Compiling applications for the Cray XC

With focus on non-accelerator applications

Compiler Driver Wrappers Refresher

- All applications that will run in parallel on the Cray XC should be compiled with the standard language wrappers.

  The compiler drivers for each language are:
  - `cc` – wrapper around the C compiler
  - `CC` – wrapper around the C++ compiler
  - `ftn` – wrapper around the Fortran compiler

- These scripts will choose the required compiler version, target architecture options, scientific libraries and their include files automatically from the current used module environment. Use the `–craype-verbose` flag to see the default options for any compiler.

- Use them exactly like you would the original compiler, e.g. To compile `prog1.f90`:

```
> ftn -c <any_other_flags> prog1.f90
```

Compiler Driver Wrappers (2)

- The scripts choose which compiler to use from the `PrgEnv` module loaded

PrgEnv	Description	Real Compilers
`PrgEnv-cray`	Cray Compilation Environment	`crayftn, craycc, crayCC`
`PrgEnv-intel`	Intel Composer Suite	`ifort, icc, icpc`
`PrgEnv-gnu`	GNU Compiler Collection	`gfortran, gcc, g++`
`PrgEnv-pgi`	Portland Group Compilers	`pgf90, pgcc, pgCC`

- Use module swap to change `PrgEnv`, e.g.

```
> module swap PrgEnv-cray PrgEnv-intel
```

- `PrgEnv-cray` is loaded by default at login
- use `module list` to check what is currently loaded
- Make sure you load the Cray MPI (`cray-mpich`).
  - To support SHMEM load the `cray-shmem` module.

Compiler Versions

- **There are usually multiple versions of each compiler available to users.**
  - The most recent version is usually the default and will be loaded when swapping the `PrgEnv`.
  - To change the version of the compiler in use, swap the Compiler Module, e.g., `module swap cce cce/8.3.10`

PrgEnv	Compiler Module
PrgEnv-cray	cce
PrgEnv-intel	intel
PrgEnv-gnu	gcc
PrgEnv-pgi	pgi

EXCEPTION: Cross Compiling Environment

- **The wrapper scripts, `ftn`, `cc`, and `CC`, will create a highly optimized executable tuned for the Cray XC's compute nodes (cross-compilation).**
- **This executable may not run on the login nodes**
  - Login nodes do not support running distributed memory applications
  - Some Cray architectures may have different processors in the login and compute nodes (such as Haswell compute and Ivybridge login). Typical error is '… illegal Instruction …'
- **If you are compiling for the login nodes**
  - You should use the original direct compiler commands, e.g. `ifort`, `pgcc`, `crayftn`, `gcc`, … PATH will change with modules. All libraries will have to be linked in manually.
  - Conversely, you can use the compiler wrappers {`cc`,`CC`,`ftn`} and use the `-target-cpu=` option among {abudhabi, haswell, interlagos, istanbul, ivybridge, mc12, mc8, sandybridge, shanghai, x86_64}. The x86_64 is the most compatible but also less specific.

About the `-I`, `-L` and `-l` flags

- **For libraries and include files being triggered by module files, you should NOT add anything to your Makefile**
  - No additional MPI flags are needed (included by wrappers)
  - You do not need to add any `-I`, `-l` or `-L` flags for the Cray provided libraries

- **If your Makefile needs an input for `-L` to work correctly, try using '.'**

- **If you really, really need a specific path, try checking '`module show <X>`' to see which environment variables that the module is setting**

- **Order in which you load modules should not matter... But is good practice to load PrgEnv first, then any others**

Dynamic vs Static linking

- **Currently static linking is default for CPU nodes**
  - May change in the future
  - When building for accelerators, modules force dynamic linking

- **To decide how to link,**
  1. you can either set `CRAYPE_LINK_TYPE` to “`static`” or “`dynamic`”
  2. Or pass the ‘`-static`’ or ‘`-dynamic`’ option to the linking wrapper (cc, CC or ftn).

- **Features of dynamic linking :**
  - smaller executable, automatic use of new libs
  - Might need longer startup time to load and find the libs
  - Environment (loaded `modules`) should be the same between your compiler setup and your batch script (e.g., when switching to `PrgEnv-intel`)
- **Features of static linking :**
  - Larger executable (usually not a problem)
  - Faster startup
  - Application will run the same code every time it runs (independent of environment)

- **If you want to hardcode the rpath into the executable use**
  - Set `CRAY_ADD_RPATH=yes` during compilation
  - This will always load the same version of the lib when running, independent of the version loaded by modules

The three styles of dynamic linking

Shared libraries mean applications may use a different versions of a library at runtime than was linked at compile time. On the Cray XC40 there are three ways to control which version is used

1. *Default* – Follow the default Linux policy and at runtime use the system default version of the shared libraries (so may change as and when system is upgraded)
2. *pseudo-static* – Hardcodes the path of each library into the binary at compile time. Runtime will attempt to use this version when the application start (as long as lib is still installed). Set `CRAY_ADD_RPATH=yes` at compile
3. *Dynamic modules* – Allow the currently loaded PE modules to select library version at runtime. App must not be linked with `CRAY_ADD_RPATH=yes` and must add "`export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH`" to run script

OpenMP

- **OpenMP is support by all of the `PrgEnvs`.**
  - CCE (`PrgEnv-cray`) recognizes and interprets OpenMP directives by default. If you have OpenMP directives in your application but do not wish to use them, disable OpenMP recognition with `–hnoomp`.

PrgEnv	Enable OpenMP	Disable OpenMP
`PrgEnv-cray`	`-homp`	`-hnoomp`
`PrgEnv-intel`	`-openmp`	
`PrgEnv-gnu`	`-fopenmp`	
`PrgEnv-pgi`	`-mp`	

- **Intel OpenMP spawns an extra helper thread which may cause oversubscription. Hints on that will follow.**

Compiler man Pages

- For more information on individual compilers

PrgEnv	C	C++	Fortran
PrgEnv-cray	man craycc	man crayCC	man crayftn
PrgEnv-intel	man icc	man icpc	man ifort
PrgEnv-gnu	man gcc	man g++	man gfortran
PrgEnv-pgi	man pgcc	man pgCC	man pgf90
Wrappers	man cc	man CC	man ftn

- Cray docs on Fortran, C, etc., generally only talk about Cray differences from standard
- To verify that you are using the correct version of a compiler, use:
  - -V option on a cc, CC, or ftn command with PGI, Intel and Cray
  - --version option on a cc, CC, or ftn command with GNU

More Hints on Using Compilers

Quick Overview

Using Compiler Feedback

- Compilers can generate annotated listing of your source code indicating important optimizations. Useful for targeted use of compiler flags.
- CCE
  - `ftn -rm`
  - `{cc,CC} -hlist=a`
- Intel
  - `ftn/cc -opt-report3 -vec-report6`
  - If you want this written into a file: add `-opt-report-file=filename`
  - See `ifort --help reports`
- GNU
  - `-ftree-vectorizer-verbose=9`
- PGI
  - `-Minfo=<…>`

Compiler feedback: Loopmark

- For example, compiling with the Cray compiler using "-rm" to generate myfile.lst

```
%%%    L o o p m a r k   L e g e n d    %%%
Primary Loop Type       Modifiers
------- ---- ----       ---------
A   - Pattern matched   a - vector atomic memory operation
                        b – blocked
C   - Collapsed         f – fused
D   - Deleted           i – interchanged
E   - Cloned            m - streamed but not partitioned
I   - Inlined           p - conditional, partial and/or computed
M   - Multithreaded     r – unrolled
P   - Parallel/Tasked   s – shortloop
V   - Vectorized        t - array syntax temp used
                        w - unwound
```

Compiler feedback: Loopmark (cont.)

```
29.  b-------<  do i3=2,n3-1
30.  b b-----<     do i2=2,n2-1
31.  b b Vr--<        do i1=1,n1
32.  b b Vr            u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
33.  b b Vr    *         + u(i1,i2,i3-1) + u(i1,i2,i3+1)
34.  b b Vr            u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
35.  b b Vr    *         + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
36.  b b Vr-->        enddo
37.  b b Vr--<        do i1=2,n1-1
38.  b b Vr            r(i1,i2,i3) = v(i1,i2,i3)
39.  b b Vr    *        - a(0) * u(i1,i2,i3)
40.  b b Vr    *        - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
41.  b b Vr    *        - a(3) * ( u2(i1-1) + u2(i1+1) )
42.  b b Vr-->        enddo
43.  b b----->     enddo
44.  b------->  enddo
```

Compiler Feedback: Loopmark (cont.)

```
 ftn-6289 ftn: VECTOR File = resid.f, Line = 29
  A loop starting at line 29 was not vectorized because a
recurrence was found on "U1" between lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 29
  A loop starting at line 29 was blocked with block size 4.
ftn-6289 ftn: VECTOR File = resid.f, Line = 30
  A loop starting at line 30 was not vectorized because a
recurrence was found on "U1" between lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 30
  A loop starting at line 30 was blocked with block size 4.
ftn-6005 ftn: SCALAR File = resid.f, Line = 31
  A loop starting at line 31 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 31
  A loop starting at line 31 was vectorized.
ftn-6005 ftn: SCALAR File = resid.f, Line = 37
  A loop starting at line 37 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 37
  A loop starting at line 37 was vectorized.
```

What did that loopmark note mean? Use “explain” for more information

```
% explain ftn-6289
```

VECTOR: A loop starting at line %s was not vectorized because a recurrence was found on "var" between lines num and num.

Scalar code was generated for the loop because it contains a linear recurrence. The following loop would cause this message to be issued:

```
DO I = 2,100
   B(I) = A(I-1)
   A(I) = B(I)
ENDDO
```

“explain” utility works for any Cray PE messages, e.g., ftn-*, cc-*, ld-*

Some Cray, Intel, and GNU compiler flags

Feature	Cray	Intel	GNU
Listing	-ra (“report all”) or -rmo (“loop Mark” and “Opts used”)	-list -vec-report3 -opt-report -opt-report-file=name	-fdump-tree-all
Diagnostic	(produced by -ra)	-help diagnostic	-Wall (and other opts)
Free format	-f free	-free	-ffree-form
Preprocessing	-eZ	-P –fpp (Fortran)	-cpp
Suggested Optimization	-O2 (default)	-O3 –xAVX	-O2 -mavx -ftree-vectorize -ffast-math -funroll-loops
Aggressive Optimization	-O3,fp3	-ffast-math -funroll-loops -ftree-vectorize –xAVX	-Ofast -mavx -funroll-loops
Variables size	-s real64 -s integer64	-real-size 64 -integer-size 64	-fdefault-real-8 -fdefault-integer-8
Byte swap	-h byteswapio	-convert big_endian	-fconvert=swap
Enab. OpenMP	(default)	-openmp	-fopenmp

Recommended compiler optimization levels

- **Cray compiler**
  - The default optimization level (i.e., no flags) is equivalent to `-O3` of most other compilers. CCE optimizes rather aggressively by default, but this is also most thoroughly tested configuration
  - Try with `-O3 -hfp3` (also tested this thoroughly)
    - `-hfp3` gives you a lot more floating point optimization, esp. 32-bit
    - In case of precision errors, try a lower `–hfp<number>` (`-hfp1` first; only try `-hfp0` if absolutely necessary)
- **GNU compiler**
  - Almost all HPC applications compile correctly using `-O3`, so try that instead of the cautious default.
  - `-ffast-math` may give some extra performance
  - Add `–march=haswell –mtune=haswell`
  - `-funroll-loops` or `-funroll-all-loops` benefit most apps
- **Intel compiler**
  - The default optimization level (equal to `-O2`) is safe.
  - For haswell: -xCORE-AVX2
  - Try with `-O3.` If that works still (gives right answers), you may try with `–Ofast` or `-fp-model fast=2` for added performance
    - Further relaxed precision with `-fno-prec-div -fno-prec-sqrt`
  - Loop unrolling with `-funroll-loops` or `-funroll-aggressive` may also be beneficial

