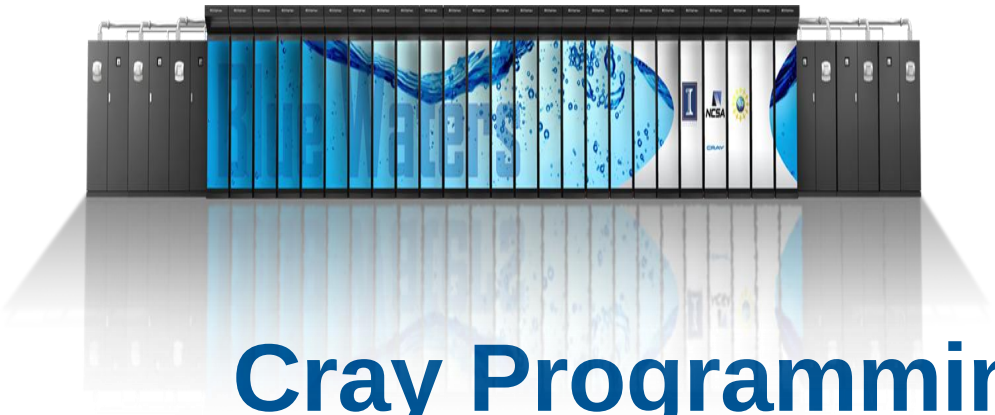
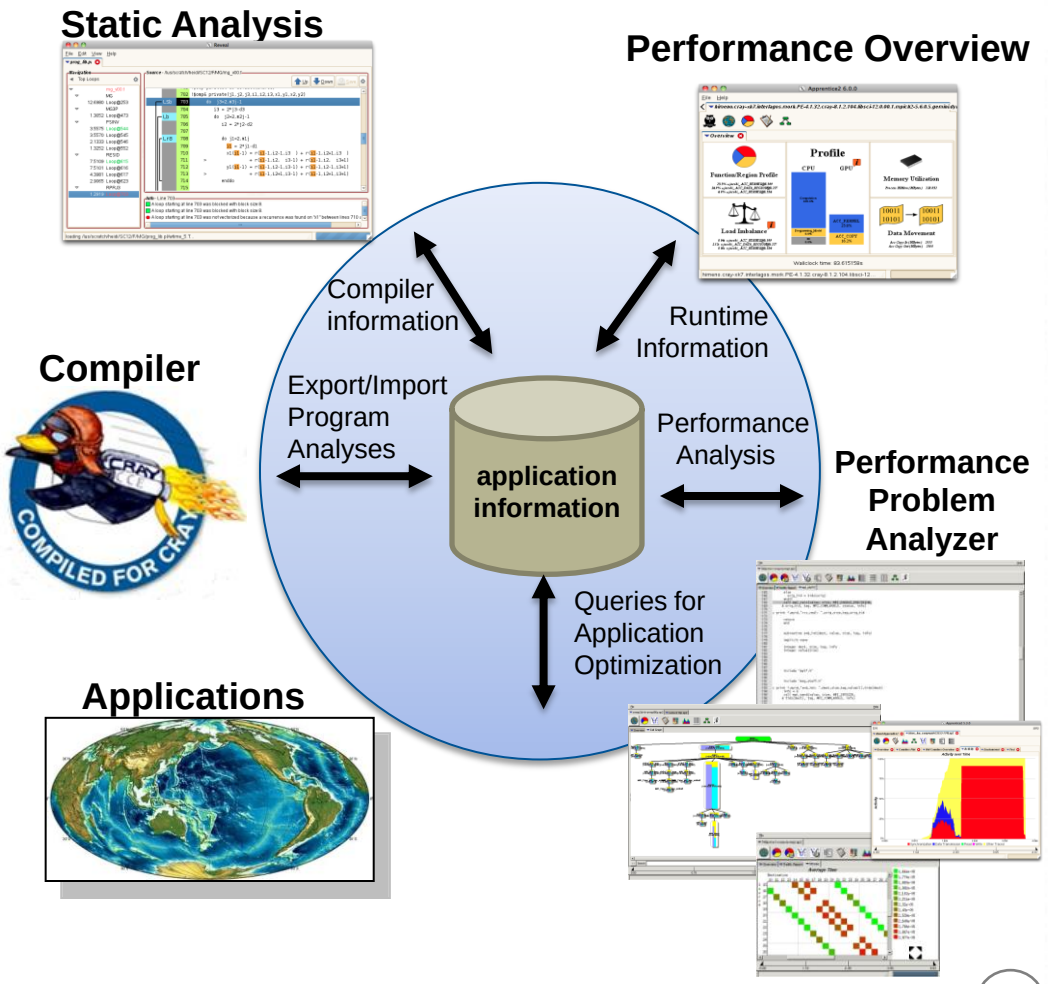


