CRAY
THE SUPERCOMPUTER COMPANY

XE6 Porting & Tuning Tips

XE6 – Where to start

- **For most users and applications, using default settings work very well**
- **For users who want to experiment to get the best performance they can, the following presentation gives you some information on compilers and settings to try**
  - While it doesn't cover absolutely everything, the presentation tries to address some of the tunable parameters which we have found to provide increased performance in the situations discussed

1. Load the proper xtpe-<arch>

xtpe-mc12

If no module is loaded, and no 'arch' specified in the compiler options, the compilers default to the node type on which the compiler is running: Which may not be the same as the compute nodes !

2. Use the best Compiler

The *best* compiler is not the same for every application

Compiler Choices – Relative Strengths

…from Cray's Perspective

- **PGI – Very good Fortran, okay C and C++**
  - Good vectorization
  - Good functional correctness with optimization enabled
  - Good manual and automatic prefetch capabilities
  - Very interested in the Linux HPC market, although that is not their only focus
  - Excellent working relationship with Cray, good bug responsiveness
- **Pathscale – Good Fortran, C, probably good C++**
  - Outstanding scalar optimization for loops that do not vectorize
  - Fortran front end uses an older version of the CCE Fortran front end
  - OpenMP uses a non-pthreads approach
  - Scalar benefits will not get as much mileage with longer vectors
- **(Not NERSC supported) Intel – Good Fortran, excellent C and C++ (if you ignore vectorization)**
  - Automatic vectorization capabilities are modest, compared to PGI and CCE
  - Use of inline assembly is encouraged
  - Focus is more on best speed for scalar, non-scaling apps
  - Tuned for Intel architectures, but actually works well for some applications on AMD

Compiler Choices – Relative Strengths

…from Cray's Perspective

- **GNU so-so Fortran, outstanding C and C++ (if you ignore vectorization)**
  - Obviously, the best for gcc compatability
  - Scalar optimizer was recently rewritten and is very good
  - Vectorization capabilities focus mostly on inline assembly
  - Note the last three releases have been incompatible with each other (4.3, 4.4, and 4.5) and required recompilation of Fortran modules
- **CCE – Outstanding Fortran, very good C, and okay C++**
  - Very good vectorization
  - Very good Fortran language support; only real choice for Coarrays
  - C support is quite good, with UPC support
  - Very good scalar optimization and automatic parallelization
  - Clean implementation of OpenMP 3.0, with tasks
  - Sole delivery focus is on Linux-based Cray hardware systems
  - Best bug turnaround time (if it isn't, let us know!)
  - Cleanest integration with other Cray tools (performance tools, debuggers, upcoming productivity tools)
  - No inline assembly support

Recommended CCE Compilation Options

- **Use default optimization levels**
  - It's the equivalent of most other compilers –O3 or –fast
- **Use –O3,fp3 (or –O3 –hfp3, or some variation)**
  - -O3 only gives you slightly more than –O2
  - -hfp3 gives you a lot more floating point optimization, esp. 32-bit
- **If an application is intolerant of floating point reassociation, try a lower –hfp number – try –hfp1 first, only –hfp0 if absolutely necessary**
  - Might be needed for tests that require strict IEEE conformance
  - Or applications that have 'validated' results from a different compiler
- **Do not suggest using –Oipa5, -Oaggress, and so on – higher numbers are not always correlated with better performance**
- **Compiler feedback: -rm (Fortran) -hlist=m (C)**
- **If you know you don't want OpenMP: -xomp or -Othread0**
- **man crayftn; man craycc ; man crayCC**

Starting Points for the other Compilers

- **PGI**
  - -fast –Mipa=fast(,safe)
  - If you can be flexible with precision, also try -Mfprelaxed
  - Compiler feedback: -Minfo=all -Mneginfo
  - man pgf90; man pgcc; man pgCC; or pgf90 -help
- **Pathscale**
  - -Ofast Note: this is a little looser with precision than other compilers
  - Compiler feedback: -LNO:simd_verbose=ON
  - man eko ("Every Known Optimization")
- **GNU**
  - -O3 –ffast-math –funroll-loops
  - Compiler feedback: -ftree-vectorizer-verbose=2
  - man gfortran; man gcc; man g++
- **Intel**
  - -fast
  - Compiler feedback:
  - man ifort; man icc; man iCC

3. Library Loading

Use the xtpe-mc12 module and it is all automatic

The OpenMP threaded BLAS/LAPACK library is the default if the xtpe-mc12 module is loaded. The serial version is used if 'OMP_NUM_THREADS' is not set or set to 1.