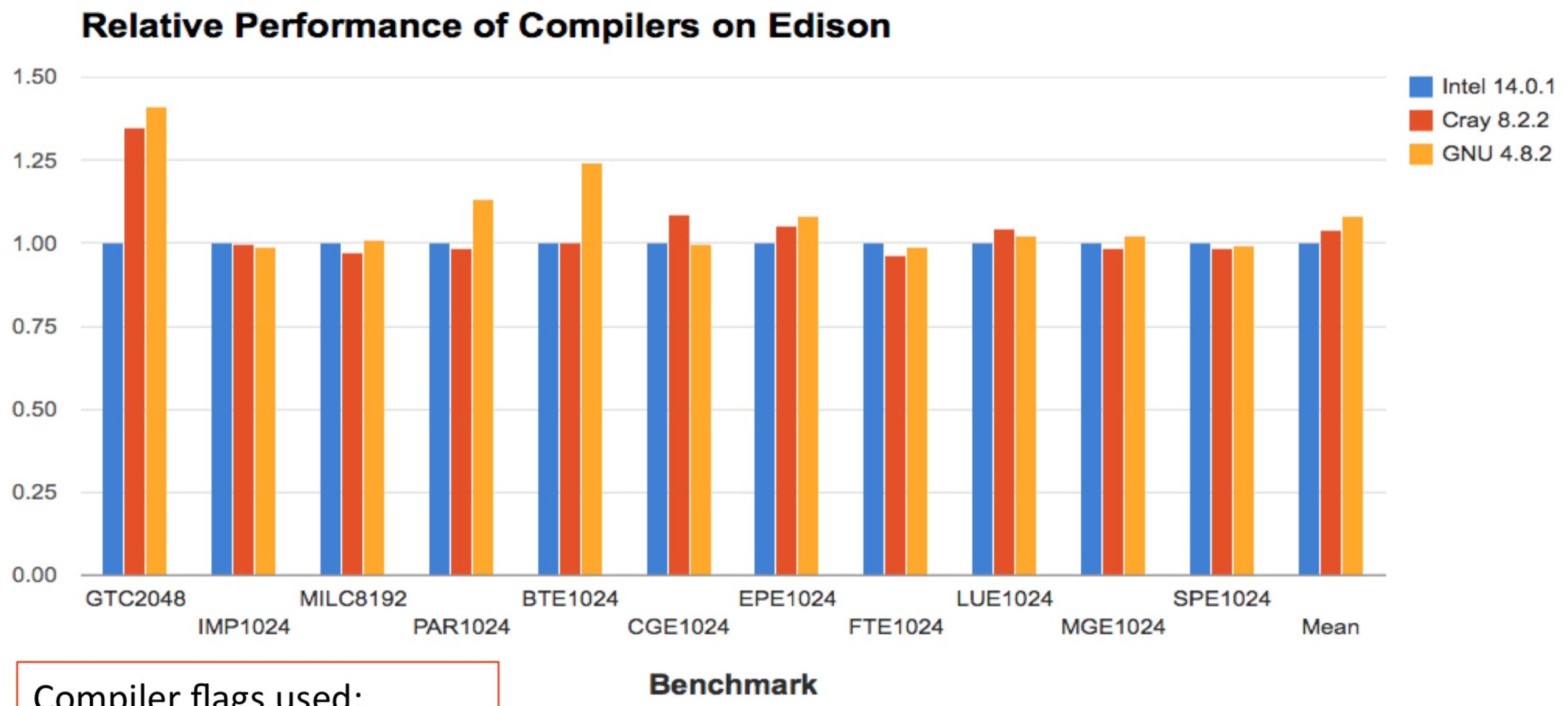
Inlining & interprocedural optimization

- **Cray compiler**
  - Inlining within a file is enabled by default.
  - Command line options `-OipaN` (ftn) and `-hipaN` (cc/CC) where N=0..4, provides a set of choices for inlining behaviour
    - 0 disables inlining, 3 is the default, 4 is even more elaborate
  - The `-Oipafrom=` (ftn) or `-hipafrom=` (cc/CC) option instructs the compiler to look for inlining candidates from other source files, or a directory of source files.
  - The `-hwp` combined with `-h pl=`… enables whole program automatic inlining. *[Be careful – this can slow things down.]*
- **GNU compiler**
  - Quite elaborate inlining enabled by `-O3`
- **Intel compiler**
  - Inlining within a file is enabled by default
  - Multifile inlining enabled by the flag `-ipo`

Loop transformations

- **Cray compiler**
  - Most useful techniques in their aggressive state already by default
  - One may try to improve loop restructuration for better vectorization with `–h vector3`
- **GNU compiler**
  - Loop blocking (aka tiling) with`-floop-block`
  - Loop unrolling `-funroll-loops` or `-funroll-all-loops`
- **Intel compiler**
  - Loop unrolling with `-funroll-loops` or `-unroll-aggressive`

Compilers and NERSC recommended compiler optimization flags



Compiler flags used:

Intel: **-fast –no-ipo**

Cray: **default**

GNU: **-Ofast**

Courtesy of Mike Stewart at NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

COMPUTE | STORE | ANALYZE

Directives for the Cray Compiler

- If you see from the compiler feedback that a loop has not been blocked, unrolled, or vectorized but you are convinced that it should be, you can use compiler directives instead of rising the optimization level –O…
- Cray compiler supports a full and growing set of directives and pragmas, e.g.,
  - !dir$ concurrent
  - !dir$ ivdep
  - !dir$ interchange
  - !dir$ unroll
  - !dir$ loop_info [max_trips] [cache_na]
  - !dir$ blockable
- More information given in
  - man directives
  - man loop_info

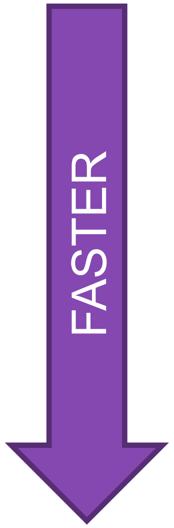
```
!dir$ blockable(j,k)
!dir$ blockingsize(16)
    do k = 6, nz-5
       do j = 6, ny-5
          do i = 6, nx-5
          ! stencil
          end do
       end do
    end do
```

Why are CCE's results sometimes different to other compilers?

- **Cray expect applications to be conformant to language requirements**
  - This include not over-indexing arrays, no overlap between Fortran subroutine arguments, and so on
  - Applications that violate these rules may lead to incorrect results or segmentation faults
  - Note that languages do not require left-to-right evaluation of arithmetic operations, unless fully parenthesized
    - This can often lead to numeric differences between different compilers
    - Some applications expect left-to-right evaluation
    - Use **-hadd_paren** to add automatically parenthesis to select associative operations (+,–,*). Default is **-hnoadd_paren**
- **We are also fairly aggressive at floating point optimizations that violate IEEE requirements**
  - Use **-hfp[0-4]** flag to control that

About reproducibility

- **CCE compilers guarantee that repeated runs with same number of ranks and threads will give identical results. This is not the case for all other compilers. However:**
- **Results can vary with the number of ranks or threads**
  - Use **-hflex_mp=option** to control the aggressiveness of optimizations which may affect floating point and complex repeatability when application requirements require identical results when varying the number of ranks or threads.
  - **option** in order from least aggressive to most is:
    - intolerant: has the highest probability of repeatable results, but also has the highest performance penalty
    - strict: uses some safe optimizations, with high probability of repeatable results.
    - conservative: uses more aggressive optimization and yields higher performance than intolerant, but results may not be sufficiently repeatable for some applications
    - default: uses more aggressive optimization and yields higher performance than conservative, but results may not be sufficiently repeatable for some applications
    - tolerant: uses most aggressive optimization and yields highest performance, but results may not be sufficiently repeatable for some applications



Recommended for bit reproducibility

- **Start from this set**

```
-hflex_mp=conservative –hfp1 –hadd_paren
```

- **Please note:**
  - We only strive to maintain bit reproducibility for applications that are designed correctly to be bit reproducible. The compiler cannot make a non bitrep code reproducible.

WARNING – Intel Helper Threads

- **The Intel OpenMP runtime creates more threads than you might expect. You think you asked for n threads, but....**
  - It creates an extra “helper-thread” (n+1 threads in total)
  - When running with OpenMP, a helper-thread is spawned after the master thread (thread 0). That helper-thread can shift the mapping of threads to cores and so cause over-subscription, because worker-threads of the last MPI-rank per node get placed on the same cores as worker-threads of the first MPI-rank per node;
  - It also has its own method of binding to CPUs (KMP_AFFINITY)

- **Unfortunately both of these options can cause complications with CLE binding of PEs to cores**

- **Cray advice…**
  - Don’t use KMP_AFFINITY to bind threads. “-cc depth” is simple:
    - export KMP_AFFINITY=disabled
    - aprun –cc depth <exe>

Quick aside: “-cc flag” for aprun

- Defines a CPU list for binding, or can also use keywords:
  - *cpu* (default) binds a PE to a single CPU in the NUMA node (socket)
  - *numa_node* binds a PE to the CPUs within a NUMA node (socket)
  - *depth* creates a mask –d PEs wide
  - *none* specifies no binding
- Thus, the default binds each MPI rank and/or OpenMP thread to a particular CPU
- Can also specify binding explicitly:
  - aprun –n 8 –N 4 –S 2 –d 6 –cc 0-5:6-11:12-17:18-23 ./a.out
  - The example above contains four cpu lists. The first (0-5) is applied to the first PE (MPI rank) created and any threads or child processes that result. The second (6-11) is applied to the second PE created and any threads or child processes that result, etc.
  - So the first rank and its 6 children are bound to cores 0 to 5, etc.

About that Intel helper thread

- When running a MPI-OpenMP-hybrid binary built with the Intel compiler suite, one "helper-thread" is spawned per MPI-rank after the master-thread 0 of each MPI-rank, which can shift the binding of threads to cores, causing over-subscription with up to O(50)% performance impact!
- Using default binding (-cc cpu): wrapping around causes double-occupancy, e.g.,
- Helper-thread of each MPI-rank shifts threads 1-5 to the right:

```
cpu-0----1----2----3----4----5----6----7----8----9---10---11
r0t0|r0tH|r0t1|r0t2|r0t3|r0t4|r0t5|r1t0|r1tH|r1t1|r1t2|r1t3|
r3t2|r3t3|r3t4|r3t5|    |    |    |    |    |    |    |    |
cpu12---13---14---14---16---17---18---19---20---21---22---23
r1t4|r1t5|r2t0|r2tH|r2t1|r2t2|r2t3|r2t4|r2t5|r3t0|r3tH|r3t1|
```

- Workaround: **prevent that shift.**
  - export KMP_AFFINITY=disabled
  - export OMP_NUM_THREADS=6
  - aprun –n 8 –N 4 –S 2 –d 6 –cc numa_node a.out # "stay on numa"
  - aprun –n 8 –N 4 –S 2 –d 6 –cc depth a.out # "stay within '-d ...'
  - aprun –n 8 –N 4 –S 2 –d 6 \ # "don't bind helper"
  - –cc 0,x,1-5:6,x,7-11:12,x,13-17:18,x,19-23 a.out

Summary

- **Four compiler environments available on the XC40:**
  - Cray (PrgEnv-cray, default)
  - Intel (PrgEnv-intel)
  - GNU (PrgEnv-gnu)
  - PGI (PrgEnv-pgi)
  - All of them accessed through the wrappers ftn, cc and CC – just do module swap to change a compiler or a version.
- **There is no universally fastest compiler**
  - Performance strongly depends on the application (even input)
  - We try, however, to excel with the Cray Compiler Environment
  - If you see a case where some other compiler yields better performance, let us know! Compiler group often treat that as a bug.
- **Compiler flags do matter**
  - Be ready to spend some effort for finding the best ones for your application.

