

Cray XE6 Architecture and Performance Top 10

Jeff Larkin

<larkin@cray.com>

Cray XE6 Tuning Top 10

- 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 or xtpe-interlagos*
- *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*

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
- 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

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)

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)

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: -vec-report1
 - man ifort; man icc; man iCC

3. Enable Compiler Feedback

- What does your code look like to the compiler?

Compiler Feedback Examples: PGI

```
! Matrix Multiply
do k = 1, N
  do j = 1, N
    do i = 1, N
      c(i,j) = c(i,j) + &
        a(i,k)*b(k,j)
    end do
  end do
end do
```

mm: Click to edit the
outline text format
produces reordered loop
nest: **Second Outline**
level
Generated 3
alternate loops for the
loop - **Third Outline**
Level
Generated vector
sse code for the loop
Generated 2
prefetch instructions for
the loop - **Fifth**
Outline
Level
- **Sixth**
Outline

Compiler Feedback Examples: Cray

```
18.  ib-----<          do k = 1, N
19.  ib ibr4-----<      do j = 1,
20.  ib ibr4 Vbr4--<      do i =
21.  ib ibr4 Vbr4          c(i,j) = c(i,j) + &
22.  ib ibr4 Vbr4          a(i,k) * b(k,j)
23.  ib ibr4 Vbr4-->      end do
24.  ib ibr4----->      end do
25.  ib----->          end do
```

ftn-6007 ftn: SCALAR File = mm.F90, Line = 18

A loop starting at line 18 was interchanged with the loop starting at line 19.

ftn-6254 ftn: VECTOR File = mm.F90, Line = 18

A loop starting at line 18 was not vectorized because a recurrence was found on "C" at line 21.

ftn-6049 ftn: SCALAR File = mm.F90, Line = 18

Compiler Feedback Examples: Pathscale

```
(mm.F90:20) Vectorization is not likely to  
be beneficial (try -LNO:simd=2 to vectorize  
it). Loop was not vectorized.
```

```
(mm.F90:20) Vectorization is not likely to  
be beneficial (try -LNO:simd=2 to vectorize  
it). Loop was not vectorized.
```

```
(mm.F90:20) Vectorization is not likely to  
be beneficial (try -LNO:simd=2 to vectorize  
it). Loop was not vectorized.
```

```
(mm.F90:20) Vectorization is not likely to  
be beneficial (try -LNO:simd=2 to vectorize  
it). Loop was not vectorized.
```

```
(mm.F90:19) Generated 40 prefetch  
instructions for this loop
```

Compiler Feedback Examples: Intel

```
mm.F90(20) : (col. 9) remark: LOOP WAS VECTORIZED.
```

```
mm.F90(20) : (col. 9) remark: LOOP WAS VECTORIZED.
```

```
mm.F90(20) : (col. 9) remark: LOOP WAS VECTORIZED.
```

Compiler Feedback Examples: GNU

```
mm.F90:20: note: LOOP VECTORIZED.
```

```
mm.F90:11: note: vectorized 1 loops in function.
```

4. Library Loading

- Use the `xtpc-<arch>` module and it is all automatic
- *The OpenMP threaded BLAS/LAPACK library is the default if the `xtpc-<arch>` module is loaded. The serial version is used if 'OMP_NUM_THREADS' is* 16

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.

6. Touch your memory, or someone else will.

- Memory Allocation: Make it local

Memory Allocation: Make it local

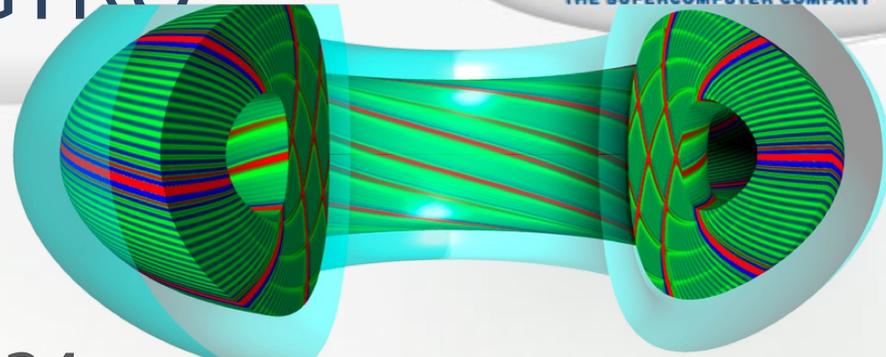
- Linux has a “first touch policy” for memory allocation
 - *alloc functions don’t actually allocate your memory
 - Memory gets allocated when “touched”
- Problem: A code can allocate more memory than available
 - Linux assumes “swap space,” we don’t have any
 - Applications won’t fail from over-allocation 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

