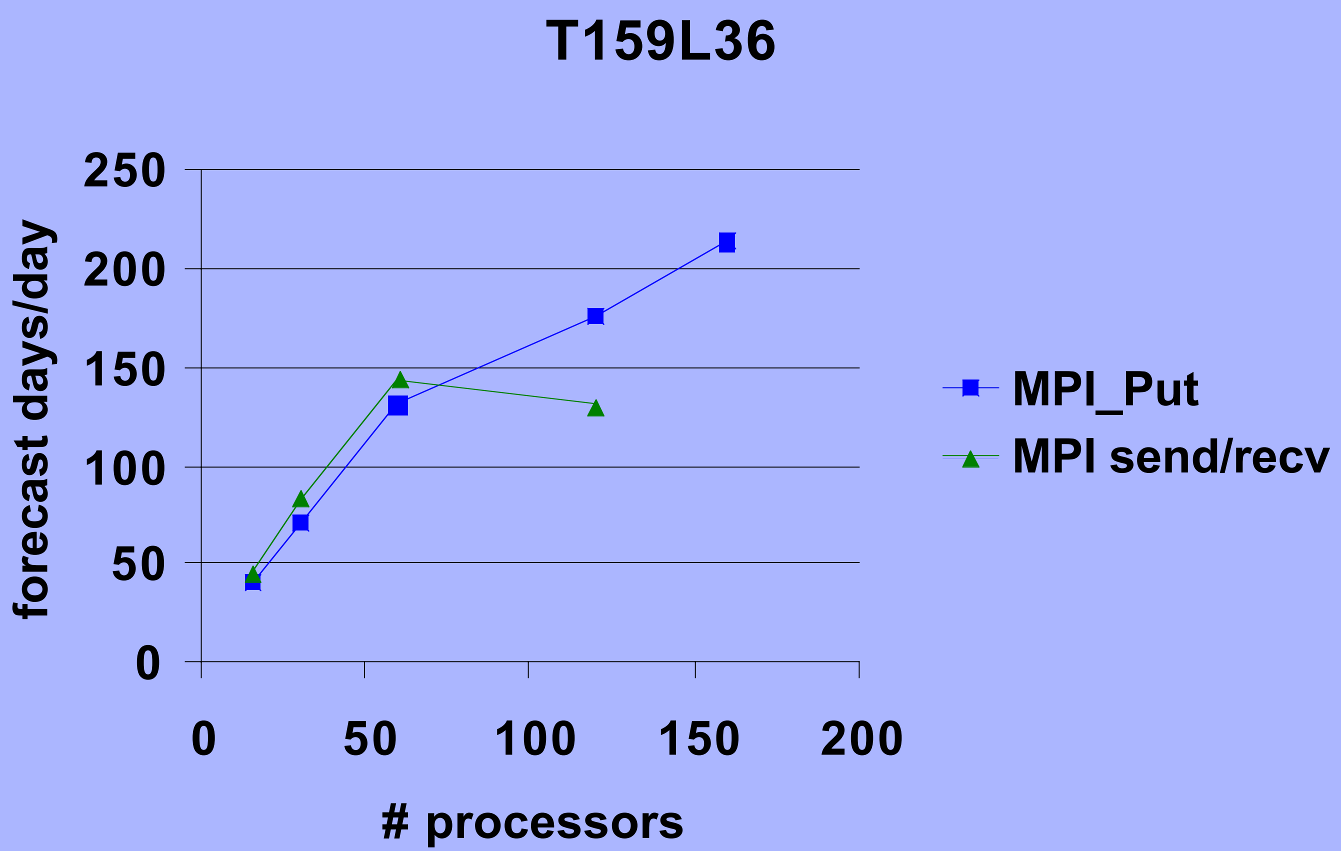


MPI One-Sided with NOGAPS

- NOGAPS is a weather forecast code used by FNMOC
- Spectral algorithm with global transposes
- Transposes were bandwidth limited
- Re-wrote transposition routines to use MPI get/put functions

Performance comparison of NOGAPS using Send/Recv vs Get/Put

Origin 2000, 250MHz R10k, 48hr forecast T159L36



Tunable Optimizations

- Eliminate Retries
- Single Copy Optimization
- Single Copy (MPI_XPMEM_ON)
- Single Copy (BTE copy)
- Control NUMA Placement
- NUMA Placement of MPI/OpenMP hybrid codes