4. MPT Environment Variables

Experiment with the Environment Variable Grab Bag

MPICH_GNI_DMAPP_INTEROP

- Only relevant for mixed MPI/SHMEM/UPC/CAF codes
- Normally want to leave enabled so MPICH2 and DMAPP can share the same memory registration cache
- May have to disable for codes that call shmem_init after MPI_Init.
- May have to set to disable if one gets a traceback like this:

```
Rank 834 Fatal error in MPI_Alltoall: Other MPI error, error stack:
MPI_Alltoall(768).....: MPI_Alltoall(sbuf=0x2aab9c301010,
scount=2596, MPI_DOUBLE, rbuf=0x2aab7ae01010, rcount=2596, MPI_DOUBLE,
comm=0x84000004) failed
MPIR_Alltoall(469).....:
MPIC_Isend(453).....:
MPID_nem_lmt_RndvSend(102).....:
MPID_nem_gni_lmt_initiate_lmt(580).....: failure occurred while attempting to
send RTS packet
MPID_nem_gni_iStartContigMsg(869).....:
MPID_nem_gni_iSendContig_start(763).....:
MPID_nem_gni_send_conn_req(626).....:
MPID_nem_gni_progress_send_conn_req(193):
MPID_nem_gni_smsg_mbox_alloc(357).....:
MPID_nem_gni_smsg_mbox_block_alloc(268).: GNI_MemRegister
GNI_RC_ERROR_RESOURCE)
```

MPICH_GNI_MAX_EAGER_MSG_SIZE

- Default is 8192 bytes
- Maximum size message that can go through the eager protocol.
- May help for apps that are sending medium size messages, and do better when loosely coupled. Does application have a large amount of time in MPI_Waitall? Setting this environment variable higher may help.
- Max value is 131072 bytes.
- Remember for this path it helps to pre-post receives if possible.
- Note that a 40-byte CH3 header is included when accounting for the message size.

MPICH_GNI_NDREG_LAZYMEM

- Default is enabled
- Controls whether or not to use a lazy memory deregistration policy inside UDREG.
- May help for apps that
  - Transferring a lot of messages via LMT path
  - And use 4KB pages
- Disabling results in a significant drop in measured bandwidth for large transfers ~40-50%.
- May be better off trying to use large pages than setting this to 'disabled'.

MPICH_GNI_NUM_BUFS

- Default is 64 32K buffers (2M total)
- Controls number of 32K DMA buffers available for each rank to use in the Eager protocol described earlier
- May help to modestly increase. But other resources constrain the usability of a large number of buffers.

MPICH_GNI_RDMA_THRESHOLD

- Default value is 1024.
- Controls the threshold at which the GNI netmod switches from using FMA for RDMA read/write operations to using the BTE.
- Since BTE is managed in the kernel, BTE initiated RDMA requests can progress even if the application isn't in MPI, allowing possibly for slightly better chances of getting some overlap of communication with computation.
- Owing to Opteron/HT quirks, the BTE is often better for moving data to/from memories that are farther from the Gemini.

MPICH_SMP_SINGLE_COPY_SIZE

- Default value is 8192 bytes.
- Specifies threshold at which the shared memory channel switches to a single-copy (XPMEM) protocol for intra-node messages from a double copy protocol.

