Using Cray MPI: Tips for Development on the Cray XT5

Mark Fahey, Group Leader
mfahey@utk.edu

Glenn Brook
glenn-brook@tennessee.edu

NERSC/OLCF/NICS
Joint Cray XT5 Workshop
February 1 – 3, 2010

NICS Scientific Computing Group
Overview

• Assumes knowledge of MPI

• Assumes experience on a Cray XT (3, 4, or 5)
  – or a cluster with MPICH

• Suggests ways to improve MPI performance on Cray XT5
  – Some examples from both Jaguarpf and Kraken
  – No silver bullets
Outline

• MPT
• Environment Variables
• Rank Placement
• MPI Programming Techniques
• OpenMP
• Other
MPT – Cray’s MPI library

• Customization of MPICH-2 built atop Portals
  – Message protocols: Eager for short & Rendezvous for long

• Performs best when every message is expected prior to receipt, but ensuring such can be difficult or impossible

• Special handling of unexpected messages for both MPI and Portals to maximize performance and scalability

• Excessively bad application behavior can exhaust available resources for handling unexpected messages and events, resulting in application termination.
  – Short term fix: allocate additional resources via env variables
  – Long term fix: modify application to improve comm. behavior
MPT – Cray’s MPI library

• Use a recent version of MPT (default 3.5.0, current 4.0.1)
  – Significant improvements (e.g. allgatherv in 4.0.0 and later)

• Many users continue to set environment variables specifying buffer sizes due to previous versions of MPT
  – Current versions attempt to set the right buffer sizes at launch time based on job size rather than using static settings

• Suggestion: if you use env vars based on previous versions, try using recent versions w/o env vars

• Status:
  – Kraken: default 3.5.0
  – Jaguar: default 3.5.0
Environment Variables
MPI

• Next few slides will cover environment variables that are associated with MPI performance

• Find much of this information with “man mpi”
  – Yes, you should read the MPI man page!

• Default settings generally focus on attaining the best performance for most codes
  – Some codes may benefit from alternative settings

• The MPI environment can change between MPT versions
  – It is important to re-read the MPI man pages and other related documents provided by Cray
Environment Variables

MPICH_FAST_MEMCPY

• If set, enables an optimized memcpy routine in MPI. The optimized routine is used for local memory copies in the point-to-point and collective MPI operations.
  – This can help performance of some collectives that send large (256K and greater) messages.
    • Collectives are almost always faster
    • Speedup varies by message size
    • Example: If message sizes are known to be greater than 1 megabyte, then an optimized memcpy can be used that works well for large sizes, but may not work well for smaller sizes.
  – Default is not enabled (because there are a few cases that experience performance degradation)
  – Ex: PHASTA at 2048 processes: reduction from 262 s to 195 s
Environment Variables
MPICH_COLL_SYNC

• If set, a Barrier is performed at the beginning of each specified MPI collective function. This forces all processes participating in that collective to sync up before the collective can begin.
  – To enable this feature for all MPI collectives, set the value to 1. Default is off.

• Can be enabled for a selected list of MPI collectives

• There are rare examples where this helps
  – If the code has lots of collectives and MPI profiling shows imbalance (lots of sync time), this may help
  – Ex: PHASTA (CFD-turbulent flows) many MPI_Allreduce calls
    • At 2048 processes: reduction from 262 sec to 218 sec.
  – Ex: But slowed down NekTarG (CFD-Blood Flow) by about 7%
Environment Variables
MPICH_MPIIO_HINTS

• If set, overrides the default value of one or more MPI-IO hints. This also overrides any value set in the application code with calls to the MPI_Info_set routine.

• Hints are applied to the file when it is opened with an MPI_File_open() call.

• MPICH_MPIIO_HINTS_DISPLAY
  – If set, causes rank 0 in the participating communicator to display the names and values of all MPI-IO hints that are set for the file being opened with the MPI_File_open call.

Default settings:
PE 0: MPIIO hints for c2F.TILT3d.hdf5:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cb_buffer_size</td>
<td>16777216</td>
</tr>
<tr>
<td>romio_cb_read</td>
<td>automatic</td>
</tr>
<tr>
<td>romio_cb_write</td>
<td>automatic</td>
</tr>
<tr>
<td>cb_nodes</td>
<td>#nodes/8</td>
</tr>
<tr>
<td>romio_no_indep_rw</td>
<td>false</td>
</tr>
<tr>
<td>ind_rd_buffer_size</td>
<td>4194304</td>
</tr>
<tr>
<td>ind_wr_buffer_size</td>
<td>524288</td>
</tr>
<tr>
<td>romio_ds_read</td>
<td>automatic</td>
</tr>
<tr>
<td>romio_ds_write</td>
<td>automatic</td>
</tr>
<tr>
<td>direct_io</td>
<td>false</td>
</tr>
<tr>
<td>cb_config_list</td>
<td>*:1</td>
</tr>
</tbody>
</table>
Environment Variables
MPICH_MPIIO_HINTS (cont.)