Cray Programming Environment on Blue Waters

Luiz DeRose
Sr. Principal Engineer
Programming Environments Director
Cray Inc.

Cray Programming Environment Mission

- It is the role of the Programming Environment to **close the gap** between observed performance and achievable performance
- Provide a **tightly coupled** programming environment with compilers, libraries, and tools that will **hide the complexity** of the system
  - Address issues of scale and complexity of HPC systems
  - Target **ease of use** with extended **functionality** and increased **automation**
  - Close **interaction with users**
    - For feedback targeting functionality enhancements



Cray Programming Environment
Focus on Performance and Productivity

Programming Languages

- Fortran
- C
- C++
- Python

Programming models

- Distributed Memory (Cray MPT)
  - MPI
  - SHMEM
- Shared Memory
  - OpenMP 3.0
  - OpenACC
- PGAS & Global View
  - UPC (CCE)
  - CAF (CCE)
  - Chapel

Compilers

- Cray Compiling Environment (CCE)
- GNU
- 3rd Party Compilers

Tools

- Environment setup
  - Modules
- Debuggers
  - TotalView
  - DDT
  - lgdb
- Debugging Support Tools
  - Abnormal Termination Processing
  - STAT
- Performance Analysis
  - CrayPat
  - Cray Apprentice²
- Scoping Analysis
  - Reveal

Optimized Scientific Libraries

- LAPACK
- ScaLAPACK
- BLAS (libgoto)
- Iterative Refinement Toolkit
- Cray Adaptive FFTs (CRAFFT)
- FFTW
- Cray PETSc (with CASK)
- Cray Trilinos (with CASK)

I/O Libraries

- NetCDF
- HDF5

Cray developed
Licensed ISV SW
3rd party packaging
Cray added value to 3rd party

Cray Programming Environment Roadmap

The Cray Compiling Environment

- **Cray technology focused on scientific applications**
  - Takes advantage of **automatic vectorization**
  - Takes advantage of **automatic shared memory parallelization**

- **PGAS languages (UPC & Fortran Coarrays) fully optimized and integrated into the compiler**
  - UPC 1.2 and Fortran 2008 coarray support
  - No preprocessor involved
  - Target the network appropriately
  - Full debugger support with Allinea's DDT

- **Standard conforming languages and programming models**
  - **Fortran 2008 standard compliant**
  - C++98/2003 compliant (working on C++11)
  - OpenACC 1.0 (working on OpenACC 2.0)
  - OpenMP 3.1 compliant (working on OpenMP 4.0)

Cray MPI & Cray SHMEM

- **MPI**
  - Implementation based on MPICH2 from ANL
  - Optimized Remote Memory Access (one-sided) fully supported including passive RMA
  - Full MPI-2 support with the exception of
    - Dynamic process management (MPI_Comm_spawn)
  - MPI3 Forum active participant

- **Cray SHMEM**
  - Fully optimized Cray SHMEM library supported
    - Cray implementation close to the T3E model
    - Cray XE & XC implementation on top of the Distributed Memory Applications API (DMAPP)
  - Later enhancements include:
    - Leveraging local memory access through Cross Process Memory Mapping (XPMEM)
      - Provides the ability for one process to map arbitrary portions of another local process
    - Distributed locking
    - Collectives optimization