MPICH_SMP_SINGLE_COPY_OFF

- Starting with MPT 5.0.2, the default is single copy via XPMEM is enabled (= '0'). In older versions single copy via XPMEM is disabled (= '1').
- Specifies whether or not to use a XPMEM-based single-copy protocol for intra-node messages of size MPICH_SMP_SINGLE_COPY_SIZE bytes or larger.

5. Tweak the MPICH_GNI_MAX_EAGER_MSG_SIZE

This allows for more async message transfer.

But the additional copy on the receiving side may offset the gain.

This is important enough that we are mentioning twice!

6. Touch your memory, or someone else will.

Memory Allocation: Make it local

Memory Allocation: Make it local

- **Linux has a "first touch policy" for memory allocation**
  - *alloc functions don't actually allocate your memory
  - Memory gets allocated when "touched"
- **Problem: A code can allocate more memory than available**
  - Linux assumes "swap space," we don't have any
  - Applications won't fail from over-allocation until the memory is finally touched
- **Problem: Memory will be put on the core of the "touching" thread**
  - Only a problem if thread 0 allocates all memory for a node
- **Solution: Always initialize your memory immediately after allocating it**
  - If you over-allocate, it will fail immediately, rather than a strange place in your code
  - If every thread touches its own memory, it will be allocated on the proper socket / die.

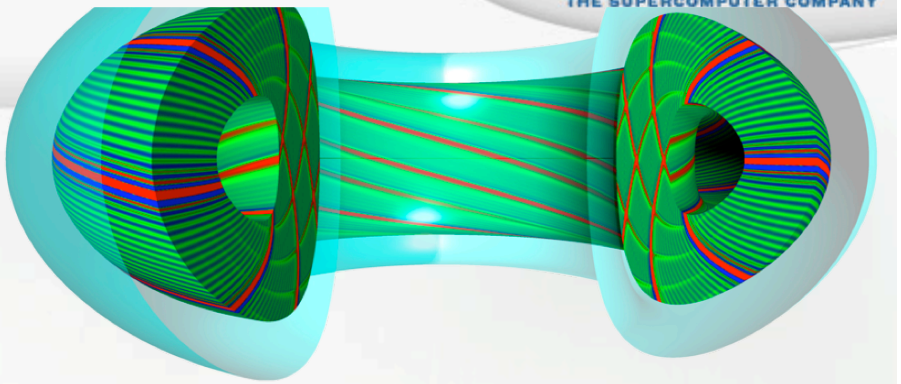
7. Try different MPI Rank Orders

Is your nearest neighbor really your nearest neighbor?
And do you want them to be your nearest neighbor?

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: (DEFAULT) 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
  - Spread out IO across nodes

Reordering example: GYRO



- **GYRO 8.0**
  - B3-GTC problem with 1024 processes
- **Run with alternate MPI orderings**

Reorder method	Comm. time
1 – SMP (Default)	11.26s
0 – round-robin	6.94s
2 – folded-rank	6.68s

Note:

• The rank reordering only works on nodes. If you want to pack within a node in a special way use the aprun –cc 'cpu list' .

• Hence to get a bit more out of the folded-rank option use aprun –cc 0,6,12,18,19,13,7,1,2,8,14,20,21,15,9,3,4,10,16,22,23,17,11,5

This folds across nodes, and folds within dies on a node.