7. Try different MPI Rank Orders

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

- 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`

0,6,12,18,19,13,7,1,2,8,14,20,21,15,9,3,4,10,16,22,23,17,1
1 5

Reordering example: TGYRO

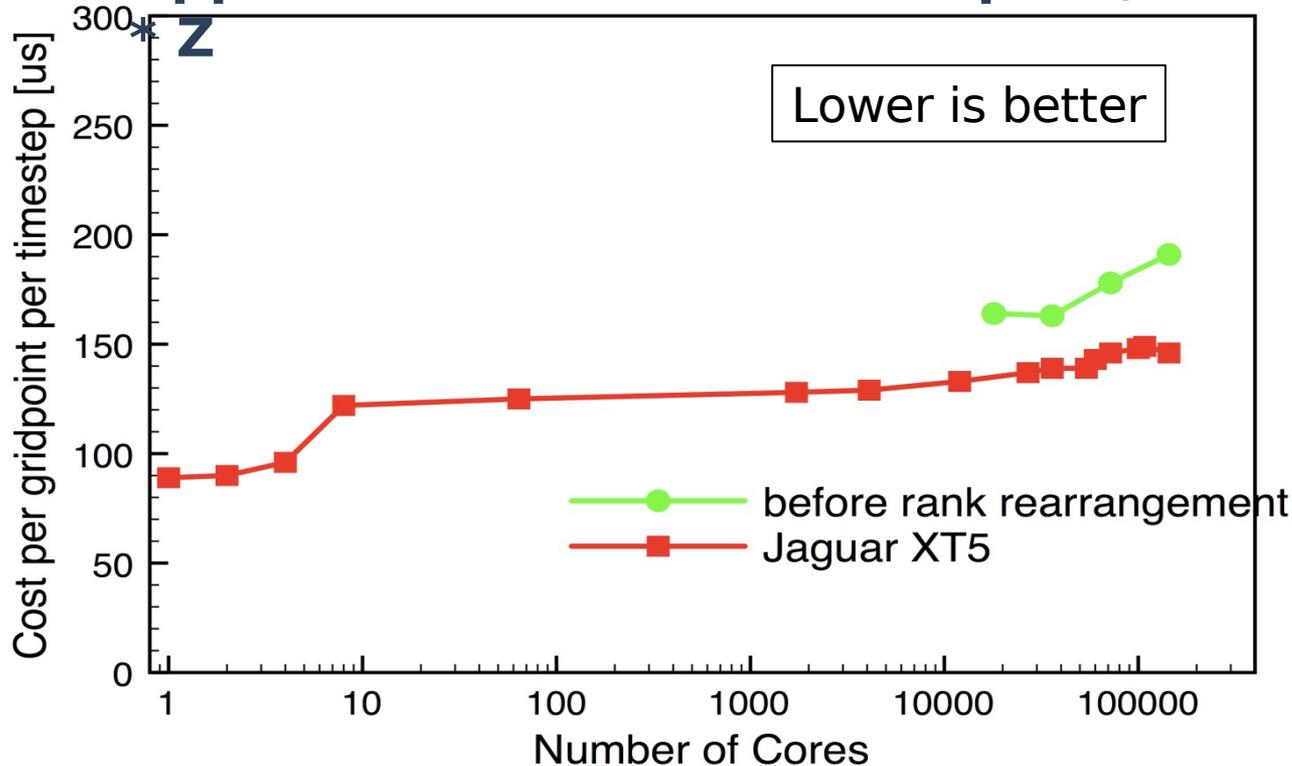
- TGYRO 1.0
 - Steady state turbulent transport code using GYRO, NEO, TGLF components
- ASTRA test case
 - Tested MPI orderings at large scale
 - Originally testing weak-scaling, but found reordering very useful

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

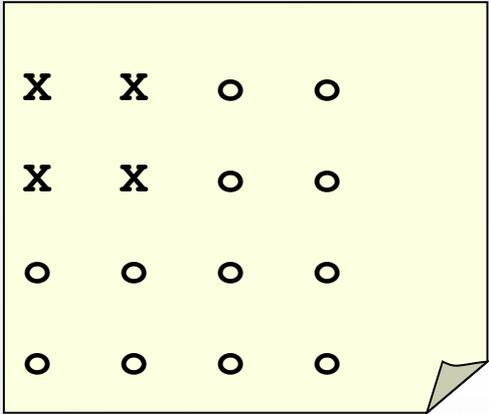
 Huge win!

Rank Reordering Case Study

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



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

- Click to edit the outline network format
- Check out pat_report and grid_order (must have perftools module loaded) for generating custom rank orders based on:
 - Second Outline Level
 - Third Outline Level
 - Fourth Outline Level
 - Fifth Outline Level
 - Sixth Outline Level
- Measure data communication patterns
- Data decomposition

• Cray is also working on an "automatic grid detection"

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).

- Module load `craype-hugepages2M`
 - Must be done both at compile and run time
 - Other sizes are also available to try
- 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.

- 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

- 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

10. Learn the ins and outs of aprun

- Are you launching the job that you think you are?

Running Jobs: Basic aprun options

**Level
Eighth
Outline
Level**

**Ninth Outline Level
Click to edit Master text styles**

● Option ● Description

- | | | |
|--|---|---|
| <ul style="list-style-type: none"> • -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 • -r Reserve some number of cores for handling interrupts (core specialization). This will help a small number of users whose performance is bound by collectives | <ul style="list-style-type: none"> • Click to edit the outline text format • Second Outline Level • Third Outline Level • Fourth Outline Level • Fifth Outline Level | <ul style="list-style-type: none"> • Click to edit the outline text format • Second Outline Level • Third Outline Level • Fourth Outline Level • Fifth Outline Level |
|--|---|---|

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
```

CRAY
THE SUPERCOMPUTER COMPANY