Tunable Optimizations

- Eliminate Retries (Use MPI statistics)

  setenv MPI_STATS

  or

  mpirun -stats -prefix “%g:” -np 8 a.out

```
3: *** Dumping MPI internal resource statistics...
3:
3:  0 retries allocating mpi PER_PROC headers for collective calls
3:  0 retries allocating mpi PER_HOST headers for collective calls
3:  0 retries allocating mpi PER_PROC headers for point-to-point calls
3:  0 retries allocating mpi PER_HOST headers for point-to-point calls
3:  0 retries allocating mpi PER_PROC buffers for collective calls
3:  0 retries allocating mpi PER_HOST buffers for collective calls
3:  0 retries allocating mpi PER_PROC buffers for point-to-point calls
3:  0 retries allocating mpi PER_HOST buffers for point-to-point calls
3:  0 send requests using shared memory for collective calls
3:  6357 send requests using shared memory for point-to-point calls
3:  0 data buffers sent via shared memory for collective calls
3:  2304 data buffers sent via shared memory for point-to-point calls
3:  0 bytes sent using single copy for collective calls
3:  0 bytes sent using single copy for point-to-point calls
3:  0 message headers sent via shared memory for collective calls
3:  6357 message headers sent via shared memory for point-to-point calls
3:  0 bytes sent via shared memory for collective calls
3:  15756000 bytes sent via shared memory for point-to-point calls
```

Tunable Optimizations
Eliminate Retries

- Want to reduce number of retries for buffers to zero if possible-

MPI_BUFS_PER_PROC - increase the setting of this shell
variable to reduce retries for
per proc data buffers

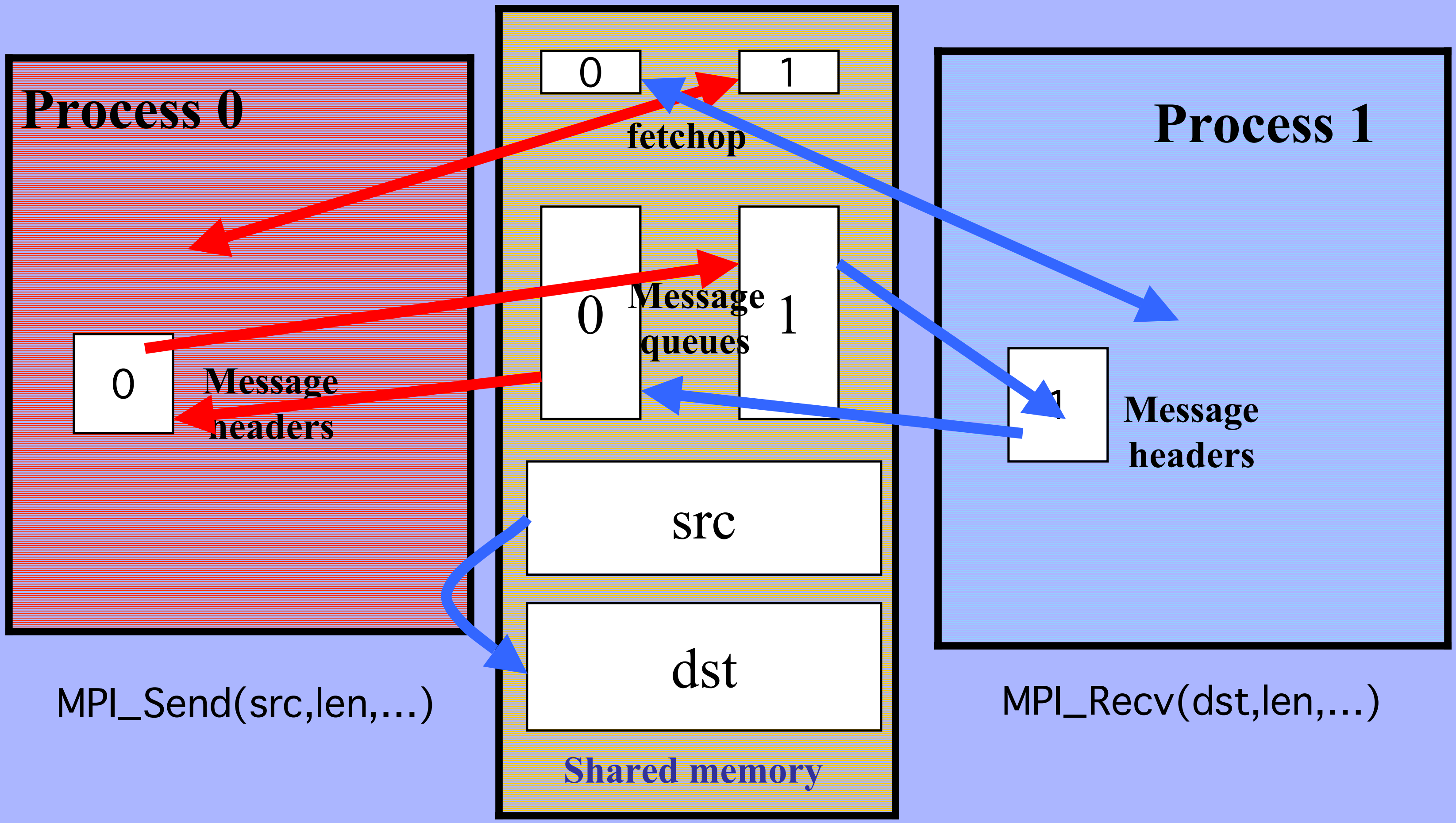
MPI_BUFS_PER_HOST - increase the setting of this shell
variable to reduce retries for
per host data buffers

