Migrating to CTS-1 and TOSS 3's Tri-Lab Common Environment (TCE)

LC Users Meeting

John Gyllenhaal
Livermore Computing

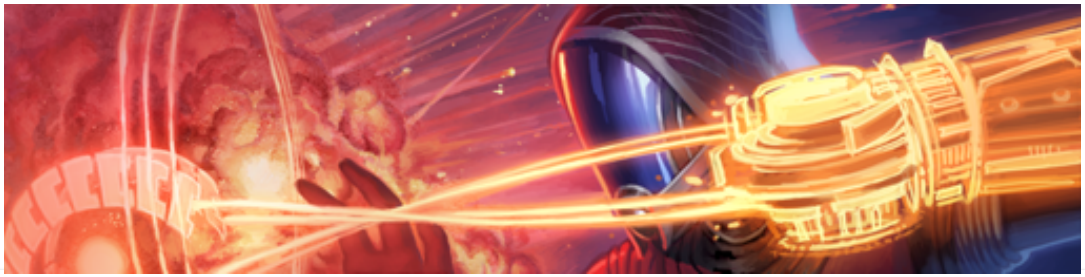
September 8, 2016

LLNL-PRES-702557
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Just some of the incredimazing topics covered by this talk, clickbait style!

- CTS-1 and the surprising myths about x86 core speed!
- Taming hyper-threading, the untold story!
- RIP LC's /usr/local: You won't believe what replaced it!
- Shocking changes to the Intel compiler's C++11 support!
- Modules? Really? What were they thinking?
- JUST SAY NO to using these modules with complex builds!
- This one weird trick will have you never typing module again!



CTS-1: Wait! I thought x86 cores get faster with each new generation!

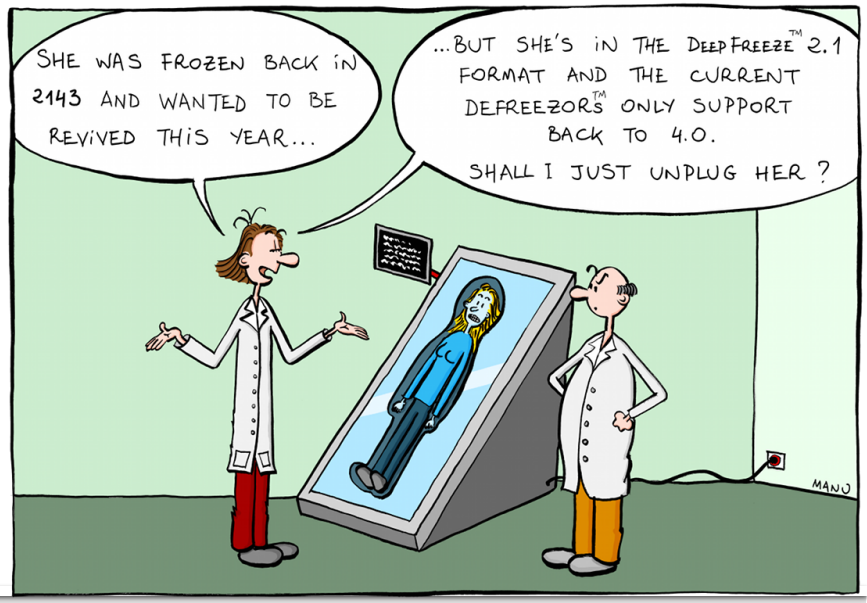
- CTS-1 design chosen that maximized node throughput instead of core speed
  - 36 cores/node (max throughput) versus 16 cores/node (slightly faster cores than TLCC2)
  - 256 core job uses 1/2 the nodes but may run 20% slower (or just as fast, it is app dependent)
- Power and temperature limited with significant processor fabrication variations
  - Turbo mode dynamically adjusts clock rate to keep within bounds (typically power limited)
  - Clocks up to 50% faster with 1 core per socket versus all cores per socket used
  - Culled slowest nodes out of Jade and Quartz and moved to LC's test system Opal
- TLCC2: Zin, Cab, RZAlastor, Etc.
  - 115W per 8 core socket (Sandy Bridge)
  - 2.6GHz base frequency
  - 2.6-3.3GHz Turbo mode (Linpack -> 2.6GHz)
- CTS-1: Jade, Quartz, RZGenie, etc.
  - 120W per 18 core socket (Broadwell)
  - 2.1GHz base frequency
  - 2.1-3.3GHz Turbo mode (Linpack -> 1.9GHz)