Running applications on the Cray XC

How applications are run on a Cray XC – recap.

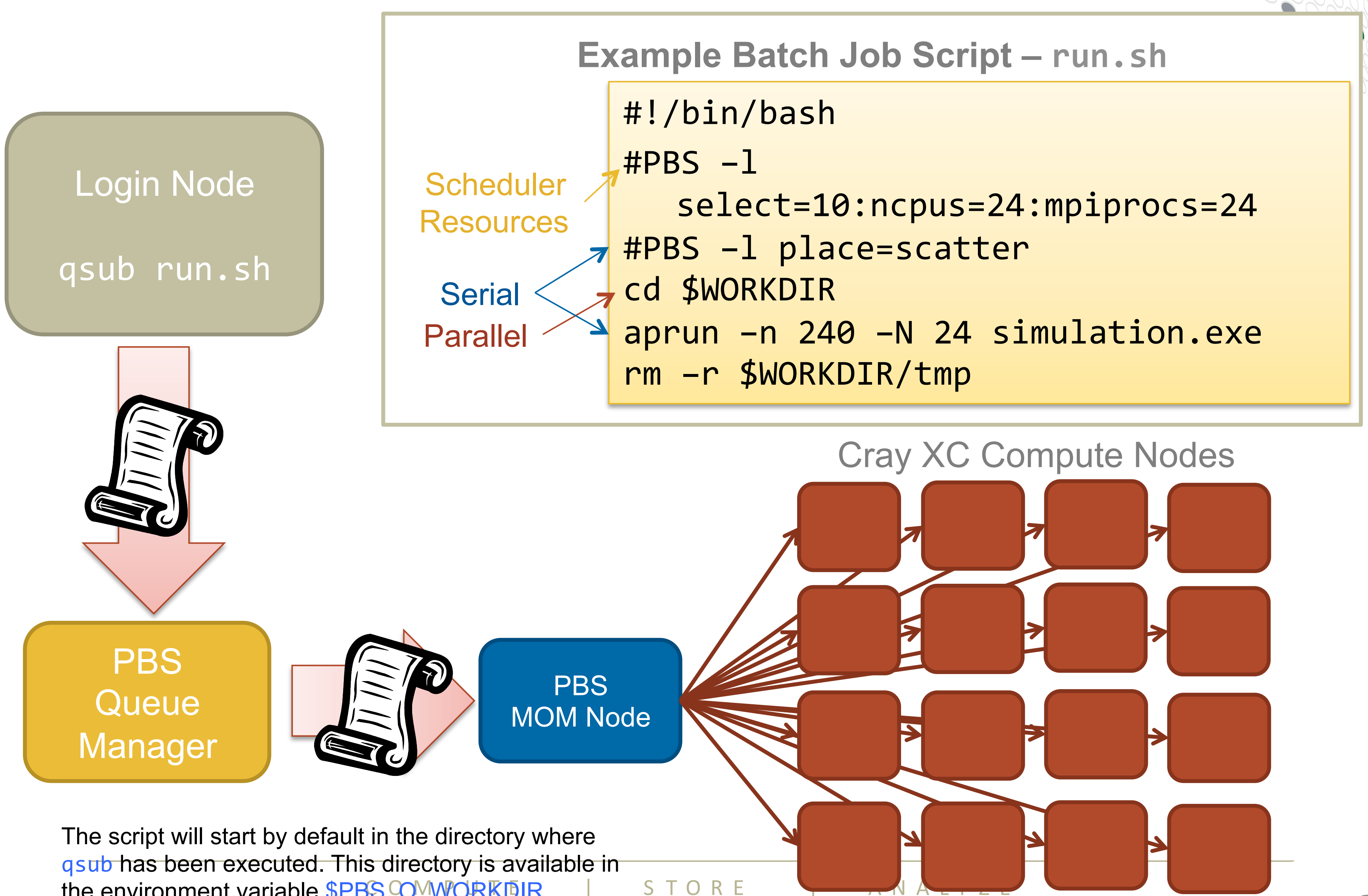
- **The Cray XC is a batch system.**
  - Users submit batch job scripts to the PBS scheduler from a login node for execution at some point in the future. Each job requires resources and a prediction of how long it will run.
  - The scheduler (running on an external server) chooses which jobs to run and allocates appropriate resources
  - The batch system will then execute the user's job script on an a different node than the login node (MOM node).
  - The scheduler monitors the job and kills any that overrun their runtime prediction.
- **User job scripts typically contain two types of statements.**
  1. Serial commands that are executed by the MOM node, e.g.,
     - quick setup and post processing commands, e.g., `rm`, `cd`, `mkdir` ,etc.
  2. Parallel executables that run on compute nodes.
     1. Launched using the `aprun` command.

PBS on the XC40

- **Main PBS commands:**
  - `qsub` – Submit a batch script to SLURM.
  - `aprun` – Run parallel jobs.
  - `qdel`– Signal jobs under the control of SLURM
  - `qstat` – information about running jobs

- **The entire information about your simulation execution is contained in a batch script which is submitted via `qsub`.**
- **The batch script contains one or more parallel job runs executed via `aprun` (job step). Nodes are used exclusively.**
- **The simulations have to be executed on `/mnt/lustre/`…**
- **Useful environment variables:**
  - PBS_NODEFILE: “cat $PBS_NODEFILE | uniq -c | sort” is a file that shows you which nodes you are running on
  - PBS_O_WORKDIR: directory from which qsub was run

Running a batch job



Example Batch Job Script – `run.sh`

```
#!/bin/bash
#PBS -l
   select=10:ncpus=24:mpiprocs=24
#PBS -l place=scatter
cd $WORKDIR
aprun -n 240 -N 24 simulation.exe
rm -r $WORKDIR/tmp
```

The script will start by default in the directory where `qsub` has been executed. This directory is available in the environment variable $PBS_O_WORKDIR

Running an application on the Cray XC ALPS + aprun

- **ALPS : Application Level Placement Scheduler**
- **aprun is the ALPS application launcher**
  - It **must** be used to run application on the XC compute nodes: interactively or in a batch job
  - If `aprun` is not used, the application is launched on the MOM node (and will most likely fail).
  - `aprun` launches groups of Processing Elements (PEs) on the compute nodes
    (PE == (MPI RANK || Coarray Image || UPC Thread || ..))
  - `aprun` man page contains several useful examples
  - The 3 most important parameters to set are:

Description	Option
Total Number of PEs used by the application	-n
Number of PEs per compute node	-N
Number of threads per PE (More precise, the "stride" between 2 PEs on a node)	-d

Script example (large queue on SERC system)

```
#!/bin/sh
#PBS -N jobname
#PBS -l select=343:ncpus=24
#PBS -l walltime=24:00:00
#PBS -l place=scatter
#PBS -l accelerator_type="None"
#PBS -j oe

! Date stamps at top and bottom of script for reference
date

! Useful to take note of where job is launched
cd $PBS_O_WORKDIR; pwd

! Don't necessarily need to load modules at runtime, but
! In case you do (e.g., for dynamic linking):
. /opt/modules/default/init/sh
! (or "source /opt/modules/default/init/csh" for csh)
! Then can do "module load X", "module list" etc.

! Set up to run in the lustre directory /mnt/lustre for any
! parallel application (use diff directory for each run here)
RUNDIR=/mnt/lustre/USERNAME/myapp/run.$$
mkdir –p $RUNDIR
cd $RUNDIR

! Useful info when wondering later where run output might
! have gone
pwd

! Executable can be in lustre or in home directory
! Here, let's copy it to our run directory
EXECDIR=/ufs/home/USERNAME/mybuilddir
cp $EXECDIR/a.out $RUNDIR

! Copy any input data you need or symlink it.  Large input
! (and output) data files should be on lustre
cp /mnt/lustre/USERNAME/INPUTDATA/input_file .

! Useful info when looking back at run output
export MPICH_ENV_DISPLAY=1
export MPICH_VERSION_DISPLAY=1

! Run the executable.  Use timers around aprun as a habit.
! This example uses linux "time" and also calculates walltime in
! a different way.
export  beg_secs=`date +"%s"`

time aprun -j 1 -n 8209 -N 24 ./a.out < input_file > output_file

export  end_secs=`date +"%s"`
let wallsecs=$end_secs-$beg_secs
echo "Time taken in seconds is " $wallsecs

date
! Date stamps at top and bottom of script for reference

! Maybe write output_file to stdout if useful and not too huge
cat output_file
```

How long did my job take?

- **Use internal timers written by application itself**
  - Can be useful to record time between mpi_init and mpi_finalize (print out from rank 0 only!)
- **Date stamp or "time" (or /usr/bin/time) command**
  - This times the aprun command itself
  - Generally aprun takes a few seconds to shutdown at end of run
- **After run similar to that on previous slide (1152 cores):**

```
Application 9545044 resources: utime ~1479438s, stime ~970s, Rss ~81740,
inblocks ~19277790, outblocks ~453745893
```

Produced after aprun finishes

Might be useful: Inblocks/outblocks in units of 512-byte blocks. Rest not too useful.

```
5.932u 0.540s 21:50.60 0.4%     0+0k 0+0io 0pf+0w
```

Output of "time" command

Time taken in seconds is 1311

Calculated from date stamps

COMPUTE | STORE | ANALYZE

What resources did it use?

- **Can be good to record contents of $PBS_NODEFILE during batch session to note what nodes were used (though list will be long if use lots of nodes!)**
  - cat $PBS_NODEFILE | sort | uniq -c
  - Or look at “apstat –avv apid” when job is running to see placement

- **See upcoming information on Cray Performance Tools**
  - perftools-lite is good place to start

- **For accelerators, environment variables are available to produce job statistics**

More info about my running job....

```
>apstat -avv  614502

Total (filtered) placed applications: 1
  Apid  ResId   User  PEs Nodes   Age State  Command
614502 355534 casnan 2400   100 8h36m   run cesm.exe

Application detail
Ap[101]: apid 614502, pagg 0xc6000012c5, resId 355534, user casnan,
       gid 208, account 0, time 0, normal
  Batch System ID = 70460.sdb
  Reservation flags = 0x100000
  Application flags = 0x142001
  Created at Mon Oct  5 00:47:19 2015
  Originator: aprun on NID 198, pid 5168
  Number of commands 1, control network fanout 32
  Network: cookies 0x2a80000/0x2a90000, NTTgran/entries 0/0
  Cmd[0]: cesm.exe -n 2400 -N 24 -j 1, 2730MB, XT, nodes 100, exclusive
  Placement list entries: 2400
  Placement list: 168-172,187-189,212-219,221-227,249,292-319,324-353,374-382,414-422
```

Runtime Variability: What can I do, when should I worry?