Debugging on Cray Systems

- **Systems with thousands of threads of execution need a new debugging paradigm**
- **Cray's focus is to build tools around traditional debuggers with innovative techniques for productivity and scalability**
  - Support for traditional debugging mechanism
    - RogueWave TotalView and Allinea DDT
  - Scalable Solutions based on MRNet from University of Wisconsin
    - **STAT - Stack Trace Analysis Tool**
      - Scalable generation of a single, merged, stack backtrace tree
    - **ATP - Abnormal Termination Processing**
      - Scalable analysis of a sick application, delivering a STAT tree and a minimal, comprehensive, core file set.
  - lgdb 2.0
    - Ability to see data from multiple processors in the same instance of lgdb
      - without the need for multiple windows
    - **Comparative debugging**
      - A **data-centric paradigm** instead of the traditional control-centric paradigm
      - Collaboration with Monash University and University of Wisconsin for scalability
  - Fast Track Debugging
    - Debugging optimized applications
    - Added to Allinea's DDT 2.6 (June 2010)

Cray Performance Analysis Tools

- From performance measurement to performance analysis
- Assist the user with application performance analysis and optimization
  - Help user identify important and meaningful information from potentially massive data sets
  - Help user identify problem areas instead of just reporting data
  - Bring optimization knowledge to a wider set of users
- Focus on ease of use and intuitive user interfaces
  - Automatic program instrumentation
  - Automatic analysis
- Target scalability issues in all areas of tool development