Tunable Optimizations

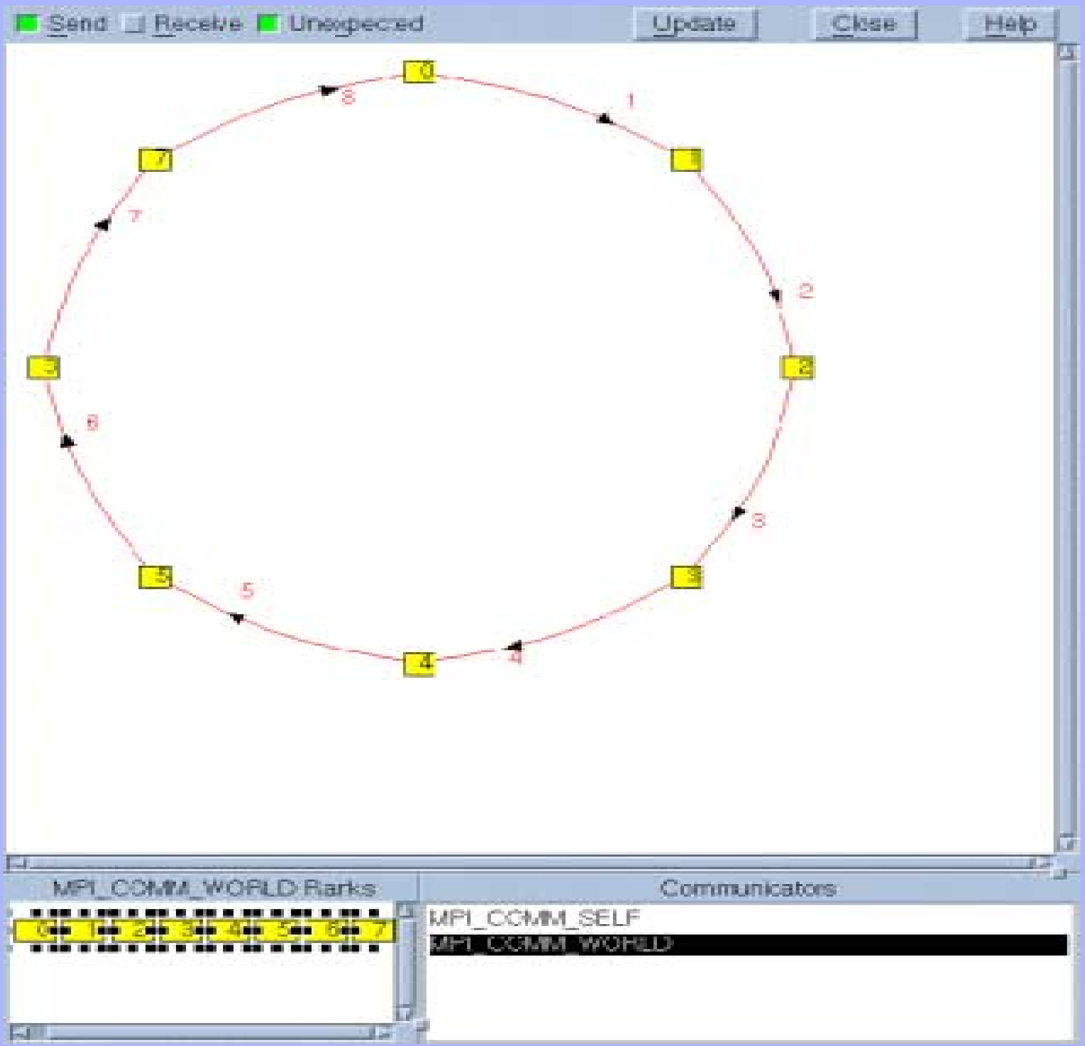
Single Copy optimization

- significantly improved bandwidth
- requires senders data to reside in globally accessible memory
  - not a requirement if using MPI_XPMEM_ON
- must be compiled -64
  - not a requirement if using MPI_XPMEM_ON
- need to set MPI_BUFFER_MAX
- works for simple and contiguous data types

MPI Message exchange using single copy (on host)



User Assumptions about MPI Send



Misuse of single copy can cause deadlock.

(Displayed with Totalview message queue viewer)

MPI Single Copy Bandwidths
(Shared Memory Point-to-Point)

protocol	O2K 400 MHz R12K	O3K 400 MHz R12K