- **In a heavily-used batch environment such as at SERC, a certain degree of runtime variability must be expected**
- **The level of variability can depend on many things**
  - The communication patterns in my code
  - The I/O patterns in my code
  - The I/O and communication patterns in the jobs of others on the system at the same time
- **Placement of a particular job on particular nodes on the Cray XC system is generally not the solution – see next slides**
- **Hugepages might help, particularly if code does a lot of alltoall communications, but worth trying in all cases**
- **Use knowledge of code or CrayPAT profiling to consider:**
  - Maximising on-node communications (rank reordering) – see later
  - Eliminating bad I/O patterns (e.g., all ranks reading one file), maybe moving to parallel I/O – again, see later
- **If variation seems particularly bad, consider recording nodes used during each run to try to detect a possible bad node (?)**

xtnodestat

```
crayadm@clogin72:~> xtnodestat
Current Allocation Status at Tue Oct 20 06:53:26 2015

    C0-0                                  C1-0                        C4-0                                C5-0
  n3   acacaFaGaGaHaIaIaHaHaHaJaKaKaJ   aKaKaHaHaMaMaMaMaMaKaUaMaaaMaM      n3 babaa8a4aMaMagbdajaDaDbeajaDaDbd  aEaEaEaEaEaEaEaEaEaNaNaOaOaOagbf
  n2 SSaEacaFaFaGaEaIaIaHaJaHaJaKaKaa SSaKaKaKaHaMaMaMaMaMaMaUaVaaaMaM  n2 babaaMaMaMaMagagaDaDajbeajajaDbd  aEaEaEaEaEaEaEaEaEaNaNaNaOaOagbf
  n1 SSaEacaFaFaGaEaIaIaIaJaHaKaKaJaK  SSaKaKaKaHaTaMaMaMaMaMaUaVaMaMaM  n1 babaaMa8aMaMbdagaDajajbebeajaDbe   aEaEaEaEaEaEaEaEaEaNaNaOaOagbf
c2n0   aDacaFaFaGaGaIaIaIaGaHaKaKaJaK   aKaKaKaHaTaMaMaMaMaMaHaUaMaaaM    c2n0 babaaMa8a4aMbdagagajaDbebeajaDbe aEaEaEaEaEaEaEaEaEaNaNaOaOagbf
  n3   aoapXXananajajat----aC----ajah             aKaQaTaaaHaHaHaHaHaHaKaHaKaKaK  n3 aMa7a8aMaMaMaMbcaQaQ--a9a9a9aLa9 a7bfababaDababababababaDbfaDbfaD--
  n2 SSanananananajajasav--aB----ajaj            SSaQaQaTaaaHaHaHaHaHaHaHaHaKaK  n2 aMa7a8aMaMaMaMbcaQaQaQa9a9a9a9a9 a7a7ababa7ababZZababaDabaDbfaD--
  n1 SSanananananajajarauaxazan--ajaj           SSaQaKaTaTaHaHaHaHaKaHaHaKaHaHaK  n1 aMa7a7a8aMaMaMbcaQaQaQa9a9a9ada9  a7a7aeaba7aDababaDabaDabaDbfbfaD
c1n0   amanananananajajaq--awayan--ajaj           aKaKaQaTaaaHaHaHaKaKaHaKaHaHaK   c1n0 aMaMa7a8aMaMaMbcbcaQaQ--a9a9ada9  a7a7bfababaDababaDababababbfaDbfaD
  n3     aaacadaeaeadagaiaiahakakaaaj          aLaMaaaaaaaaaaaQaPaaaKaaaKaK      n3 agaaaaaaaaaaaaaaaaaP--aaaaaLaaaM        bdaHaHajaHaHaHaHajaHaHaabea7a7a7
  n2 SSSSaaacacaeaeadafaiaiaiakakakaj         SSSSaJaLaaaaaOaaaaaPaQaaaKaFaaaK   n2 aaaaaaaaaaaaaaaaaaaP--aaaaaFa7aM        bdaHaHajajaHaHaHajaDaDaabebfa7a7
  n1 SSSS—aaacaeaeaeadahaiaiaiakakaj         SSSSaJaLaaaaaNaaaaaPaQaPaKaKaaaK   n1 aa--aaaaaaaaaaaaaaaa--aaaaaFa7aM         bdaHaHajajaHaHaHaHaDaDaaaaaDa7a7
c0n0     --abacaeaeSSadahaiaiajakakaj          aJaLaaaaaaSSaaaPaQaPaKaKaaaK     c0n0 aaagaaaaaaaaaaaSSaaaa--aaaa--aLaa        bdbdaHajajaHaHSSaHajaDaHaaaDa7a7
   s00112233445566778899aabbccddeeff 00112233445566778899aabbccddeeff          s00112233445566778899aabbccddeeff 00112233445566778899aabbccddeeff

    C2-0                                  C3-0
  n3   a5a4a4aTaMaUaFa6a6aUa6a6a6a6a7     a6a6a6a8a9a6aGaaaaaaa6a6aMa7aM        C6-0                                 C7-0
  n2 SSa5a4a4a4aMaUaFa2a6aUaUa6a6a6a2 SSa6a6a6aga9a6aGbaaaaabba6aMaPaM      n3 bgbga3a3a3aHbfbfbfaabhaaaaaaaaa7    aaaaaaaMaMa5a5a5a5aaaaa5a5a5a5--
  n1 SSa5a4a4a4aTaUaFaFa6a6aUa6a6a6a5  SSa6a6a6aEa9a6a8baaaaaa6a6a6aPaa      n2 bgbga3a3a3aHbfbfbfbfaabhaaaaaaa7    aaaaa7aMaMbha5a5aaaaaaa5a5a5a5bk
c2n0   a5a5a4a4aTaUaUaFa6a6aUa6a6a6a6   a6a6a6a6a9a8aQaGaaaaaaa6a6aga7      n1 bgbgbga3a3aHbfbfbfbfaabhaaaaaaa7     aaaaaabhaMaMa5a5aaaaa5a5aMa5a5aH
  n3   a3aUa4a4a4a4a4a4a4aMaMa5a5a5a5    aUa6a6a6a6a6a6a6a6a6a6a6a6a6a6   c2n0 bgbgbga3a3a3aHbfbfbfaabhaaaaaaaaa  aaaaaabhaMaMa5aaa5aaaaaaaaa5a5aa
  n2 SSa3aUa4a4a4a4a4a4a4aMaMaMa5a5a5 SSaUa6a6a6a6a6a6a6a6a6a6a6a6a6a6      n3 ----------bgbgbfbgbgbgbgbgbgbgbg XX     aabibibibibibjbjaaaabjbjbj------
  n1 SSa3aUa4a4a4a4a4a4a4aMaMaMa5a5a5 SSaUa6a6a6a6a6a6a6a6a6a6a6a6a6a6      n2 ------------bgbfa7bgbgbgbgbgbgbg XX     aabibibibibibjbjaaaabjbjbj------
c1n0   a3aUaUa4a4a4a4a4a4a4aMaMa5a5a5   aUAAa6a6a6a6a6a6a6a6a6a6a6a6a6      n1 ------------bga7bfbgbgbgbgbgbgbg          aaaabibibibibibibja7aabjbjbj------
  n3     aDa2a3a3a3a3a3a3a3a3a3a3a3a3        a6a7abaUaaaaaFa6a6aUaUaUaUa6aU  c1n0 ------------bgbgbfbgbgbgbgbgbgbg          aaaabibibibibibibjaaaabjbjbj------
  n2 SSSSaDaDa3a3a3a3a3a3a3a3a3a3a3a3  SSa6aUabaUaaaaaFaFa6a6aUaUaUa6aU      n3 bfbfagagaHbcbcbcbcaa-----------          aaaaaaa7a7aHaHaaaabhaaaaaaaaaHaa
  n1 SSSSaDaDa2a3a3a3a3a3a3a3a3a3a3a3  SSa6a6abaUa6aaaaaFa6a6aUaUaUa6aU      n2 bfbfbfagaHaHbcbcbcbcaa----------         aaaaaaaaa7aHaHaaaabhbhaaaaaaaHaa
c0n0     aDaDa2a3a3SSa3a3a3a3a3a3a3a3     a7a6a7aba6aaSSaFa6a6aUaUaUa6a6      n1 bfbfbfagaHaHbcbcbcbcaa----------         aaaaaaaaa7aaaHaaaaaabhaaaaaHaaaa
   s00112233445566778899aabbccddeeff 00112233445566778899aabbccddeeff     c0n0 bfbfbfagaHaHbcSSbcbcaa----------        a7aaaaaaa7a7aHSSaaaabhaaa7aHaaaa
                                                                               s00112233445566778899aabbccddeeff 00112233445566778899aabbccddeeff

                                                                          Legend:
                                                                             nonexistent node                      S  service node
                                                                          ;  free interactive compute node          -  free batch compute node
                                                                          A  allocated (idle) compute or ccm node  ?  suspect compute node
                                                                          W  waiting or non-running job             X  down compute node
                                                                          Y  down or admindown service node         Z  admindown compute node

                                                                          Available compute nodes:         0 interactive,        79 batch
```

You have a VERY busy machine...!

^^^^^^^^^^^^^

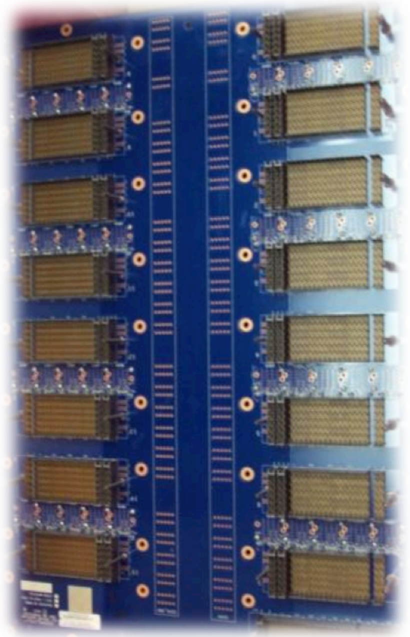
Hardly any free nodes!

C O M P U T E | S T O R E | A N A L Y Z E

Aries and the Dragonfly topology

Cray XC Network

- The Cray XC system is built around the idea of optimizing interconnect bandwidth and associated cost at every level