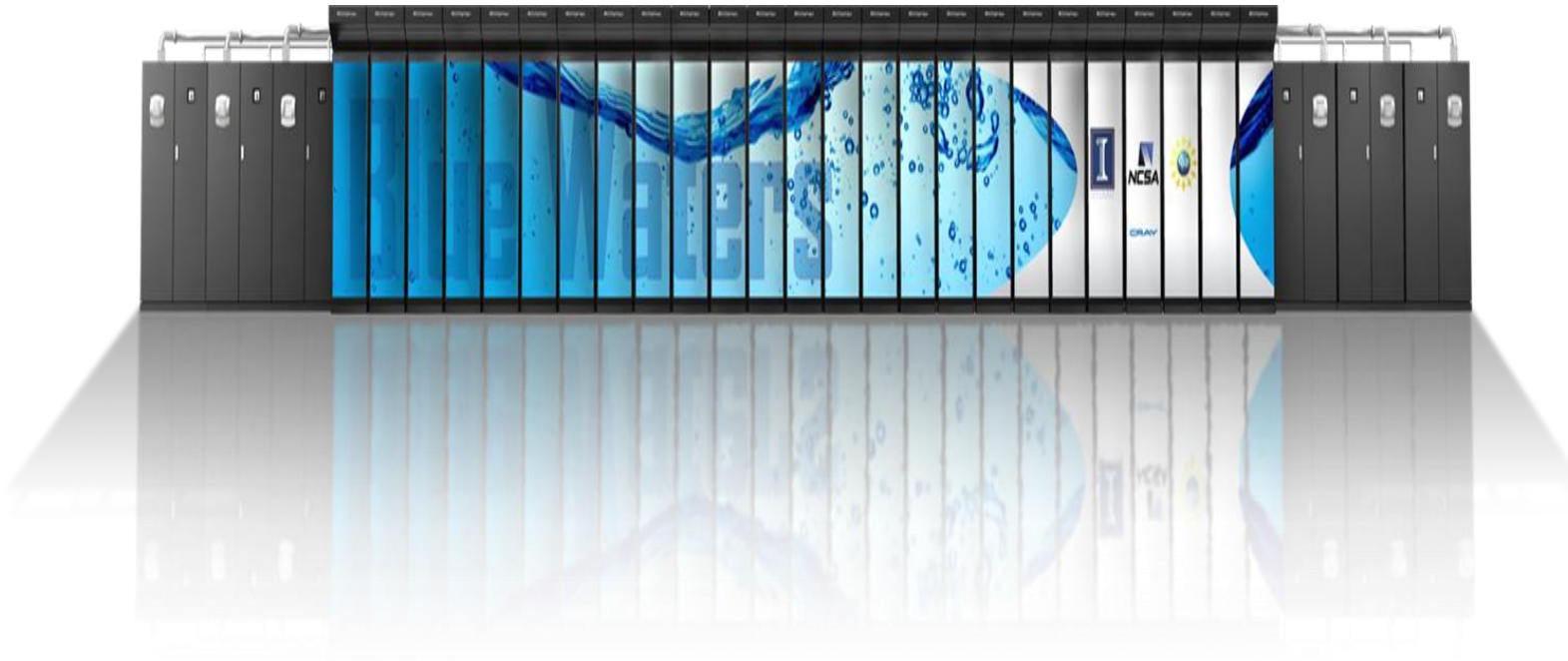
Cray Adaptive Scientific Libraries

- **Scientific Libraries today have three concentrations to increase productivity with enhanced performance**
  - **Standardization**
  - **Autotuning**
  - **Adaptive Libraries**

- **Cray adaptive model**
  - Runtime analysis allows **best** library/kernel to be **used dynamically**
  - Extensive offline testing allows **library to make decisions** or remove the need for those decisions
  - Decision depends on the system, on previous performance info, obtained previously, and characteristics of calling problem

- **What makes Cray libraries special:**
  - **Node performace**
  - **Network performance**
  - **Highly adaptive software**

Environment Setup



Environment Setup

- **The Cray systems use *modules* in the user environment to support multiple software versions and to create integrated software packages**
  - As new versions of the supported software and associated man pages become available, they are added automatically to the Programming Environment, while earlier versions are retained to support legacy applications
  - The modules tool is used to handle different versions of packages. You can use the default version of a product, or choose another version
    - e.g.: module load compiler_v1
    - e.g.: module swap compiler_v1 compiler_v2
    - e.g.: module load perftools
- **Modules take care of changing of PATH, MANPATH, LM_LICENSE_FILE,**
  - Modules also provide a simple mechanism for updating certain environment variables, such as PATH, MANPATH, and LD_LIBRARY_PATH
  - In general, you should make use of the modules system rather than embedding specific directory paths into your startup files, makefiles, and scripts
- **It is also easy to setup your own modules for your own software**

module list

```
derose@jyc1:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.7
  2) nodestat/2.2-1.0401.37252.2.1.gem
  3) sdb/1.0-1.0401.38148.4.27.gem
  4) MySQL/5.0.64-1.0000.5053.22.1
  5) lustre-cray_gem_s/1.8.6_2.6.32.59_0.7.1_1.0401.6845.9.1-1.0401.40514.13.1
  6) udreg/2.3.2-1.0401.5929.3.3.gem
  7) ugni/4.0-1.0401.5928.9.5.gem
  8) gni-headers/2.1-1.0401.5675.4.4.gem
  9) dmapp/3.2.1-1.0401.5983.4.5.gem
 10) xpmem/0.1-2.0401.36790.4.3.gem
 11) hss-llm/7.0.0
 12) Base-opts/1.0.2-1.0401.35378.1.3.gem
 13) craype-network-gemini
 14) xt-asyncpe/5.17
 15) cce/8.1.5.102
 16) xt-libsci/12.0.00
 17) pmi/4.0.1-1.0000.9421.73.3.gem
 18) rca/1.0.0-2.0401.38656.2.2.gem
 19) atp/1.6.1
 20) PrgEnv-cray/4.1.40
 21) moab/7.1.3-r3-b12-SUSE11
 22) torque/4.1.5-snap.201301211739
 23) xtpe-interlagos
 24) cray-mpich2/5.6.1
 25) java/jdk1.6.0_24
 26) globus/5.2.0
 27) altd/altd
 28) scripts
 29) user-paths
```