buffered	80	174
single copy (globally accessible memory)	155	290
single copy (stack or private heap)*	-	276
single copy (BTE copy)	-	581

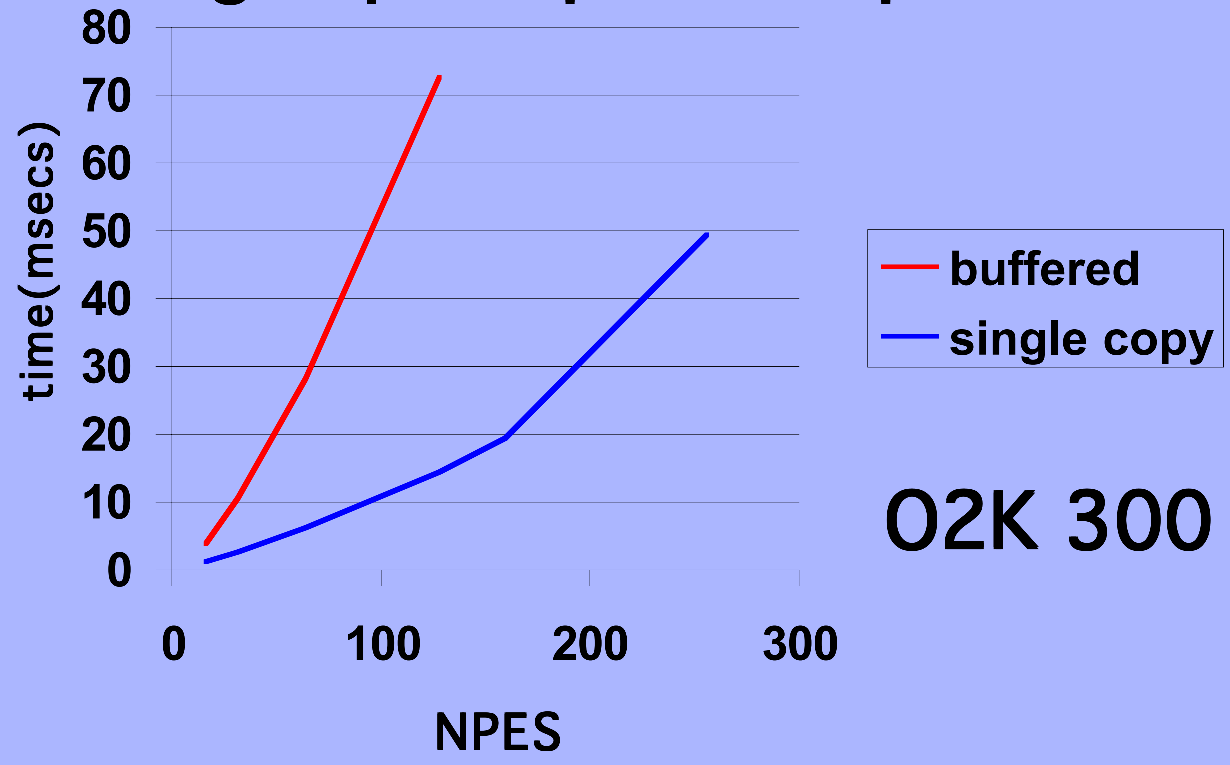
MPI_XPMEM_ON set

cache-aligned buffers

Bandwidths in MB/sec

Single Copy Speed-up

Alltoallv type communication pattern using explicit point to point calls



O2K 300 MHz

Tunable Optimizations

- Control NUMA Placement
- Assign Ranks to Physical CPUs
  - setenv MPI_DSM_CPULIST 0-15
  - setenv MPI_DSM_CPULIST 0,2,4,6
- Maps ranks 0 - N onto the physical CPUs in the order specified
- Useful only on quiet systems
- Easier than using dplace command
- Use MPI_DSM_MUSTRUN