Examples:

• Syntax
  
  – export MPICH_MPIIO_HINTS=data.hdf5:direct_io=true

• For FlashIO at 5000 processes writing out 500MB per MPI thread, the following improved performance:

  romio_cb_write = "ENABLE"
  romio_cb_read = "ENABLE"
  cb_buffer_size = 32M
  
  – When enabled, all collective reads/writes will use collective buffering. When disabled, all collective reads/writes will be serviced with individual operations by each process. When set to automatic, ROMIO will use heuristics to determine when to enable the optimization.

• For S3D at 10K cores:

  romio_ds_write = 'disable' - specifies if data sieving is to be done on read. Data sieving is a technique for efficiently accessing noncontiguous regions of data
  romio_no_indep_rw = 'true' - specifies whether deferred open is used.
  
  – Romio docs say that this indicates no independent read or write operations will be performed. This can be used to limit the number of processes that open the file.
Environment Variables
MPICH_MPIIO_CB_ALIGN

- If set to 1, new algorithms that take into account physical I/O boundaries and the size of I/O requests are used to determine how to divide the I/O workload when collective buffering is enabled.
  - This can improve performance by causing the I/O requests of each collective buffering node (aggregator) to start and end on physical I/O boundaries and by preventing more than one aggregator making reference to any given stripe on a single collective I/O call.
  - If set to zero or not defined, the algorithms used prior to MPT release 3.1 are used.
  - Default: not set
Environment Variables

MPICH_ENV_DISPLAY

• If set, causes rank 0 to display all MPICH environment variables and their current settings at MPI initialization time.

• Default: Not enabled.

• Useful for debugging purposes.

• MPICH_VERSION_DISPLAY - displays the version of cray MPT being used
Environment Variables

MPICH_SMP_OFF

• If set, disable the on-node SMP device and use the Portals device for all MPI message transfers

• Use in a rare cases where code benefits from using Portals matching instead of MPI matching.

• Default: Not enabled.

• Useful for debugging reproducibility issues.
Environment Variables
Buffer Sizes

Based on experience running S3D up to 150,000 cores

- **MPICH_UNEX_BUFFER_SIZE** often runs out of space
  - When this buffer size cannot be increased sufficiently, MPICH_MAX_SHORT_MSG_SIZE should be reduced.
  - Making this smaller switches the threshold for short vs long messages. Long messages are not received unless they are expected (a receive is already posted).
  - There is a performance penalty due to reducing the max short message size, but it will get it working.

- **MPICH_PTL_UNEX_EVENTS** and **MPICH_PTL_OTHER_EVENTS** have a low default value.
  - They are almost never adequate for large jobs. The following are good at O(10 thousand) cores.
    - `MPICH_PTL_UNEX_EVENTS=400000`
    - `MPICH_PTL_OTHER_EVENTS=100000`

- When an error that says 'MPI_MSGS_PER_PROC' is not sufficient is received, increase **MPICH_MSGS_PER_PROC**. It is an error in the error message.

- Buffer size variables can be set using k, M and G - Instead of having to type powers of 2 or count zeroes.
  - `% export MPICH_UNEX_BUFFER_SIZE=1G`  #sets it to 1 gigabyte
Environment Variables

MPICH_PTL_MATCH_OFF

• If set, disables registration of receive requests with portals.
  – Setting this allows MPI to perform the message matching for the portals device. It may be beneficial to set this variable when an application exhausts portals internal resources and for latency-sensitive applications.
  – Example: Used for LS-DYNA

MPICH_PTL_SEND_CREDITS

• Enables flow control to prevent the Portals event queue from being overflowed.
  – Value of ‘-1’ should prevent queue overflow in any situation
  – Should only be used as needed, as flow control will result in less optimal performing code. If the Portals unexpected event queue can not be increased enough, then flow control may need to be enabled.
Environment Variables
MPICH_PTL_MATCH_OFF