Reordering example: TGYRO

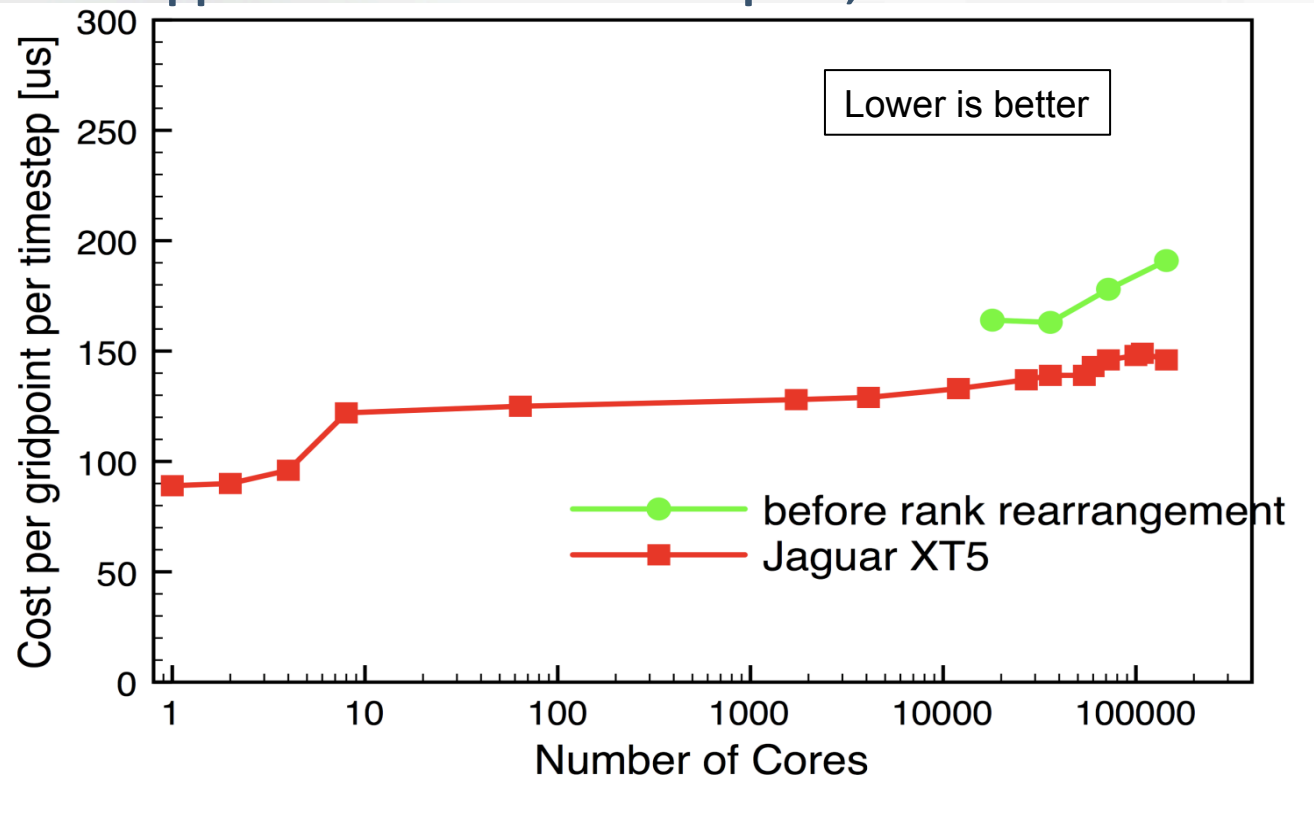
- **TGYRO 1.0**
  - Steady state turbulent transport code using GYRO, NEO, TGLF components
- **ASTRA test case**
  - Tested MPI orderings at large scale
  - Originally testing weak-scaling, but found reordering very useful

Reorder method	TGYRO wall time (min)		
	20480 Cores	40960 Cores	81920 Cores
Default	99m	104m	105m
Round-robin	66m	63m	72m

Huge win!

Rank Reordering Case Study

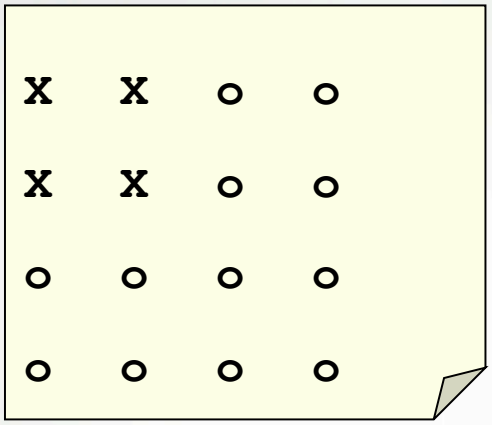
Application data is in a 3D space, X * Y * Z



- Communication is nearest-neighbor. (Halo exchange)
- Default ordering results in 12x1x1 block on each node. (Istanbul example, 12 cores)
- A custom reordering is now generated: 3x2x2 blocks per node, resulting in more on-node communication

Note: Using blocks or slabs within a node may help some communications. For a 6x4 chunk, you could try a 4 6x1, or 4 3x2 chunks, with each die getting one chunk.