The **Base-opts** modules is loaded by default into your user environment

You should
never unload the Base-opts module
it contains the setup for CLE

What is xtpe-arch?

```
derose@jyc1:~> module show xtpe-interlagos
-------------------------------------------------------------------
/opt/cray/xt-asyncpe/default/modulefiles/xtpe-interlagos:

conflict        xtpe-barcelona
conflict        xtpe-quadcore
conflict        xtpe-shanghai
conflict        xtpe-istanbul
conflict        xtpe-interlagos-cu
conflict        xtpe-mc8
conflict        xtpe-mc12
conflict        xtpe-xeon
prepend-path    PE_PRODUCT_LIST XTPE_INTERLAGOS
setenv          XTPE_INTERLAGOS_ENABLED ON
setenv          CRAY_CPU_TARGET interlagos
setenv          INTEL_PRE_COMPILE_OPTS  -msse3
-------------------------------------------------------------------
```

It'd probably be a really bad idea to load two architectures at once

I should build for the right compute-node architecture

Which SW Products and Versions Are Available

- **module avail [avail-options] [path...]**
  - List all available modulefiles in the current MODULEPATH
- **Useful options for filtering**
  - -U, --usermodules
    - List all modulefiles of interest to a typical user
  - -D, --defaultversions
    - List only default versions of modulefiles with multiple available versions
  - -P, --prgenvmodules
    - List all PrgEnv modulefiles
  - -T, --toolmodules
    - List all tool modulefiles
  - -L, --librarymodules
    - List all library modulefiles
  - % module avail <product>
    - List all <product> versions available

module avail -U -D

```
derose@jyc1:~> module avail -U -D

--------------------------- /opt/cray/gem/modulefiles ---------------------------
blcr/0.8.4-1.0401.655.4.2.gem(default)
pmi/4.0.1-1.0000.9421.73.3.gem(default)

---------------------------- /opt/cray/modulefiles ----------------------------
atp/1.6.1(default)                  netcdf/4.2.0(default)
cray-libsci/12.0.00(default)        netcdf-hdf5parallel/4.2.0(default)
cray-mpich2/5.6.1(default)          ntk/1.5.0(default)
cray-petsc/3.3.03(default)          onesided/1.5.0(default)
cray-petsc-complex/3.3.03(default) papi/5.0.1(default)
cray-shmem/5.6.1(default)           parallel-netcdf/1.3.1(default)
cray-tpsl/1.3.01(default)           perftools/6.0.2.10548(default)
cray-trilinos/10.12.1.1(default)    petsc/3.3.00(default)
cudatoolkit/5.0.35.102(default)     petsc-complex/3.3.00(default)
fftw/3.3.0.1(default)               tpsl/1.3.00(default)
ga/5.0.2(default)                   trilinos/10.12.1.0(default)
hdf5/1.8.8(default)                 xt-lgdb/1.5(default)
hdf5-parallel/1.8.8(default)        xt-libsci/12.0.00(default)
iobuf/2.0.3(default)                xt-mpich2/5.6.1(default)
libfast/1.0.9(default)              xt-shmem/5.6.1(default)
libsci_acc/2.0.00.9(default)

------------------------------ /opt/modulefiles ------------------------------
PrgEnv-cray/4.1.40(default)    gcc/4.7.2(default)
PrgEnv-gnu/4.1.40(default)     java/jdk1.7.0_07(default)
PrgEnv-intel/4.1.40(default)   mrnet/3.0.0(default)
PrgEnv-pgi/4.1.40(default)     pathscale/4.0.12.1(default)
acml/5.2.0(default)            petsc/3.3.00(default)
cce/8.1.5.102(default)         petsc-complex/3.3.00(default)
chapel/1.4.0(default)          pgi/12.10.0(default)
ddt/3.2.1_26174(default)       xt-asyncpe/5.17(default)
fftw/3.3.0.1(default)
```