• Case where MPICH_PTL_MATCH_OFF fixed an MPI problem
  PtlMEMDPost() failed : PTL_NO_SPACE

• For this, try MATCH, OTHER_EVENTS or SEND_CREDITS env var
  [43] MPICH PtlEQPoll error (PTL_EQ_DROPPED): An event was dropped on the OTHER EQ handle. Try increasing the value of env var MPICH_PTL_OTHER_EVENTS (cur size is 2048).

  aborting job:
  PtlEQPoll/PtlEQGet error

  – Attempts to increase OTHER_EVENTS did not help though (in this case)
## Portals Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description / Cause</th>
<th>Suggested Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTL_PT_NO_ENTRY</td>
<td>Memory mapping error / improper stack initialization</td>
<td>Request refund and resubmit job</td>
</tr>
<tr>
<td>PTL_NAL_FAILED</td>
<td>Network layer error / node or network failure</td>
<td>Request refund and resubmit job</td>
</tr>
<tr>
<td>PTL_EQ_DROPPED</td>
<td>Event dropped from queue / insufficient space in queue</td>
<td>Increase resources with environment variables, change application communication profile</td>
</tr>
<tr>
<td>PTL_SEGV</td>
<td>Invalid user address supplied to portals</td>
<td>Fix invalid pointers in application code</td>
</tr>
<tr>
<td>PTL_PT_VAL_FAILED</td>
<td>Invalid address / invalid buffer parameter in MPI</td>
<td>Fix invalid pointers in application code (MPI)</td>
</tr>
<tr>
<td>PTL_NO_SPACE</td>
<td>Insufficient memory for internal buffers</td>
<td>Reduce application memory requirements on nodes, set MPICH_PTL_MATCH_OFF</td>
</tr>
</tbody>
</table>
Rank Placement

• In some cases, changing how the processes are laid out on the machine may affect performance by relieving synchronization/imbalance time.

• The default is currently SMP-style placement. This means that for a multi-node core, sequential MPI ranks are placed on the same node.
  – In general, MPI codes perform better using SMP placement - Nearest neighbor
  – Collectives have been optimized to be SMP aware

• For example, a 12-process job launched on a XT5 node with 2 hex-core processors would be placed as:
  
<table>
<thead>
<tr>
<th>PROCESSOR</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANK</td>
<td>0,1,2,3,4,5</td>
<td>6,7,8,9,10,11</td>
</tr>
</tbody>
</table>
Rank Placement

• The default ordering can be changed using the following environment variable:

  MPICH_RANK_REORDER_METHOD

• These are the different values that you can set it to:
  0: Round-robin placement – Sequential ranks are placed on the next node in the list. Placement starts over with the first node upon reaching the end of the list.
  1: SMP-style placement – Sequential ranks fill up each node before moving to the next.
  2: Folded rank placement – Similar to round-robin placement except that each pass over the node list is in the opposite direction of the previous pass.
  3: Custom ordering. The ordering is specified in a file named MPICH_RANK_ORDER.

• When is this useful?
  – Point-to-point communication consumes a significant fraction of program time and a load imbalance detected
  – Also shown to help for collectives (alltoall) on subcommunicators (GYRO)
  – Spread out IO across nodes (POP)
Rank Order and CrayPAT

• One can also use the CrayPat performance measurement tools to generate a suggested custom ordering.
  – Available if MPI functions traced (-g mpi or –O apa)
  – pat_build –O apa my_program
    • see Examples section of pat_build man page

• pat_report options:
  – mpi_sm_rank_order
    • Uses message data from tracing MPI to generate suggested MPI rank order. Requires the program to be instrumented using the pat_build -g mpi option.
  – mpi_rank_order
    • Uses time in user functions, or alternatively, any other metric specified by using the -s mro_metric options, to generate suggested MPI rank order.
Reordering Workflow

- module load xt-craypat
- Rebuild your code
- pat_build –O apa a.out
- Run a.out+pat
- pat_report –Ompi_sm_rank_order a.out+pat+...sdt/ > pat.report
- Creates MPICH_RANK_REORDER_METHOD.x file
- Then set env var MPICH_RANK_REORDER_METHOD=3 AND
- Link the file MPICH_RANK_ORDER.x to MPICH_RANK_ORDER
- Rerun code
CrayPAT example