Rank order choices: Many options, depends on pattern



- **Nodes marked X heavily use a shared resource**
- **If the shared resource is:**
  - Memory bandwidth: scatter the X's
  - Network bandwidth to others, again scatter
  - Network bandwidth among themselves, concentrate
- **Check out pat_report, grid_order, and mgrid_order for generating custom rank orders based on:**
  - Measured data
  - Communication patterns
  - Data decomposition

8. Try Huge Pages

Gemini loves to use Huge pages ☺

Why use Huge Pages

- The Gemini perform better with HUGE pages than with 4K pages.
- HUGE pages use less GEMINI resources than 4k pages (fewer bytes).
- Your code may run with fewer TLB misses (hence faster).

Huge Pages – How to use

- **Link in the hugetlbfs library into your code '-lhugetlbfs'**
- **Set the HUGETLB_MORECORE env in your run script.**
  - Example : export HUGETLB_MORECORE=yes
- **Use the aprun option –m###h to ask for ### Meg of HUGE pages.**
  - Example : aprun –m500h (Request 500 Megs of HUGE pages as available, use 4K pages thereafter)
  - Example : aprun –m500hs (Request 500 Megs of HUGE pages, if not available terminate launch)
- **Note: If not enough HUGE pages are available, the cost of filling the remaining with 4K pages may degrade performance.**

9. Tune malloc.

But isn't that a system call?

Trick Question: No, but it's expensive and should be done as infrequently as possible!

GNU Malloc

- **GNU malloc library**
  - malloc, calloc, realloc, free calls
    - Fortran dynamic variables
- **Malloc library system calls**
  - Mmap, munmap =>for larger allocations
  - Brk, sbrk => increase/decrease heap
- **Malloc library optimized for low system memory use**
  - Can result in system calls/minor page faults

Improving GNU Malloc

- **Detecting "bad" malloc behavior**
  - Profile data => "excessive system time"
- **Correcting "bad" malloc behavior**
  - Eliminate mmap use by malloc
  - Increase threshold to release heap memory
- **Use environment variables to alter malloc**
  - MALLOC_MMAP_MAX_ = 0
  - MALLOC_TRIM_THRESHOLD_ = 536870912 (or appropriate size)
    (only trims heap when this amount total is freed)
- **Possible downsides**
  - Heap fragmentation
  - User process may call mmap directly
  - User process may launch other processes
- **PGI's –Msmartalloc does something similar for you at compile time**

Running Jobs: Basic aprun options

Option	Description
-D	Debug (shows the layout aprun will use)
-n	Number of MPI tasks Note: If you do not specify the number of tasks to aprun, the system will default to 1.
-N	Number of tasks per Node
-m	Memory required per Task
-d	Number of threads per MPI Task. Note: If you specify OMP_NUM_THREADS but do not give a –d option, aprun will allocate your threads to a single core. You must use OMP_NUM_THREADS to specify the number of threads per MPI task, and you must use –d to tell aprun how to place those threads
-S	Number of Pes to allocate per NUMA Node
-ss	Strict memory containment per NUMA Node

Aprun examples

- **To run using 1376 MPI tasks with 4 threads per MPI task:**

  export OMP_NUM_THREADS=4
  aprun -ss -N 4 -d 4 -n 1376 ./xhpl_mp

- **To run without threading:**

  export OMP_NUM_THREADS=1
  aprun –ss –N 16 -n 5504 ./xhpl_mp

Thanks

- **Thanks to the following people for contributing expertise and slides to this presentation:**
  - Jeff Larkin, Cray ORNL Center of Excellence
  - Cray Performance Team
  - Cray MPT Group
  - Cray Programming Environment Group