Making hyper-threading work for you, not against you

- Hyper-threading always on in TOSS3, even on TLCC2 machines
  - sysconf(_SC_NPROCESSORS_ONLN) returns 2 X number of cores
  - TOSS2's dynamic hyper-threading hack caused problems and broke GPU support
  - Unfortunately Linux can place processes on threads/cores non-optimally
  - Binding is key to getting good and predictable performance
  - Shown to reduce noise impact on performance at scale (with proper binding)
- SLURM binds by default except on machines like RZGENIE
  - Should get good performance/binding on node-scheduled machines
  - Need –exclusive on RZGENIE/RZALASTOR/INCA/etc to see binding effects
  - Similar to putting 'mpibind' on srun line (srun –n 16 mpibind a.out)

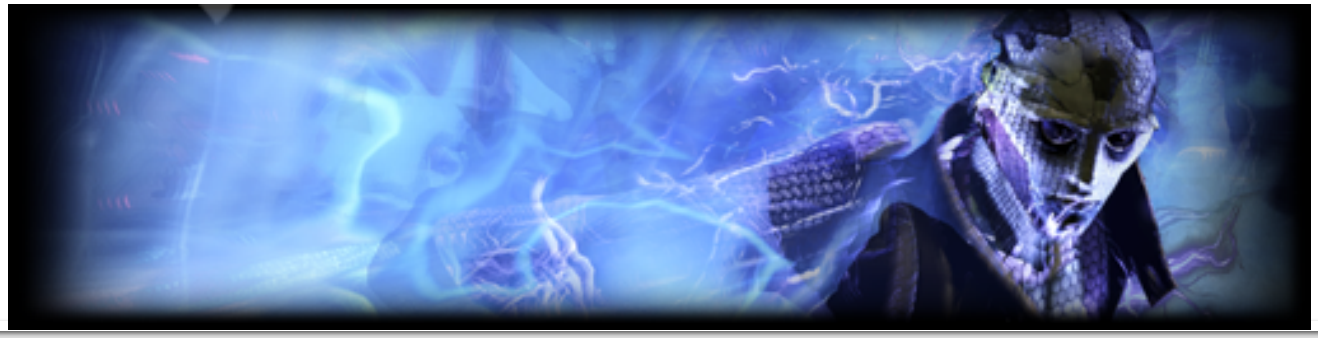
RIP LC's /usr/local: Enabling a tri-lab common environment (TCE)

- New 'local' packaging system -> new opportunities
  - Tri-lab common environment, MPI reorganization, Lmod modules, oh my!
- Goal: Enable tri-labs to install same software in same location
  - /usr/tce chosen as name-conflict-free blank slate (/usr/llnl-rocks had my vote)
  - LLNL only populating /usr/tce in TOSS3, not /usr/local
  - Unlikely that TOSS2 MPI executables can be made to work on CTS-1 running TOSS3
    - Might work if use mvapich2 v2.2 and Intel 16
    - Might be possible on TLCC2 w/TOSS3 (Zin)
      - Note: patchelf can change and lengthen rpath
      - Talk to me if want to attempt this
- /opt has different goals (avoid IMHO)
  - Functional common testing environment
  - Uses Linux's anti-rpath model
  - Versions can change with TOSS updates



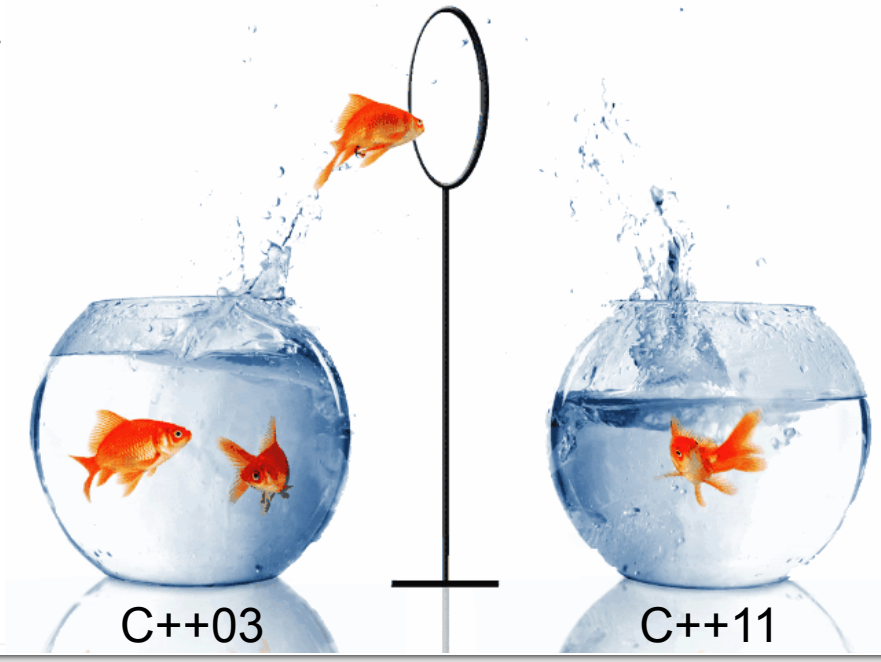
You can lead a horse to /usr/tce-laced Kool-Aid...