Tunable Optimizations

- Control NUMA Placement
  - For bandwidth limited codes, it may be better to run fewer processes per node

    setenv MPI_DSM_PPM 1

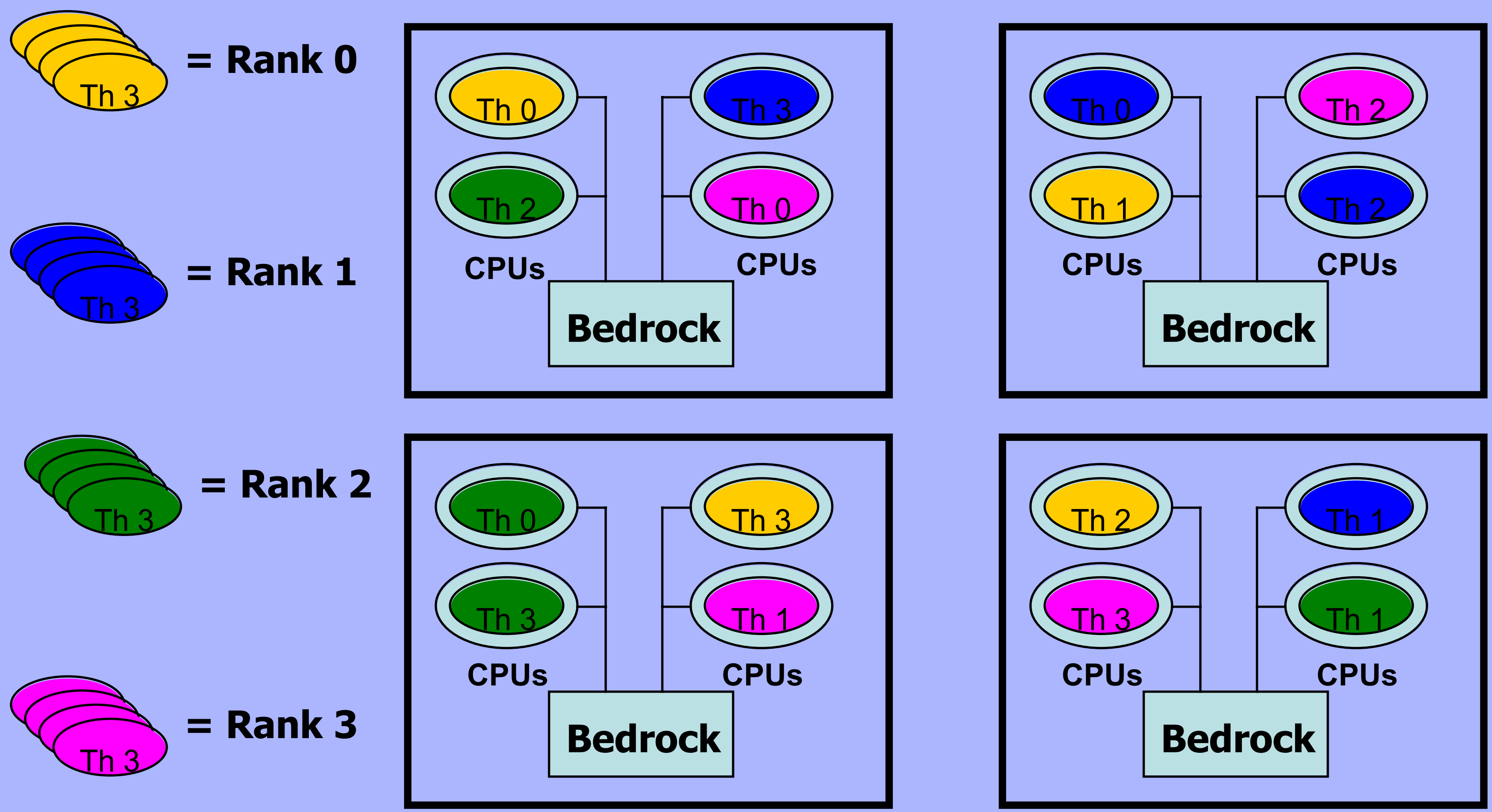
    - allows use of only 1 process per node board on Origin
    - if you have a memory bound code and extra CPUS, this may be a reasonable option
  - If lots of TLB misses set PAGESIZE_DATA env variable from 16K to 2 or 4M
  - dplace can be used for more explicit placement