Which Software Versions Are Available?

```
derose@jyc1:~> module avail perftools

---------------------------- /opt/cray/modulefiles ----------------------------
--
perftools/5.3.0                 perftools/6.0.0
perftools/5.3.0.8395            perftools/6.0.1
perftools/5.3.1                 perftools/6.0.2.10548(default)
perftools/5.3.2


derose@jyc1:~> module avail cce

------------------------------ /opt/modulefiles -------------------------------
--
cce/7.4.5                 cce/8.0.7               cce/8.1.0.170
cce/8.0.0                 cce/8.1.0               cce/8.1.1
cce/8.0.1                 cce/8.1.0.144           cce/8.1.2
cce/8.0.2                 cce/8.1.0.157           cce/8.1.3
cce/8.0.3                 cce/8.1.0.160           cce/8.1.4
cce/8.0.4                 cce/8.1.0.165           cce/8.1.5.102(default)
cce/8.0.5                 cce/8.1.0.166
cce/8.0.6                 cce/8.1.0.168
```

What Happens When I Load a Module?

```
derose@jyc1:~> module show perftools
-------------------------------------------------------------------
/opt/cray/modulefiles/perftools/6.0.2.10548:

setenv           PERFTOOLS_VERSION 6.0.2.10548
conflict         x2-craypat
conflict         xt-papi
conflict         papi
conflict         craypat
conflict         xt-craypat
conflict         apprentice2
conflict         cuda
conflict         cudatools
module           load rca
setenv           PAT_BUILD_PAPI_BASEDIR /opt/cray/papi/5.0.1
setenv           CHPL_CG_CPP_LINES 1
setenv           PDGCS_LLVM_DISABLE_FP_ELIM 1
setenv           PDGCS_DEBUG quiet,mthrd=0400
setenv           PAT_REPORT_PRUNE_NAME
_cray$mt_start_,__cray_hwpc_,f_cray_hwpc_,cstart,__pat_,pat_region_,PAT_,OMP.slave_loop,slave_entry,_new_slave_entry,__libc
_start_main,_start,__start,start_thread,__wrap_,UPC_ADIO_,_upc_,upc_,__caf_,__pgas_,syscall
module-whatis    Perftools - the Performance Tools module sets up environments for CrayPat, Apprentice2 and Reveal
prepend-path     PATH /opt/cray/perftools/6.0.2.10548/bin
prepend-path     MANPATH /opt/cray/perftools/6.0.2.10548/man
setenv           CRAYPAT_LICENSE_FILE /opt/cray/perftools/craypat.lic
prepend-path     CRAYLMD_LICENSE_FILE /opt/cray/perftools/craypat.lic
setenv           CRAYPAT_ROOT /opt/cray/perftools/6.0.2.10548
setenv           CRAYPAT_INCLUDE_OPTS  $($CRAYPAT_ROOT/sbin/pat-opts INCLUDE)
setenv           CRAYPAT_PRE_LINK_OPTS  $($CRAYPAT_ROOT/sbin/pat-opts PRE_LINK)
setenv           CRAYPAT_POST_LINK_OPTS  $($CRAYPAT_ROOT/sbin/pat-opts POST_LINK)
setenv           CRAYPAT_PRE_COMPILE_OPTS  $($CRAYPAT_ROOT/sbin/pat-opts PRE_COMPILE)
setenv           CRAYPAT_POST_COMPILE_OPTS  $($CRAYPAT_ROOT/sbin/pat-opts POST_COMPILE)
setenv           CRAYPAT_ROOT_FOR_EVAL /opt/cray/perftools/6.0.2.10548
setenv           APP2_STATE 6.0.2.10548
setenv           JH_HELPSET /opt/cray/perftools/6.0.2.10548/help/app2help.jar
setenv           JH_VIEWER /opt/cray/perftools/6.0.2.10548/help/jh2_0_05/demos/bin/hsviewer.jar
prepend-path     CRAY_LD_LIBRARY_PATH /opt/cray/perftools/6.0.2.10548/lib64
append-path      CLASSPATH /opt/cray/perftools/6.0.2.10548/help/jh2_0_05/javahelp
append-path      PE_PRODUCT_LIST PERFTOOLS
append-path      PE_PRODUCT_LIST CRAYPAT
-------------------------------------------------------------------
```