- We cannot force other labs to switch to using /usr/tce
  - DEG-equivalent folks from LANL and Sandia participated in its design
    - We worked to enable different usage models as much as possible
  - The change is as significant for their users as it is for our users
  - We believe there is strong enticements for using /usr/tce (we think it will happen)
- Just installing a requested subset of /usr/tce would enable awesomeness
  - TOSS3 binaries built here could just run there
  - Build there using same hard-coded paths to /usr/tce compilers and MPIs
  - Their users could just pretend /usr/tce simply doesn't exist!
- New /usr/projects -> /usr/apps symlink at LLNL enables common tri-lab data paths
  - Sandia expected to add /usr/projects -> /projects symlink also



Now with 50% less hoop jumping for working Intel C++11 support!

- Made gcc 4.9.3 default used by Intel and by LC TOSS3 users
  - C++11 support in icpc needed gcc 4.9.3 headers or later to work well
    - RHEL 7's gcc 4.8.5 default just doesn't cut it (normally LC uses system default)
  - Using path to pick g++ caused no end of problems (so now ignores path)
    - Use icpc -gxx-name=/usr/tce/packages/gcc/gcc-4.9.3/bin/g++ to change g++ version
  - Still need -std=c++11 option for C++11
- Change in behavior from TOSS2
  - Many C++11 users bit by old scheme



An exponential number of MPI builds, all for you

- Some TOSS2 compiler compatibility assumptions bit some folks later on
  - We lived on the edge in TOSS2 by using same MPI build for entire compiler family
    - C++ ABI is crazy-unstable but C++ MPI interface were disabled at LLNL which helped
    - Some MPIs are now written in C++(!), which can bite you even if you just use C++
    - Fortran ABIs change periodically and recently causing weird MPI problems in FORTRAN
- Buying disk space is easy, figuring out ABI compatibility matrix is hard!
  - MPI headers made incompatible with even minor version changes (ARRG!)
  - Solution: MPI build for each MPI/Compiler version (exponentially growth)
    - Newer MPIs might only be built with newer or popular compiler versions (less growth)

The great thing about naming convention standards is there are so many to choose from!

- MPI implementers moving to standard MPI compiler wrapper names
  - Many TOSS2 mvapich1-specific MPI wrapper names no longer exist on TOSS3
  - Use only these wrapper names to work across mvapich2, Open MPI, and Intel MPI
    - C: mpicc (C), mpicxx (C++), mpif77 (FORTRAN77), mpif90 (FORTRAN 90 and later)
- Path to MPI wrapper now completely specifies MPI and Compiler version
  - MPI wrappers ignores PATH when selecting compiler to prevent terribad build issues
    - /usr/tce/packages/mvapich2/mvapich2-2.2-intel-16.0.3/bin/mpicxx
    - /usr/tce/packages/openmpi/openmpi-2.0.0-pgi-16.3/bin/mpicxx
    - /usr/tce/packages/impi/impi-5.1.3-gcc-4.9.3/bin/mpicxx
  - Recommend hardcoding full paths to MPI wrappers in your build system
    - Makes builds independent MPI and compilers selected in current environment

Modules? Really?
Lmod and working in a post-dotkit world

- TACC's Lmod module implementation won our comprehensive bakeoff
  - Designed to navigate large number of compiler and MPI-specific packages
    - We project an impressive number in five years by the time TOSS3 retires
  - Growing Lmod user base and active support from TACC
  - Used by /opt in TOSS3 and supports most 'classic' env modules (which /opt mostly uses)
- Mixing dotkits with modules can lead to strange and confusing states
  - Recommend porting user dotkits to modules (it is pretty straightforward)
    - We can also help you mix modules with dotkits if you really need to
  - LC Hotline or DEG can help put user modules in /usr/apps/modulefiles (not world writable)
- Mixing /usr/tce and /opt modules can lead to strange and confusing states
  - Using just one set best (usually /usr/tce). Please let us know if you feel you need to mix them!