    mpirun -np 4 dplace -place placement_file a.out

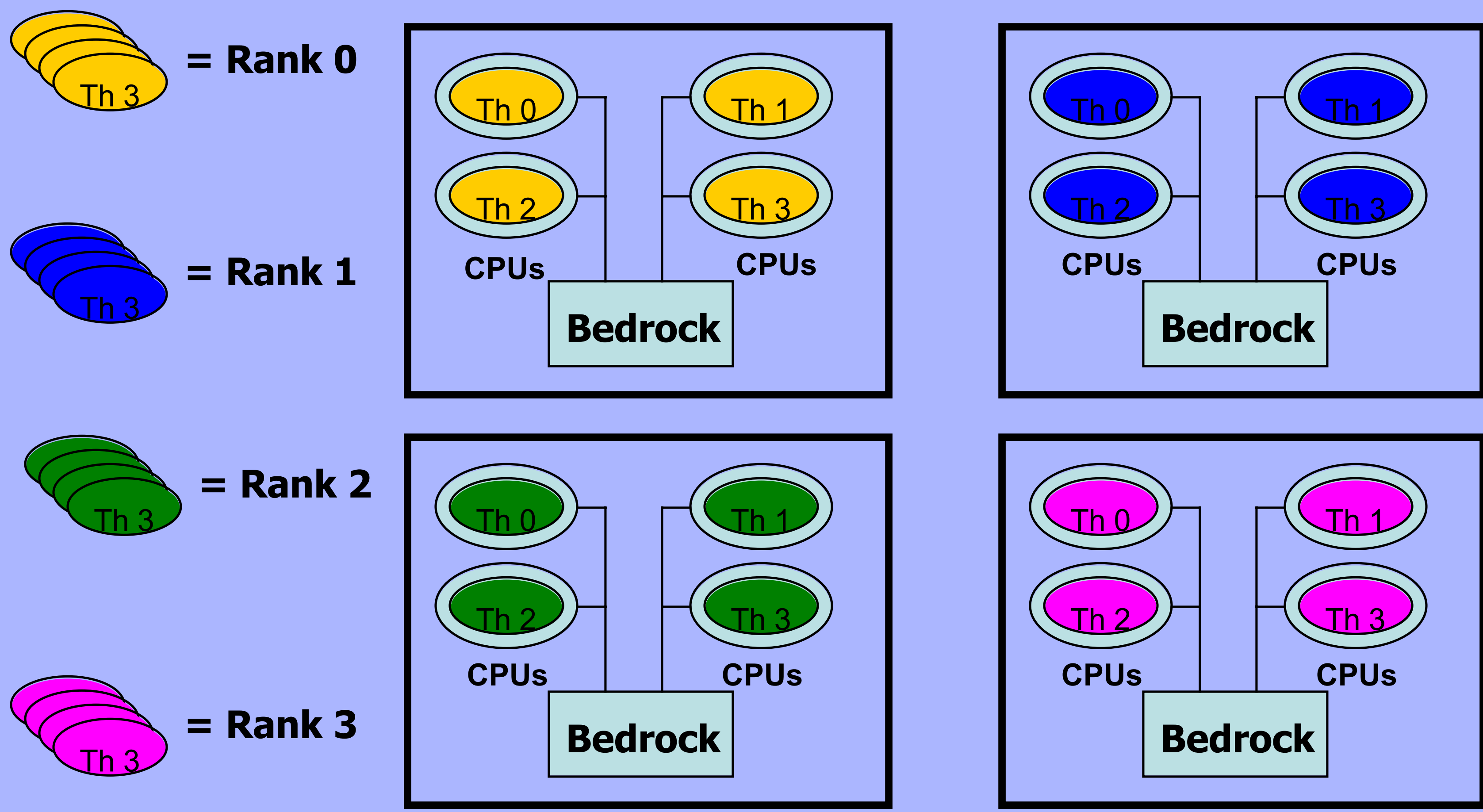
Tunable Optimizations

- Control NUMA Placement for MPI/OpenMP hybrid codes
  - set MPI_OPENMP_INTEROP
  - link with “-lmp -lmpi”
  - new in MPT 1.6