Table 1: Suggested MPI Rank Order

<table>
<thead>
<tr>
<th>Rank Order</th>
<th>USER Samp</th>
<th>SMP</th>
<th>USER Samp</th>
<th>SMP</th>
<th>Max Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>17062</td>
<td>97.6%</td>
<td>16907</td>
<td>100.0%</td>
<td>832,328,820,797,113,478,898,600</td>
</tr>
<tr>
<td>2</td>
<td>17213</td>
<td>98.4%</td>
<td>16907</td>
<td>100.0%</td>
<td>53,202,309,458,565,714,821,970</td>
</tr>
<tr>
<td>0</td>
<td>17282</td>
<td>98.8%</td>
<td>16907</td>
<td>100.0%</td>
<td>53,181,309,437,565,693,821,949</td>
</tr>
<tr>
<td>1</td>
<td>17489</td>
<td>100.0%</td>
<td>16907</td>
<td>100.0%</td>
<td>0,1,2,3,4,5,6,7</td>
</tr>
</tbody>
</table>

This suggests that
1. the custom ordering “d” might be the best
2. Folded-rank next best
3. Round-robin 3rd best
4. Default ordering last
Reordering example

GYRO

- GYRO 8.0
  - B3-GTC problem with 1024 processes
- Run with alternate MPI orderings
  - Custom: profiled with with –O apa and used reordering file MPICH_RANK_REORDER.d

<table>
<thead>
<tr>
<th>Reorder method</th>
<th>Comm. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>11.26s</td>
</tr>
<tr>
<td>0 – round-robin</td>
<td>6.94s</td>
</tr>
<tr>
<td>2 – folded-rank</td>
<td>6.68s</td>
</tr>
<tr>
<td>d-custom from apa</td>
<td>8.03s</td>
</tr>
</tbody>
</table>

CrayPAT suggestion almost right!
Reordering example
TGYRO

• TGYRO 1.0
  – Steady state turbulent transport code using GYRO, NEO, TGLF components

• ASTRA test case
  – Tested MPI orderings at large scale
  – Originally testing weak-scaling, but found reordering very useful

<table>
<thead>
<tr>
<th>Reorder method</th>
<th>TGYRO wall time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20480</td>
</tr>
<tr>
<td>Default</td>
<td>99m</td>
</tr>
<tr>
<td>Round-robin</td>
<td>66m</td>
</tr>
</tbody>
</table>

Huge win!
MPI Programming Techniques

Pre-posting receives

• If possible, pre-post receives before the matching sends
  – Optimization technique for all MPICH installations (not just MPT)
  – Put as much computation as possible between receive-send pair

• Do not go crazy pre-posting receives. You can (and will) overrun the resources available to Portals.

• Even an IBM manual states:
  – “well-written applications try to pre-post their receives.” And they also warn about posting too many.

• Code example
  – Halo update – with four buffers (n, s, e, w), post all receive requests as early as possible. Makes a big difference on CNL (not as important on Catamount).
MPI Programming Techniques

Overlapping communication with computation

• Corollary of pre-posting receives

• Use non-blocking send/recvs to overlap communication with computation whenever possible

• In some cases, it may be better to replace collective operations with point-to-point communications to overlap communication with computation
  
  – Caution: Do not blindly reprogram every collective by hand
  – Concentrate on the parts of your algorithm with significant amounts of computation that can overlap with the point-to-point communications when a [blocking] collective is replaced
MPI Programming Techniques

Example: 9-pt stencil pseudo-code

Basic

9-pt computation

Update ghost cell boundaries

East/West IRECV, ISEND, WAITALL
North/South IRECV, ISEND, WAITALL

Maximal Irecv preposting

Prepost all IRECV

9-pt computation

Update ghost cell boundaries

East/West ISEND, Wait on E/W IRECV only
North/South ISEND, Wait on the rest

*Makes use of temporary buffers
Example: 9-pt stencil update

! compute stencil
...

! update ghost cell boundaries.
! East/West

MPI_IRECV(XOUT(1,1), 1, mpi_ew_type, nbr_west, mpitag_wshift, COMM_OCN, request(3))

MPI_IRECV(XOUT(iphys_e+1,1), 1, mpi_ew_type, nbr_east, mpitag_eshift, COMM_OCN, request(4))

MPI_ISEND(XOUT(iphys_e+1-num_ghost_cells,1), 1, mpi_ew_type, nbr_east, mpitag_wshift, COMM_OCN, request(1))

MPI_ISEND(XOUT(iphys_b,1), 1, mpi_ew_type, nbr_west, mpitag_eshift, COMM_OCN, request(2))

MPI_WAITALL(4, request, status)