Release Notes

```
derose@jyc1:~> module help cce/8.1.0

----------- Module Specific Help for 'cce/8.1.0' -----------------

The modulefile, cce, defines the system paths and environment
variables needed to run the Cray Compile Environment.

Type "module avail cce" to see if other versions of this product
are available on this system. Use "module switch" to change versions.


Cray Compiling Environment 8.1.0
================================
  CCE 8.1.0
  =========
  Purpose:
  --------
    The CCE 8.1.0 release provides Fortran, C, and C++ compilers for Cray XE
    and Cray XK systems.

    The Cray Compiling Environment 8.1.0 release provides the following
    key enhancements:
      - New features as specified by the 2008 Fortran standard. This compiler
        conforms to the Fortran 2008 standard (ISO/IEC 1539-1:2010).
      - Enhanced support for accelerators on Cray Systems, including complete
        support for the OpenACC Application Programming Interface, Version 1.0.
      - Performance improvements for Cray XE and Cray XK systems
      - Support for the AMD Abu Dhabi CPU
      - Support for the Intel Sandy Bridge CPU
      - Support for the NVIDIA Kepler GPU

    Additional details can be found in the:
      Cray Compiling Environment 8.1 Release Overview, S-5212-81.
```

Release Notes (cont.)

```
Bugs Closed with CCE 8.1.0 release:
   ----------------------------------
     770618  Interop code segfaults with struct function result
     772194  Auto-allocate of scalar character with -ew issues bad warning
     773042  Incorrect error return from aio_write when called from CCE
     775600  Cray FORTRAN bug regarding pointers and save'd common blocks
     780519  cache benchmark performance issue on interlagos
     781285  OpenACC - PTX codegen error with perftools
     782940  Enzo code give NaN with Cray compiler when -O1, -O2 and -O3 are
     . . .

Dependencies:
   -------------
     The CCE 8.1.0 release is supported on Cray XE systems that run the Cray
     Linux Environment (CLE) operating system, version 3.1 and later and on the
     Cray XK systems that run the CLE operating system, version 4.0 UP01 and
     later.

     The Cray Compiling Environment 8.1 release requires the following
     supporting asynchronous software products:
       Cray Compiler Drivers (xt-asyncpe) 5.12 or later
       GNU GCC 4.4.4 must be installed, but must NOT be the default GCC
       PMI 3.0.0 or later
       Cray Scientific Libraries (LibSci) 11.0.00 or later

     The Cray Compiling Environment 8.1 release requires the following minimum
     version if these products are used:
       PETSc 3.1.05 or later
       hdf5-netcdf 1.8 (HDF5 1.85 and netcdf 4.1.1)
       MPT 5.2.3 or later
       ACML 4.4.0 or later. To use ACML 5.0, gcc 4.6.1 must be installed.
       Cray Performance Measurement and Analysis Tools 5.3.0
       Cray Performance Measurement and Analysis Tools 6.0.0 is required for
         Reveal

Known Problem:
 --------------
   Bug: 788950 Static Linking warning message about "gethostbyname" with
        PMI 4.0 and CCE 8.1.0
   Currently PMI 4.0.0 has a dependency on calling "gethostbyname" for the
   application resiliency support API feature. If PMI 4.0.0 is used, the
   following message is generated when creating static executables:
```

Using Different Compilers

- **Should use PrgEnv-cray to take maximum advantage of the Cray Programming Environment**

- **To access a different compiler:**
  - Load the corresponding Programming Environment (PE) module
    - PrgEnv-cray for CCE (the default)
    - PrgEnv-pgi for PGI
    - PrgEnv-gnu for GNU
  - Once one of these is loaded, you can then select a compiler suite
    - CCE: **module avail cce**
    - PGI: **module avail pgi**
  - For GPU programming (CUDA, OpenACC...)
    - Make sure you target the GPU when building:
      - **Example: module load craype-accel-nvidia35**