MPI and OpenMP: Random Placement



MPI and OpenMP: Coordinated Placement



Tips for Optimizing

- Single process optimization
- Avoid certain MPI constructs
- Reduce runtime variability
- MPI statistics (already mentioned)
- Use performance tools
- Use MPI profiling
- Instrument with timers

Single Process Optimization

- *Most techniques used for optimization of serial codes apply to message passing applications.*
- *Most tools available on IRIX for serial code optimization can work with MPI/SHMEM applications:*
  - *speedshop*
  - *perfex*

Avoid Certain MPI constructs

- Slow point to point calls
  - MPI_Ssend, MPI_Bsend
- Avoid MPI_Pack/MPI_Unpack
- Avoid MPI_ANY_SOURCE, MPI_ANY_TAG

Reduce runtime variability

- Don’t oversubscribe the system
- Use cpusets to divide the hosts cpus/memory between applications
- Control NUMA placement when benchmarking
- Use a batch scheduler
  - LSF(Platform)
  - PBS(Veridian)

Reduce runtime variability

- Tips for batch scheduler usage
- Use cpusets
- Avoid controlling load with memory limits
  - MPI and SHMEM use lots of virtual memory
  - SZ is actually virtual memory size, not memory usage!

Using performance analysis tools with MPI applications

Speedshop tools and perfex can be used with MPI applications

mpirun -np 4 ssrun [options] a.out

mpirun -np 4 perfex -mp [options] -o file a.out

MPI Statistics

MPI accumulates statistics concerning usage of internal resources

These counters can be accessed within an application via a set of SGI extensions to the MPI standard

See *MPI_SGI_stat* man page

MPI Statistics

setenv MPI_STATS

or

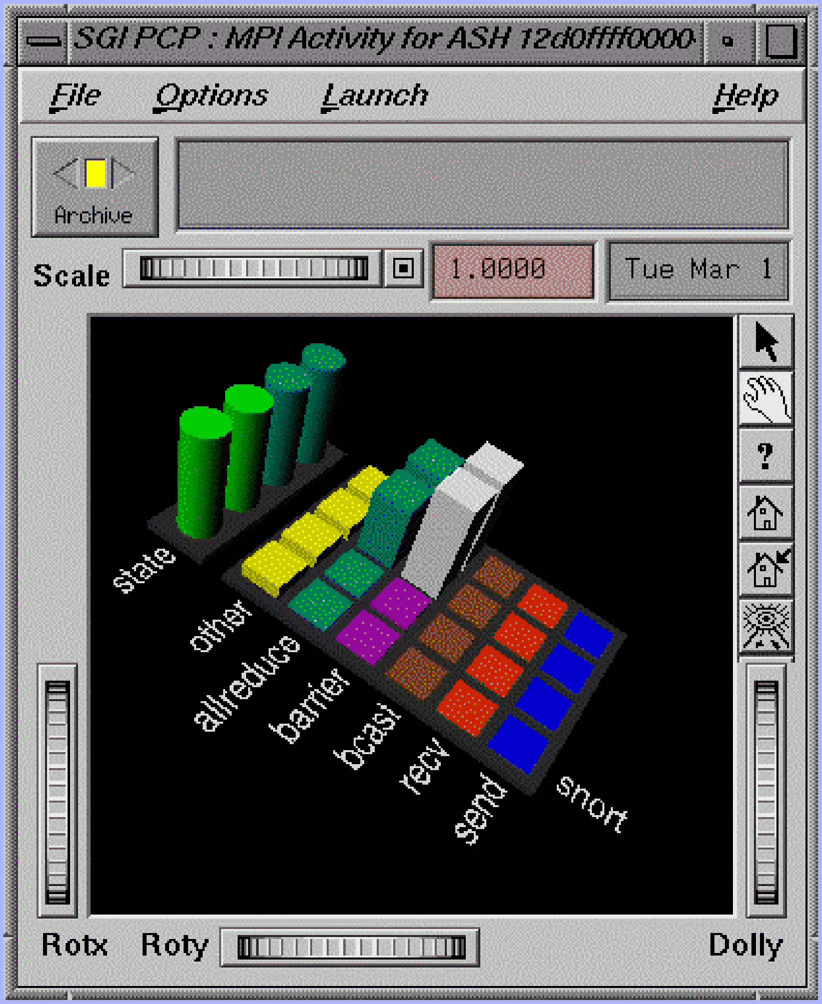
mpirun -stats -prefix “%g:” -np 8 a.out

```
3: *** Dumping MPI internal resource statistics...
3:
3:  0 retries allocating mpi PER_PROC headers for collective calls
3:  0 retries allocating mpi PER_HOST headers for collective calls
3:  0 retries allocating mpi PER_PROC headers for point-to-point calls
3:  0 retries allocating mpi PER_HOST headers for point-to-point calls
3:  0 retries allocating mpi PER_PROC buffers for collective calls
3:  0 retries allocating mpi PER_HOST buffers for collective calls
3:  0 retries allocating mpi PER_PROC buffers for point-to-point calls
3:  0 retries allocating mpi PER_HOST buffers for point-to-point calls
3:  0 send requests using shared memory for collective calls
3:  6357 send requests using shared memory for point-to-point calls
3:  0 data buffers sent via shared memory for collective calls
3:  2304 data buffers sent via shared memory for point-to-point calls
3:  0 bytes sent using single copy for collective calls
3:  0 bytes sent using single copy for point-to-point calls
3:  0 message headers sent via shared memory for collective calls
3:  6357 message headers sent via shared memory for point-to-point calls
3:  0 bytes sent via shared memory for collective calls
3:  15756000 bytes sent via shared memory for point-to-point calls
```