Rank-1
PC Board: ¢¢¢

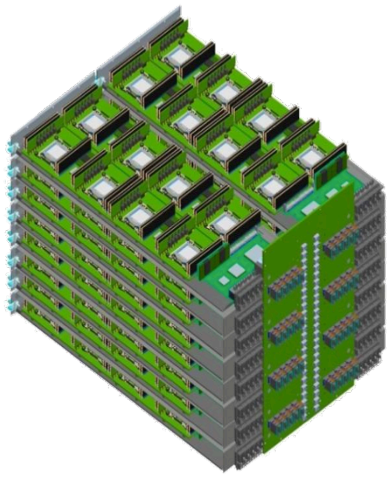
Rank-2
Passive CU: $

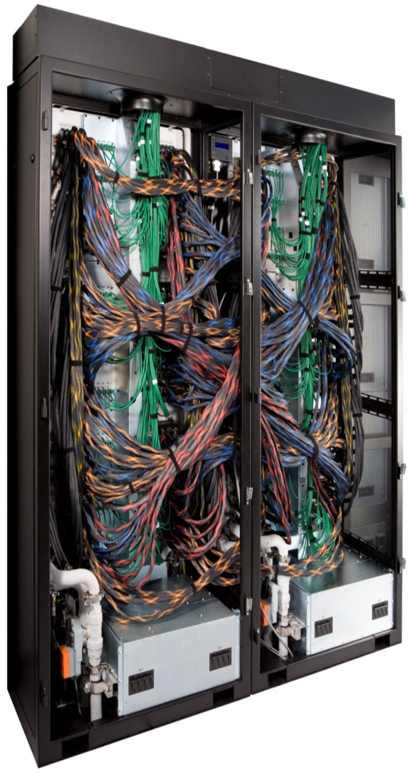
Rank-3
Active Optics: $$$$

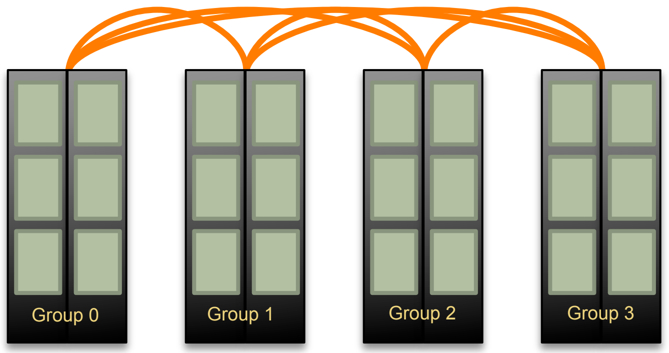
14 GBps

12.5 GBps

Cray XC Packaging Review







Rank-1
Chassis

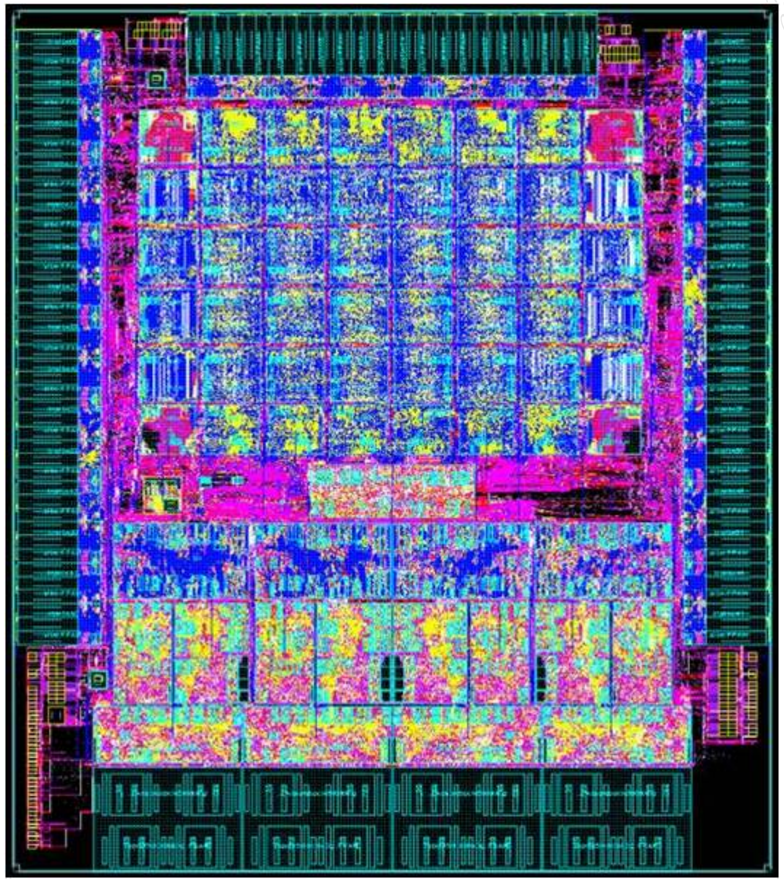
Rank-2
2 Cabinet Group

Rank-3
Between Groups

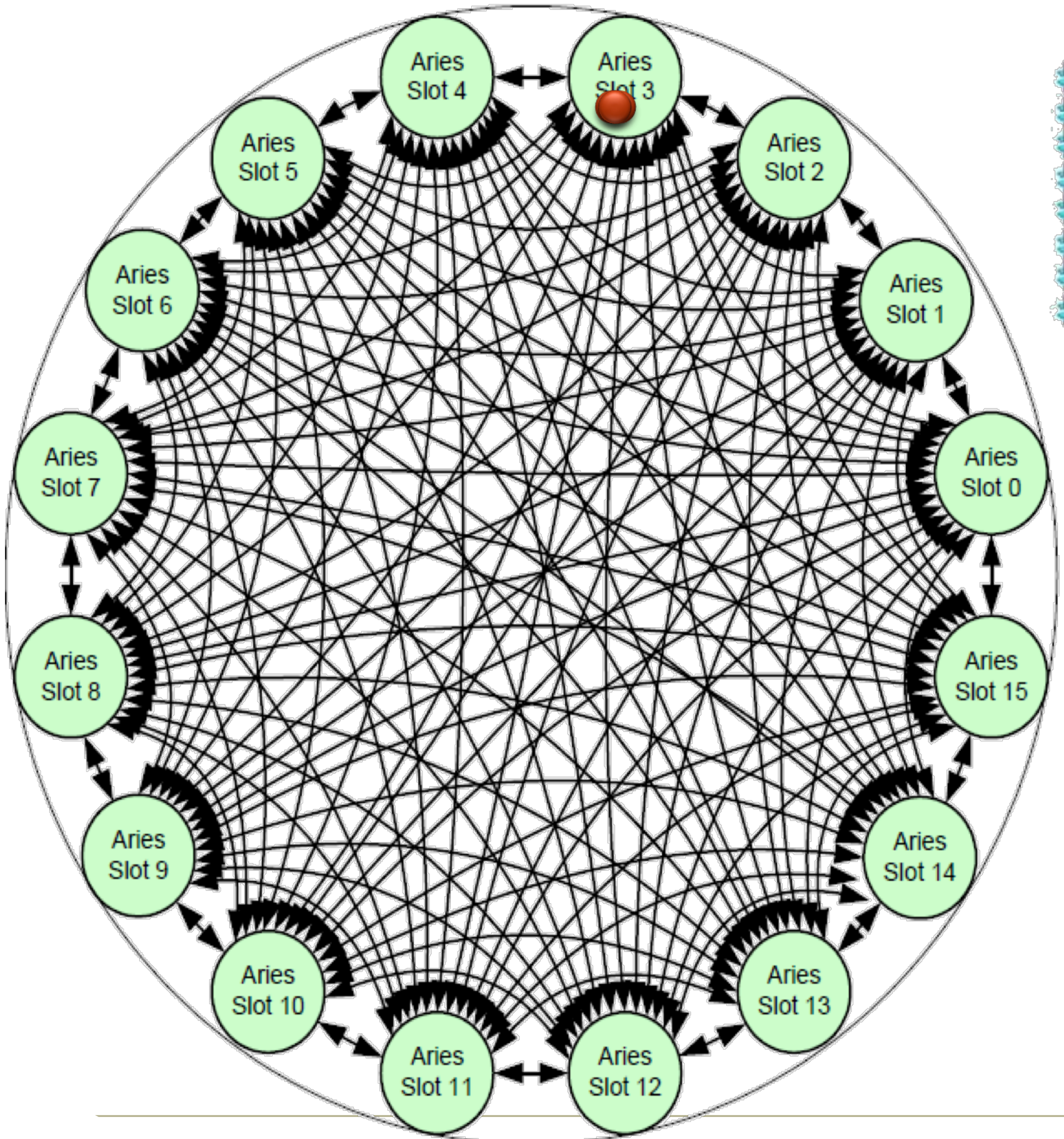
4 nodes = 1 blade; 16 blades = 1 chassis; 3 chassis = 1 cabinet; 2 cabinets = 1 group

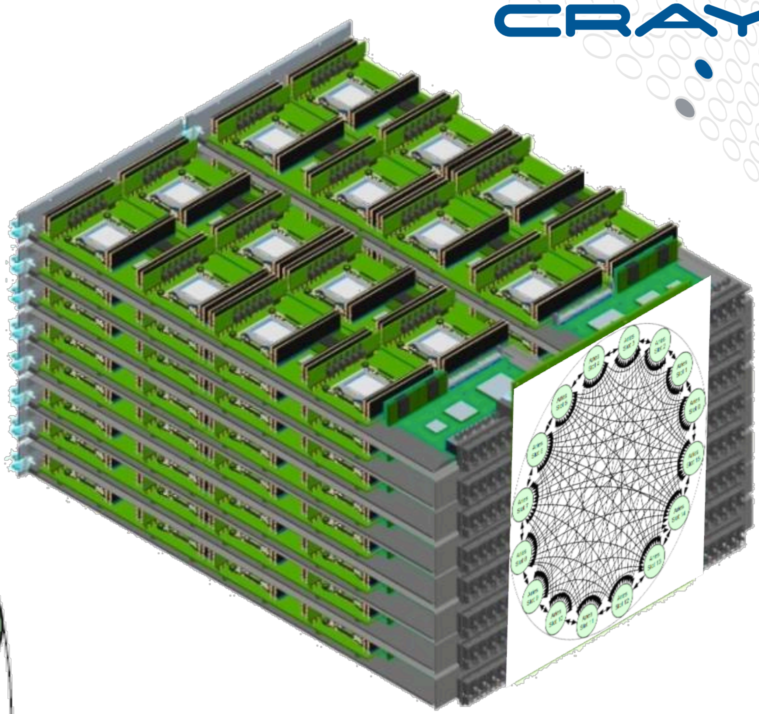
Aries

- Aries is the Cray custom interconnect ASIC used in the Cray XC product family
  - 40nm process
  - Die size: 16.6 x 18.9mm
  - Gate count: 217M
  - 184 lanes of high speed SerDes
    - SerDes=Serializer/Deserializer (SerDes pronounced sir-deez)
  - 30 optical network lanes
  - 90 electrical network lanes
  - 64 PCI Express lanes
- **4 NICs**
  - Each Aries connects 4 nodes (1 blade) to the interconnect



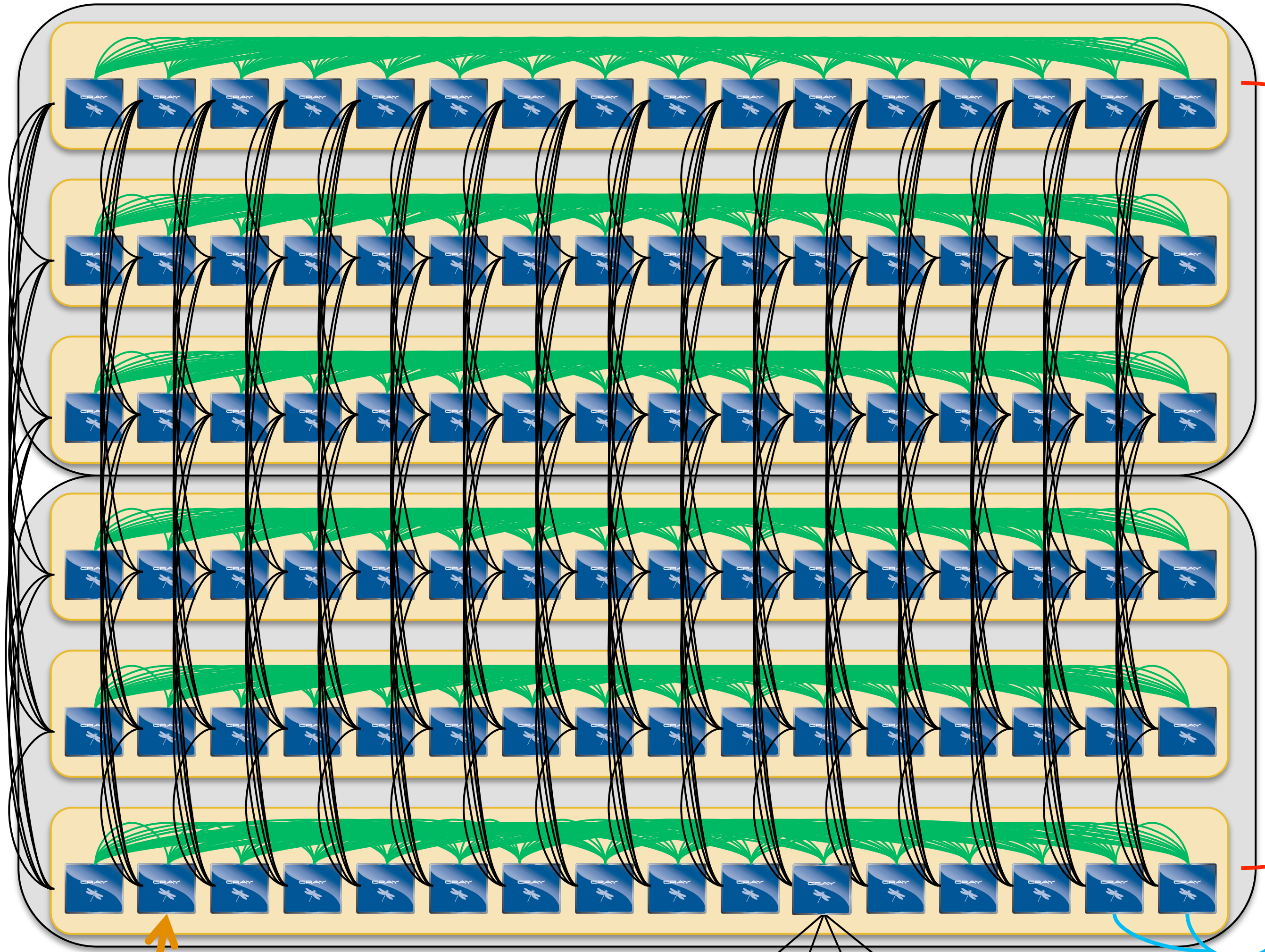
Cray XC Rank1 Network



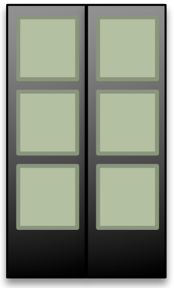


- Chassis with 16 compute blades
- 128 Sockets
- Inter-Aries communication over backplane
- Per-Packet adaptive Routing