! North/South

MPI_IRECV(XOUT(1,jphys_e+1), 1, mpi_ns_type, nbr_north, mpitag_nshift, COMM_OCN, request(3))

MPI_IRECV(XOUT(1,1), 1, mpi_ns_type, nbr_south, mpitag_sshift, COMM_OCN, request(4))

MPI_ISEND(XOUT(1,jphys_b), 1, mpi_ns_type, nbr_south, mpitag_sshift, COMM_OCN, request(1))

MPI_ISEND(XOUT(1,jphys_e+1-num_ghost_cells), 1, mpi_ns_type, nbr_north, mpitag_nshift, COMM_OCN, request(2))

MPI_WAITALL(4, request, status)

! Prepost receive requests

MPI_IRECV(buf_west_rcv, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_west, & mpitag_wshift, COMM_OCN, request(7))

MPI_IRECV(buf_east_rcv, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_east, mpitag_eshift, COMM_OCN, request(8))

MPI_IRECV(XOUT(1,jphys_e+1), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_north, mpitag_nshift, COMM_OCN, request(5))

MPI_IRECV(XOUT(1,1), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_south, mpitag_sshift, COMM_OCN, request(6))

! compute stencil
...

! send east-west boundary info

MPI_ISEND(buf_east_snd, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_east, mpitag_wshift, COMM_OCN, request(1))

MPI_ISEND(buf_west_snd, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_west, mpitag_eshift, COMM_OCN, request(2))

MPI_WAITALL(2, request(7), status_wait)

! send north-south boundary info

MPI_ISEND(XOUT(1,jphys_e+1-num_ghost_cells), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_north, mpitag_nshift, COMM_OCN, request(3))

MPI_ISEND(XOUT(1,jphys_b), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_south, mpitag_sshift, COMM_OCN, request(4))

MPI_WAITALL(6, request, status_wait)
MPI Programming Techniques

Aggregating data

• For very small buffers, aggregate data into fewer MPI calls (especially for collectives)
  – 1 all-to-all with an array of 3 reals is clearly better than 3 all-to-alls with 1 real
  – Do not aggregate too much. The MPI protocol switches from a short (eager) protocol to a long message protocol using a receiver pull method once the message is larger than the eager limit. This limit is by default 128000 bytes, but it can be changes with the MPICH_MAX_SHORT_MSG_SIZE environment variable. The optimal size for messages most of the time is less than the eager limit.

• Example – DNS
  – Turbulence code (DNS) replaced 3 AllGatherv’s by one with a larger message resulting in 25% less runtime for one routine.
MPI Programming Techniques

Aggregating data: Example from CFD

***Original***

```c
for (index = 0; index < No; index++){
    double tmp;
    tmp = 0.0;
    out_area[index] = Bndry_Area_out(A, labels[index]);
    gdsum(&outlet_area[index], 1, &tmp);
}
for (index = 0; index < Ni; index++){
    double tmp;
    tmp = 0.0;
    in_area[index] = Bndry_Area_in(A, labels[index]);
    gdsum(&inlet_area[index], 1, &tmp);
}

void gdsum (double *x, int n, double *work)
{
    register int i;
    MPI_Allreduce (x, work, n, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    /* *x = *work; */
    dcopy(n,work,1,x,1);
    return;
}
```

***Improved***

```c
for (index = 0; index < No; index++){
    out_area[index] = Bndry_Area_out(A, labels[index]);
    gdsum(&outlet_area[index], No, &tmp);
}
for (index = 0; index < Ni; index++){
    in_area[index] = Bndry_Area_in(A, labels[index]);
    gdsum(&inlet_area[index], Ni, &tmp);
}

void gdsum (double *x, int n, double *work)
{
    register int i;
    MPI_Allreduce (x, work, n, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    /* *x = *work; */
    dcopy(n,work,1,x,1);
    return;
}
```
OpenMP

• When does it pay to add/use OpenMP in my MPI code?
  – Add/use OpenMP when code is network bound
  – As collective and/or point-to-point time increasingly becomes a problem, use threading to keep number of MPI processes per node to a minimum
  – Be careful adding OpenMP to memory bound codes
    • Can hurt performance
  – It is code/situation dependent!
OpenMP aprun depth

• Must get “aprun –d” correct
  – -d (depth) Specifies the number of threads (cores) for each process. ALPS allocates the number of cores equal to depth times processes.
  – The default depth is 1. This option is used in conjunction with the OMP_NUM_THREADS environment variable.
  – Also used to get more memory per process
    • Get 1 or 2 GB limit by default (machine dependent)
  – Many have gotten this wrong, so it is important to understand how to use it properly!
    • If you do not do it correctly, a hybrid OpenMP/MPI code can get multiple threads spawned on the same core which can be disastrous.
OpenMP
aprune depth (cont.)