Profiling Tools

- Performance-CoPilot (/var/pcp/Tutorial/mpi.html)
  - mpivis/mpimon
  - no trace files necessary
- A number of third party profiling tools are available for use with MPI applications.
  - VAMPIR(www.pallas.de)
  - JUMPSHOT(MPICH tools)

Performance-CoPilot mpivis display



Performance-CoPilot mpimon display

Profiling MPI

- MPI has **PMPI***
  names

```
int MPI_Send(args)
{
    sendcount++;
    return PMPI_Send(args);
}
```

Instrument with timers

- *For large applications, a set of explicit functions to track coarse grain timing is very helpful*
- *MPI_Wtime timer*
  - *on O300, O2K and O3K single system runs the hardware counter is used*
  - *for multi-host runs uses gettimeofday*

Perfcatcher profiling library

- Source code that instruments many of the common MPI calls
- Only need to modify an RLD variable to link it in
- Does call counts and timings and writes a summary to a file upon completion
- Get it at http://freeware.sgi.com, under *Message-Passing Helpers*
- Use as a base and enhance with your own instrumentation

Perfcatcher profiling library sample output

- *Total job time 2.203333e+02 sec*
- *Total MPI processes 128*
- *Wtime resolution is 8.000000e-07 sec*

- *activity on process rank 0*
- *comm_rank calls 1 time 8.800002e-06*
- *get_count calls 0 time 0.000000e+00*
- *ibsend calls 0 time 0.000000e+00*
- *probe calls 0 time 0.000000e+00*
- *recv calls 0 time 0.00000e+00 avg datacnt 0 waits 0 wait time 0.00000e+00*
- *irecv calls 22039 time 9.76185e-01 datacnt 23474032 avg datacnt 1065*
- *send calls 0 time 0.000000e+00*
- *ssend calls 0 time 0.000000e+00*
- *isend calls 22039 time 2.950286e+00*
- *wait calls 0 time 0.00000e+00 avg datacnt 0*
- *waitall calls 11045 time 7.73805e+01 # of Reqs 44078 avg datacnt 137944*
- *barrier calls 680 time 5.133110e+00*
- *alltoall calls 0 time 0.0e+00 avg datacnt 0*
- *alltoallv calls 0 time 0.000000e+00*
- *reduce calls 0 time 0.000000e+00*
- *allreduce calls 4658 time 2.072872e+01*
- *bcast calls 680 time 6.915840e-02*
- *gather calls 0 time 0.000000e+00*
- *gatherv calls 0 time 0.000000e+00*
- *scatter calls 0 time 0.000000e+00*
- *scatterv calls 0 time 0.000000e+00*

- *activity on process rank 1*
- *...*

MPT Documentation

- Man pages
- *mpi*
- *mpirun*
- *intro_shmem*
- Relnotes
- Techpubs (techpubs.sgi.com)
- *Message Passing Toolkit: MPI Programmer's Manual*

MPI References

- MPI Standard
- *http://www.mpi-forum.org*
- MPICH related documentation
- *http://www.mcs.anl.gov/mpi/index.html*

- Texts
  - Using MPI: Portable Parallel Programming with the Message-Passing Interface*.* William Gropp, et. al., MIT Press.