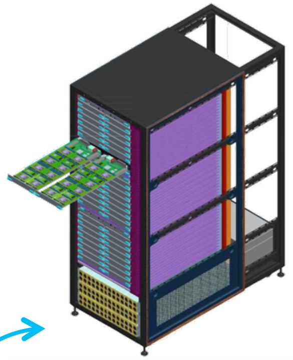
Cray XC Rank-2 Copper Network



2 Cabinet Group 768 Sockets

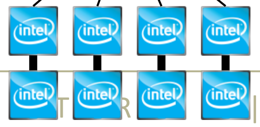


6 backplanes connected with copper cables in a 2-cabinet group: "Black Network"

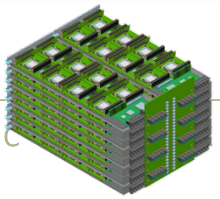


Active optical cables interconnect groups "Blue Network"

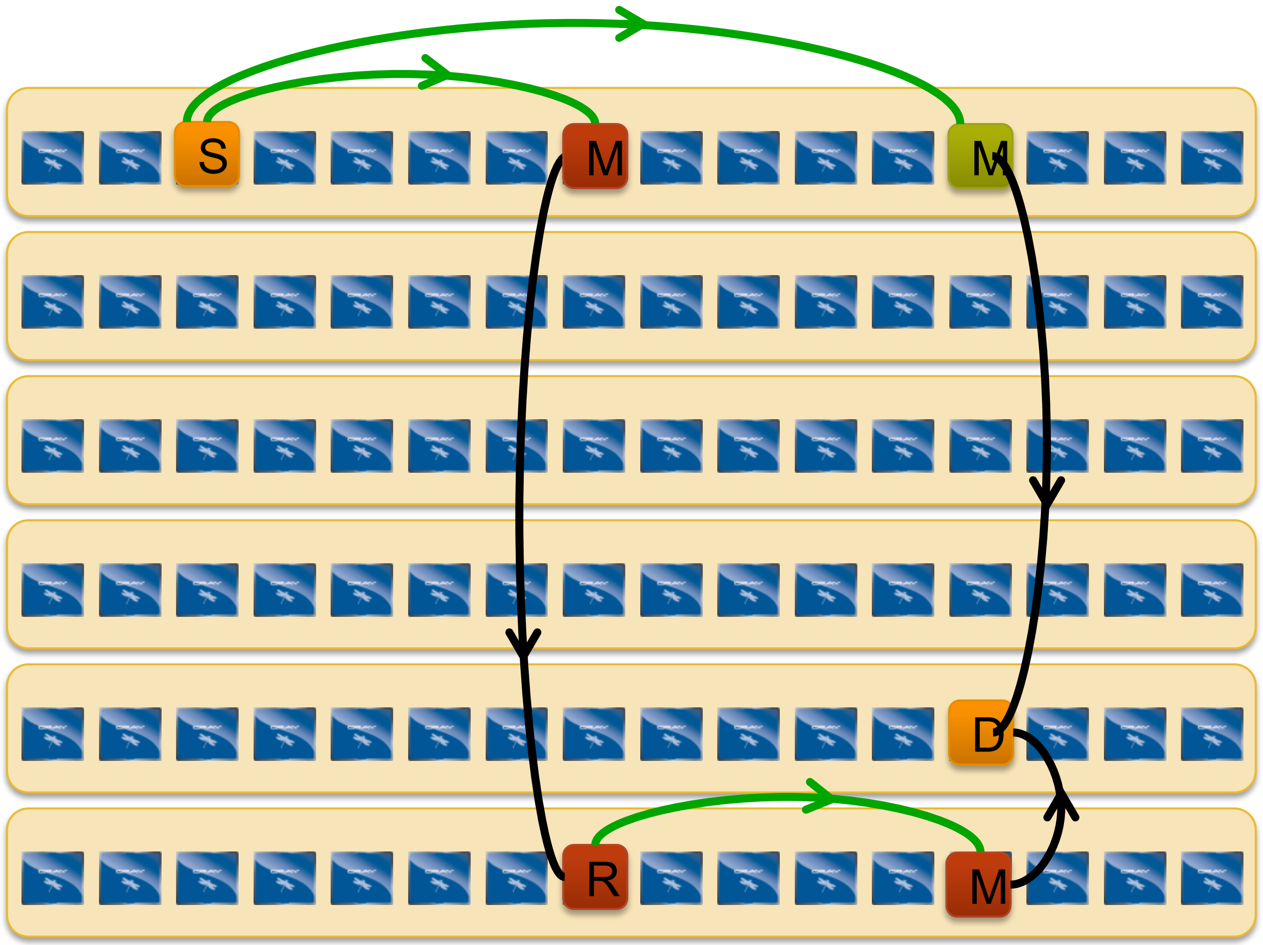
4 nodes connect to a single Aries



16 Aries connected by backplane "Green Network"



Cray XC Routing – Rank 2 Network



Minimal routes between any two nodes in a group are just two hops

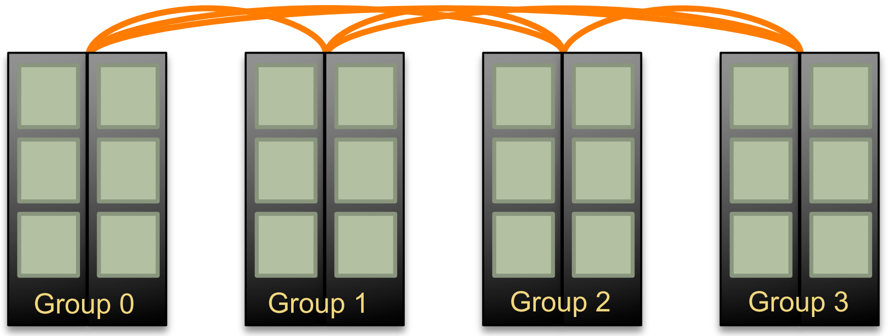
Non-minimal route requires four hops.

With adaptive routing we select between minimal and non-minimal paths based on load

The Cray XC Class-2 Group has sufficient bandwidth to support full injection rate for all 384 nodes with non-minimal routing

Cray XC Network Overview – Rank 3 Network

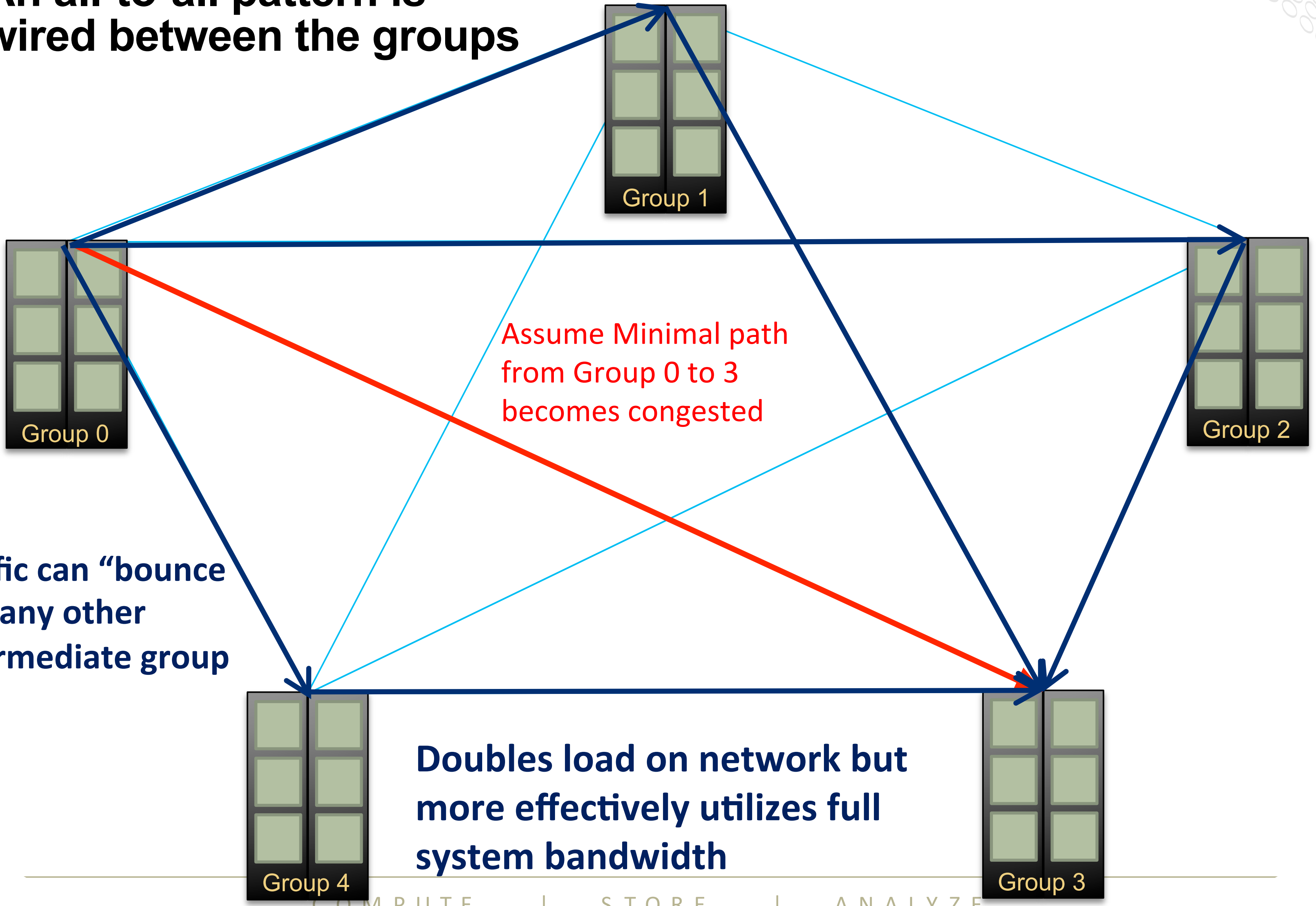
- An all-to-all pattern is wired between the groups using optical cables (blue network)
- Up to 240 ports are available per 2-cabinet group
- The global bandwidth can be tuned by varying the number of optical cables in the group-to-group connections



Example: An 4-group system is interconnected with 6 optical "bundles". The "bundles" can be configured between 20 and 80 cables wide

Adaptive Routing over the Blue Network

- **An all-to-all pattern is wired between the groups**



Assume Minimal path from Group 0 to 3 becomes congested

Traffic can "bounce off" any other intermediate group

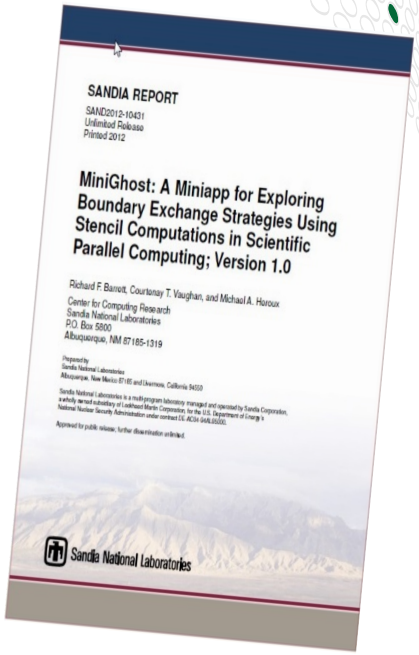
Doubles load on network but more effectively utilizes full system bandwidth

Why is the Dragonfly topology a good idea?