% setenv OMP_NUM_THREADS 4

% aprun -n 4 -q ./omp1 | sort
Hello from rank 0, thread 0, on nid00291. (core affinity = 0)
Hello from rank 0, thread 1, on nid00291. (core affinity = 0)
Hello from rank 0, thread 2, on nid00291. (core affinity = 0)
Hello from rank 0, thread 3, on nid00291. (core affinity = 0)
Hello from rank 1, thread 0, on nid00291. (core affinity = 1)
Hello from rank 1, thread 1, on nid00291. (core affinity = 1)
Hello from rank 1, thread 2, on nid00291. (core affinity = 1)
Hello from rank 1, thread 3, on nid00291. (core affinity = 1)
Hello from rank 2, thread 0, on nid00291. (core affinity = 2)
Hello from rank 2, thread 1, on nid00291. (core affinity = 2)
Hello from rank 2, thread 2, on nid00291. (core affinity = 2)
Hello from rank 2, thread 3, on nid00291. (core affinity = 2)
Hello from rank 3, thread 0, on nid00291. (core affinity = 3)
Hello from rank 3, thread 1, on nid00291. (core affinity = 3)
Hello from rank 3, thread 2, on nid00291. (core affinity = 3)
Hello from rank 3, thread 3, on nid00291. (core affinity = 3)

% aprun -n 4 -d 4 -q ./omp | sort
Hello from rank 0, thread 0, on nid00291. (core affinity = 0)
Hello from rank 0, thread 1, on nid00291. (core affinity = 1)
Hello from rank 0, thread 2, on nid00291. (core affinity = 2)
Hello from rank 0, thread 3, on nid00291. (core affinity = 3)
Hello from rank 1, thread 0, on nid00291. (core affinity = 4)
Hello from rank 1, thread 1, on nid00291. (core affinity = 5)
Hello from rank 1, thread 2, on nid00291. (core affinity = 6)
Hello from rank 1, thread 3, on nid00291. (core affinity = 7)
Hello from rank 2, thread 0, on nid00292. (core affinity = 0)
Hello from rank 2, thread 1, on nid00292. (core affinity = 1)
Hello from rank 2, thread 2, on nid00292. (core affinity = 2)
Hello from rank 2, thread 3, on nid00292. (core affinity = 3)
Hello from rank 3, thread 0, on nid00292. (core affinity = 4)
Hello from rank 3, thread 1, on nid00292. (core affinity = 5)
Hello from rank 3, thread 2, on nid00292. (core affinity = 6)
Hello from rank 3, thread 3, on nid00292. (core affinity = 7)
OpenMP – Scope all variables!

```c
int i, j, k;

#pragma omp parallel shared(t, new, old, nrl, dt, NR, NC, NITER) private(d)
#pragma omp for schedule(runtime) nowait
for (i = 2; i <= nrl-1; i++)
for (j = 1; j <= NC; j++){
    t[*new][i][j] = 0.25 *
    (t[old][i+1][j] + t[old][i-1][j] +
    t[old][i][j+1] + t[old][i][j-1]);
    d = MAX(fabs(t[*new][i][j] -
             t[old][i][j]), d);
}
```

In this particular case, the homb benchmark got wrong answers and failed to scale well when using PGI and Pathscale.
Other IO

• Also note that sometimes IO (especially at scale) causes scalability issues
  – For example, cleaning up some writes improved weak scaling of the CFD code NektarG from 70% to 95% at 1K to 8K cores
Conclusions/Last words

- Env vars are an easy way to improve performance
  - They may not always be applicable

- Good MPI programming practices are beneficial
  - Pre-posting receives important
  - Aggregating data

- Rank reordering can significantly improve performance

- Use depth option with OpenMP or for extra memory

- Be cognizant of how IO affects your overall scalability

- Some of this may not show a benefit at <1K processes, but it can reap huge gains at 10K to 100K processes

- This will become a “MPI Tips” webpage
References

• Best reference on MPT environment variables to date:
  – Geir Johansen, “Managing Cray XT MPI Runtime Environment Variables to Optimize and Scale Applications”, presentation and paper, CUG 2008

• man mpi

• man aprun