Look at all the pretty modules

- ‘module list’ shows what modules are currently loaded
  - Default: Intel/16.0.3, mvapich2/2.2, and StdEnv (adds /usr/tce/bin, etc.)
- ‘module avail’ lists modules available for loading
  - Includes modules specific to currently selected compiler and MPI
    - Because we default to a specific compiler and MPI, these typically are visible

```
---------------------- /usr/tce/modulefiles/MPI/intel/16.0.3/mvapich2/2.2 -------------------
   fftw/3.3.4 (D)

------------------------- /usr/tce/modulefiles/Compiler/intel/16.0.3 --------------------------
   impi/5.1.3    mvapich2/2.2 (L)    openmpi/1.10.2    openmpi/2.0.0 (D)

---------------------------------- /usr/tce/modulefiles/Core ----------------------------------
   StdEnv              (L)    intel/14.0.3
   gcc/4.8-redhat             intel/15.0.6
   gcc/4.9.3           (D)    intel/16.0.2
   gcc/6.1.0                  intel/16.0.3    (L,D)
   |lines 1-12|
```

← **Don’t hit control-C here, hit ‘q’**

Note: Hitting ^C in middle of ‘module avail’ puts tcsh in odd history-free state

The magic of Lmod

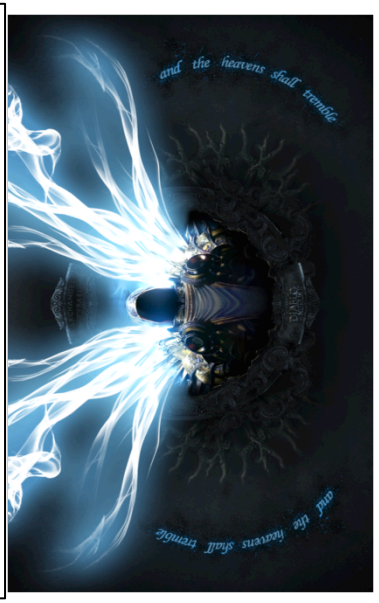
- ‘module load package1 <package2>’ load packages, updates dependences

```
> which mpicc
/usr/tce/packages/mvapich2/mvapich2-2.2-intel-16.0.3/bin/mpicc

> module load gcc/4.9.3

Due to MODULEPATH changes the following have been reloaded:
  1) mvapich2/2.2

> which mpicc
/usr/tce/packages/mvapich2/mvapich2-2.2-gcc-4.9.3/bin/mpicc
```



- ‘module save’ saves new default login state in ~/.lmod.d/default
  - Can also save and load named package sets if desired (‘module save app1’)
- ‘module restore’ puts modules back to just logged in state

JUST SAY NO to using MPI & compiler modules with complex builds

- Using modules or dotkits with complex builds is asking for big trouble
  - Build systems sometimes spawn new shells that revert to default environments
    - Already have seen this problem with early TOSS3 adopters (and on TOSS2)
    - If you have to modify your dotfiles to successfully build, you may have this problem
  - Typing 'make' in wrong window creates bad .o files deep in tree
    - Plague for developers working on multiple projects using different compilers
    - Full clean rebuilds become required first step when weird problems happen
- Use modules to find full paths to compilers and then use the full path
  - E.g., /usr/tce/packages/mvapich2/mvapich2-2.2-gcc-4.9.3/bin/mpicxx
  - E.g., /usr/tce/packages/gcc/gcc-4.9.3/bin/g++
  - /usr/tce's compilers and MPI wrappers don't require modules be used
- Using modules for quick and easy builds much less of a problem

Avoid the mutant giant spider dog bite!

- 'Module spider' will find "hidden" /opt compiler and MPI modules
  - 'Module avail' usually better choice than 'module spider'
    - Default MPI and compilers enable this
  - 'Module keyword' will also list /opt modules

```
>  module spider intel
Versions:
       intel/14.0.3
       intel/15.0          <- from /opt, not /usr/tce compatible
       intel/15.0.6
       intel/16.0          <- from /opt, not /usr/tce compatible
       intel/16.0.2
       intel/16.0.3
```

```
> module spider intel/16.0
----------------------------------------------------------------------------
  intel: intel/16.0
----------------------------------------------------------------------------
    You will need to load all module(s) on any one of the lines below before the "intel/16.0" module is available to load.
       opt          <- Don't use compiler and MPI's from /opt (other tools may be ok but tell us so we can add to /usr/tce)
```

This one weird trick will have you never typing module again!

- ml: A convenient tool (included with lmod)
  - **ml** means *module list* (what's loaded)
  - **ml avail** means *module avail* (what can be loaded)
  - **ml foo** means *module load foo*
  - **ml -bar** means module unload bar
  - **ml foo -bar** means *module unload bar; module load foo*
  - **ml** can be used everywhere *module* can be
    - Considered a bug if you ever have to type 'module' again
    - **ml swap intel gcc** (swap intel compiler module for default gcc compiler)
    - **ml show foo** (show what module foo will do)
    - **ml whatis foo** (show description of module foo)
    - **ml keyword intel** (show modules with intel in the description text)
  - See **http://lmod.readthedocs.io/en/latest/010_user.html**

Questions?


Lawrence Livermore
National Laboratory