- **Scalability**
  - Topology scales to very large systems, providing scalable global bandwidth
- **Performance**
  - More than just a case of clever wiring, this topology leverages state-of-the-art adaptive routing that Cray developed with Stanford University
  - Smoothly mixes small and large messages
  - *Cray invested in bringing it to market – IBM and Mellanox have not*
- **Simplicity**
  - Implemented *without expensive* external switches.
  - Configurable
- **Cost**
  - Dragonfly maximizes the use of backplanes and passive copper components
  - Dragonfly minimizes the use of active optical components

Dragonfly is placement insensitive

- **Example: Sandia miniGhost**
  - Finite difference mini-app, difference stencil across 3D domain (includes stencil computation, halo exchange, global summation of grid values)
- **Running on 2256 node (12 Cabinets) CSCS system (¼ global bandwidth)**
  - Runtime in seconds for 100 cycles



SANDIA REPORT
SAND2012-10431
Unlimited Release
Printed 2012

MiniGhost: A Miniapp for Exploring Boundary Exchange Strategies Using Stencil Computations in Scientific Parallel Computing; Version 1.0

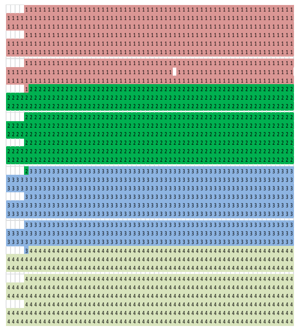
Richard F. Barrett, Courtenay T. Vaughan, and Michael A. Heroux
Center for Computing Research
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1319

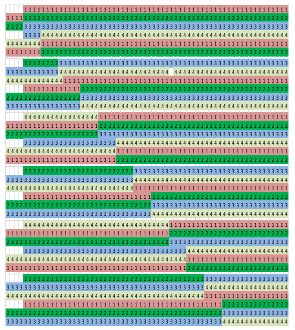
Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

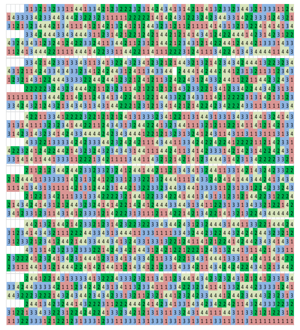
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories







Contiguous Blocks of 512 nodes			
69.0	68.8	68.9	68.9

Random blocks of 64 nodes			
69.4	69.4	69.4	69.5

Random layout of nodes			
70.9	71.0	70.6	70.5

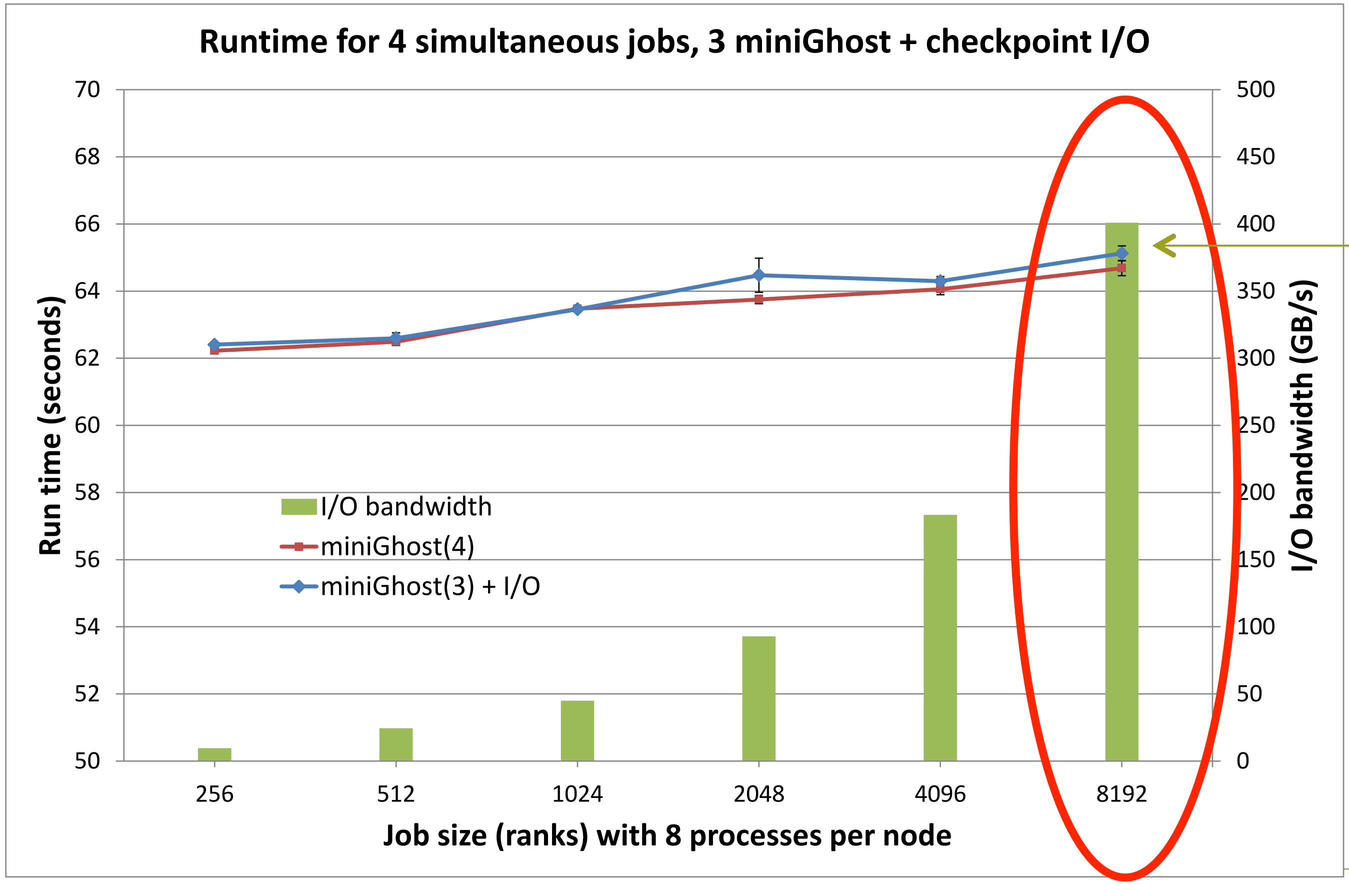
Perfect Placement

Worst-Case Placement

< 3% variance from best-case to worst-case placement

Mix of application and streaming I/O traffic

- Analysis of the impact of big I/O traffic on performance of other codes
- Compared two runs
  1. Four miniGhost jobs spread out across the whole machine vs.
  2. Three miniGhost plus one performing big many-to-few I/O



I/O Job sustaining 400GB/sec (95% clients to 5% servers)

Impact to compute jobs is tiny (64.5 sec to 65 sec)

More Placement Data from NERSC system

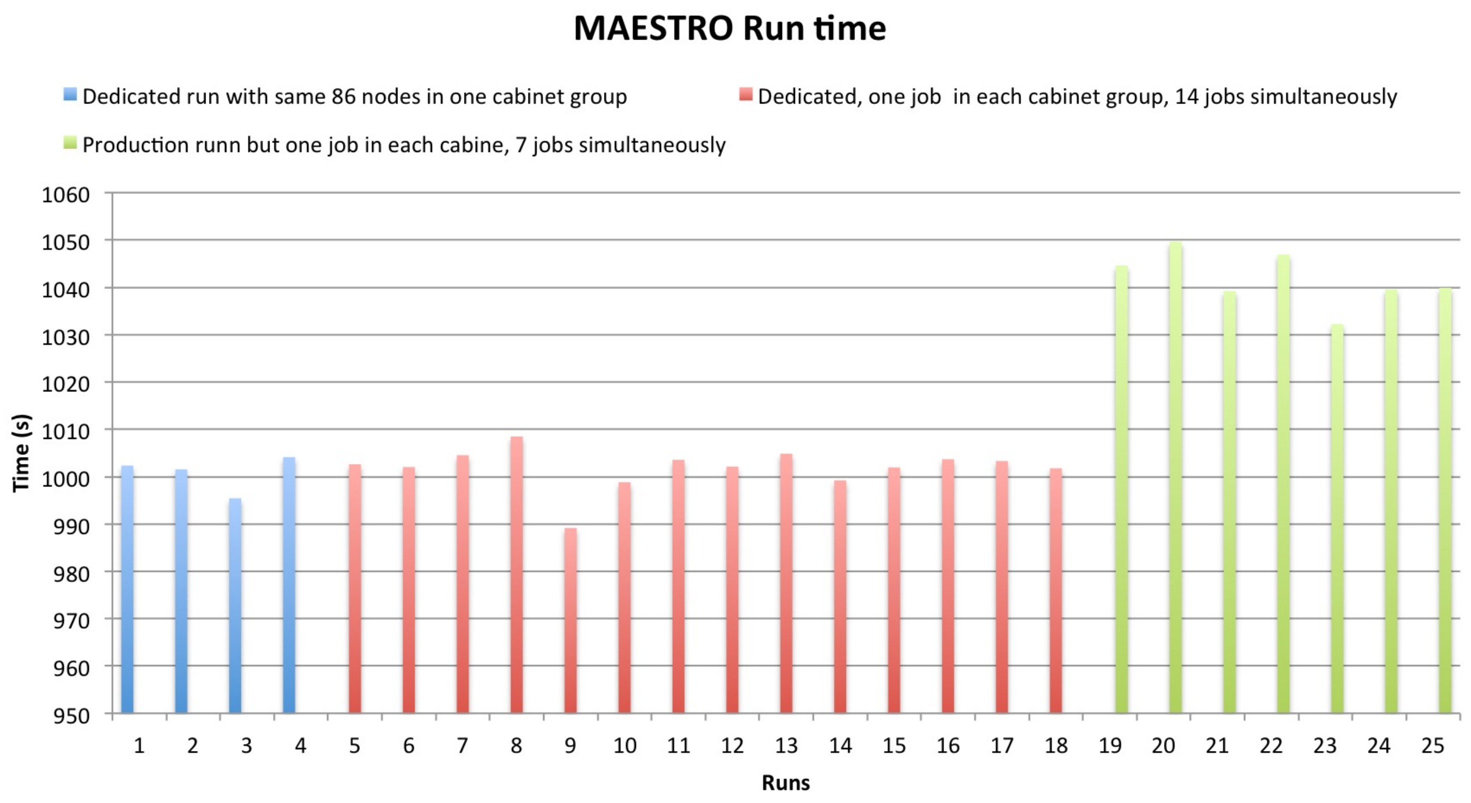
NERSC-6 application benchmark production and dedicated time comparison

Application	CAM	GAMESS	GTC	IMPACT-T	MAESTRO	MILC	PARATEC
Concurrency	240	1024	2048	1024	2048	8192	1024
Streams/Core	2	2	2	2	1	1	1
Dedicated Time (s)	273.08	1,125.80	863.88	579.78	935.45	446.36	173.51
Production Time(s)	277.07	1,218.17	871.06	597.25	996.70	482.87	198.45
Slowdown[1)]	1.5%	8.2%	0.8%	3.0%	6.5%	8.2%	14.4%

1) Slowdown=Time(Production)/Time(Dedicated)

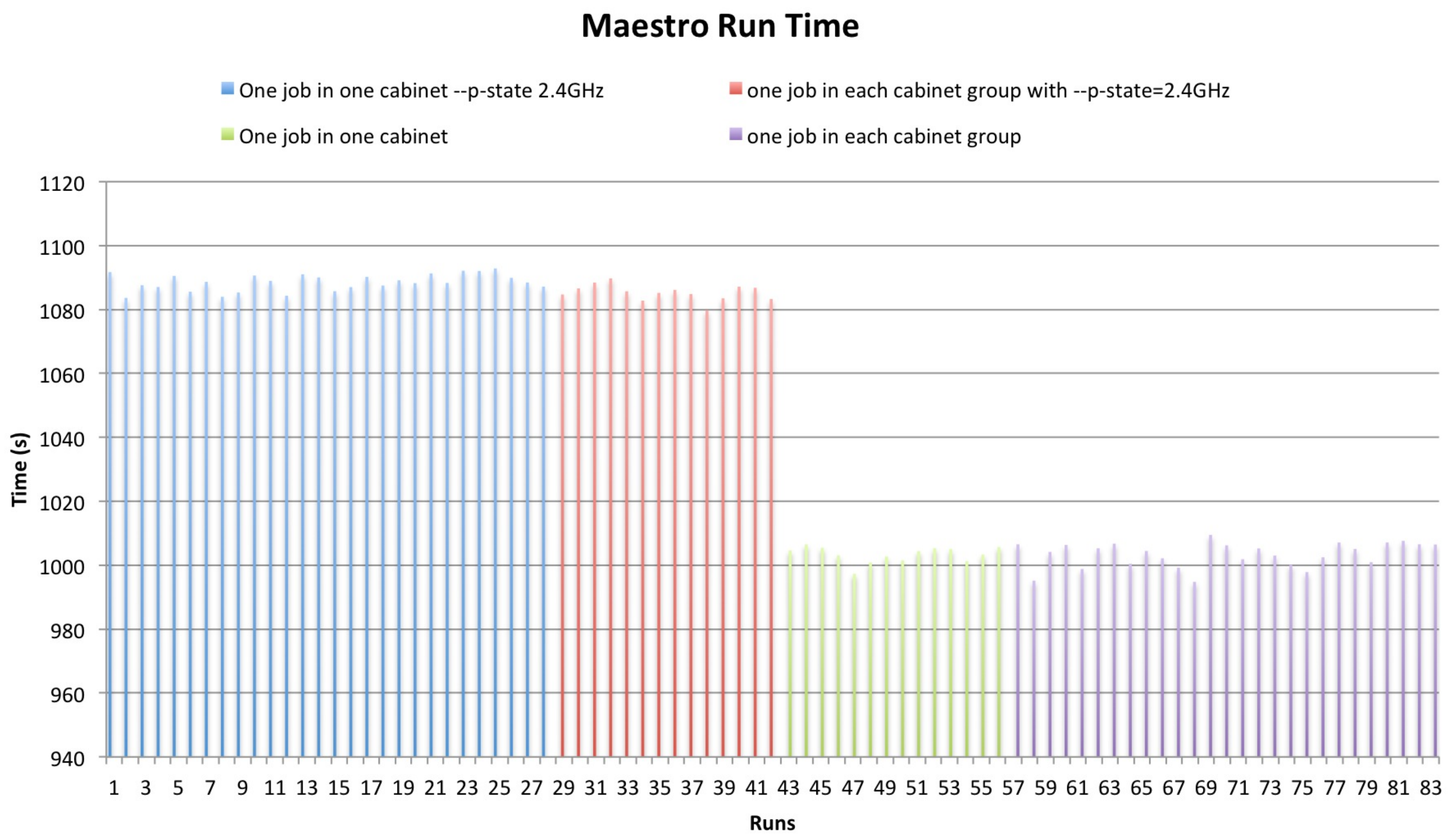
COMPUTE | STORE | ANALYZE

Node placements and run time



C O M P U T E | S T O R E | A N A L Y Z E

Node placements and run time



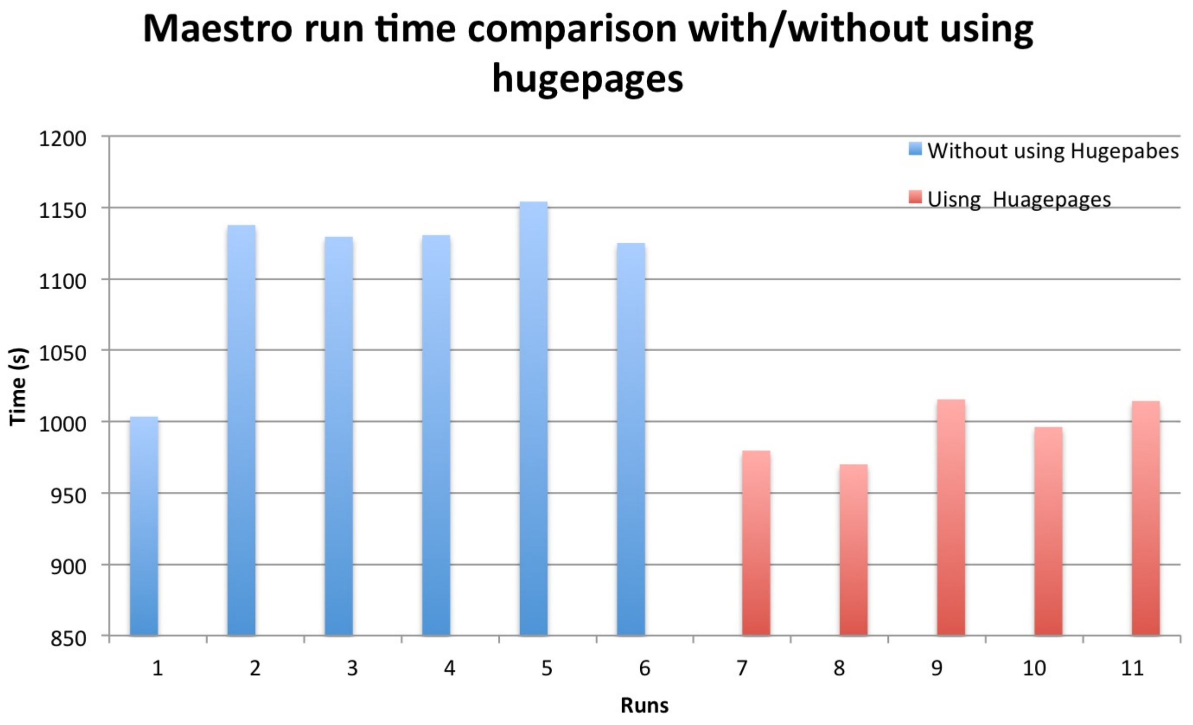
C O M P U T E | S T O R E | A N A L Y Z E

Hugepages may improve code performance

- **Hugepages may improve memory performance for common access patterns on large data sets and for apps doing heavy I/O**
- **The Aries may perform better with HUGE pages than with the default 4K pages.**
  - HUGE pages use less Aries resources than 4k pages
  - More important when remotely access large percentage of nodes memory in an irregular manner
- **May get "cannot run errors" if there are not enough Hugepages memory available (memory page fragmentation)**
- **HUGE="more than 4K". Use modules to change default page sizes (man intro_hugepages)**
  - craype-hugepages2M, craype-hugepages4M, craype-hugepages8M, craype- hugepages16M, craype-hugepages32M, craype-hugepages64M, craype-hugepages128M,craype-hugepages256M,craype-hugepages512M
- **Users are recommended to experiment with hugepages**
- **This feature is implemented at link and run time. To use:**

```
module load craype-hugepages2M
cc –o my_app my_app.c
Then run with the same huge pages module loaded
```

Hugepages may improve your code performance



Mastro run time improves by 11% by average when using hugepage memory compared to not using the hugepages.

Finally...I/O Changes can help Performance

- **Users are encouraged to experiment with Lustre stripe count, size to obtain a good I/ O performance for their workloads, with a general guidance that a larger stripe count may increase bandwidth but subject more contention, and vice versa.**
- **Many factors may affect the I/O performance of your jobs**
  - Contentions for the resources with other users
  - Hardware failure or downgraded performance
  - File system fragmentation
  - Bad user practices
- **For example:**
  - A user used fixed offset and stripecount 1 and filled up one of the OSTs a couple of times.
  - Using too large stripe counts for small file I/O inviting contention with other users unnecessarily and get widely varying I/O time
  - Writing tens of thousands of small files from all ranks simultaneously, placing pressure on the file system for all other users
- **More on I/O later!**

Basics about communication

Costs of communication

- All parallel applications communicate data between individual processes
  - unless they're embarrassingly parallel

- The cost of any communication is usually defined by two properties of the underlying network (or memory system)

1. Latency
2. Bandwidth

Costs of communication

1. **Latency**
   - The time from a message being sent to it reaching its destination
   - Dominates the performance of small messages
   - Combination of factors from:
     - constant software and/or hardware overheads.
     - the physical and topological distance between the nodes (hops)

2. **Bandwidth**
   - The maximum rate at which data can flow over the network.
   - Dominates the performance of larger messages.
   - Bandwidth between nodes generally depends upon the number of possible paths between nodes on the network (topology)

How message size affects communication performance

- (As with all things) the decisions made by application developer can affect the overall performance of the application.
- The size of messages sent between processes affects how important latency and bandwidth costs become.
- When a message is small the network latency is dominant.
- Therefore it is advisable to try and bundle multiple small messages into fewer larger message to reduce the number of latency penalties.
- Try to post `MPI_IRecv` call before the `MPI_ISend` call
  - can avoid unnecessary buffer copies

Understanding Inter- and Intra- node performance

- **The rise of multi-core has led to fat nodes being common**
  - Five years ago there may have one or two CPUs per node…
  - Now we routinely see 16-32 CPUs per node.
  - This will only increase in the future (e.g., Intel Phi)
- **Codes usually have multiple MPI ranks per node**
  - Many (even most) codes are flat MPI
    - rather than hybrid with, for instance, OpenMP threads
  - Even hybrid codes usually have more than one rank per node
    - as threading does not usually scale well across NUMA regions (sockets)
- **Latency, bandwidth is different for on- and off-node messages**
  - messages between PEs on the same node (intra-node) will be faster
  - messages between PEs on different nodes (inter-node) will be slower
- **We can optimise application performance by maximising communication between processes on the same node – more later on**

Miscellaneously Useful Flags for MPI

- **Performance enhancements**
  - `export MPICH_COLL_SYNC=1`
    - Adds a barrier before collectives,
    - use this if CrayPAT makes your code run faster.
- **Reporting**
  - `export MPICH_CPUMASK_DISPLAY=1`
    - Shows the binding of each MPI rank by core and hostname
  - `export MPICH_ENV_DISPLAY=1`
    - Print the value of all MPI environment variables at runtime (STDERR)
  - `export MPICH_MPIIO_STATS=1`
    - Prints some MPI-IO stats useful for optimisation (STDERR)
  - `export MPICH_RANK_REORDER_DISPLAY=1`
    - Prints the node that each rank is residing on, useful for checking `MPICH_RANK_REORDER_METHOD` results.
  - `export MPICH_VERSION_DISPLAY=1`
    - Display library version and build information.
- **For more information: `man intro_mpi`**

How can I make my MPI faster? Some hints

- **Runtime options**
  - Try to maximise on-node transfers (rank reordering – see later)
  - Try using optimised collectives; or DMAPP collectives (relink needed)
    - See intro_mpi man page for more on collectives
- **Help the MPI library get better overlap**
  - use non-blocking MPI calls
    - MPI_Isend, MPI_Irecv, MPI_Iallgather...
- **Try to reorder code to give more potential for overlap**
  - local computation (or I/O) that can be done while messages transfer