Using the Compilers

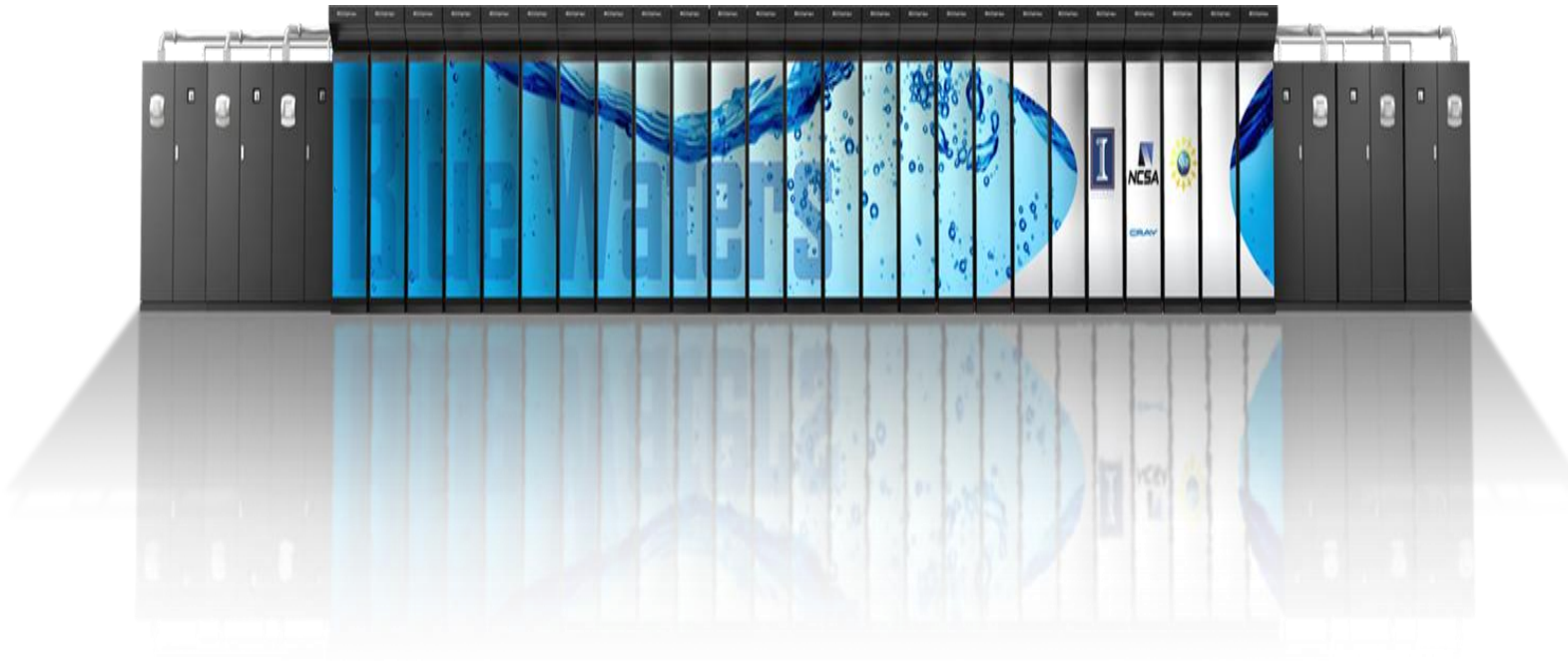
- **Cray Systems come with compiler wrappers to simplify building parallel applications (similar the mpicc/mpif90)**
  - Fortran Compiler: **ftn**
  - C Compiler: **cc**
  - C++ Compiler: **CC**
- **Using these wrappers ensures that your code is built for the compute nodes and linked against important libraries**
  - Cray MPT (MPI, Shmem, etc.)
  - Cray LibSci (BLAS, LAPACK, etc.)
  - …
- **Do not call the PGI, Cray, etc. compilers directly**
- **Cray Compiler wrappers try to hide the complexities of using the proper header files and libraries**

How About the –L and –l Flags

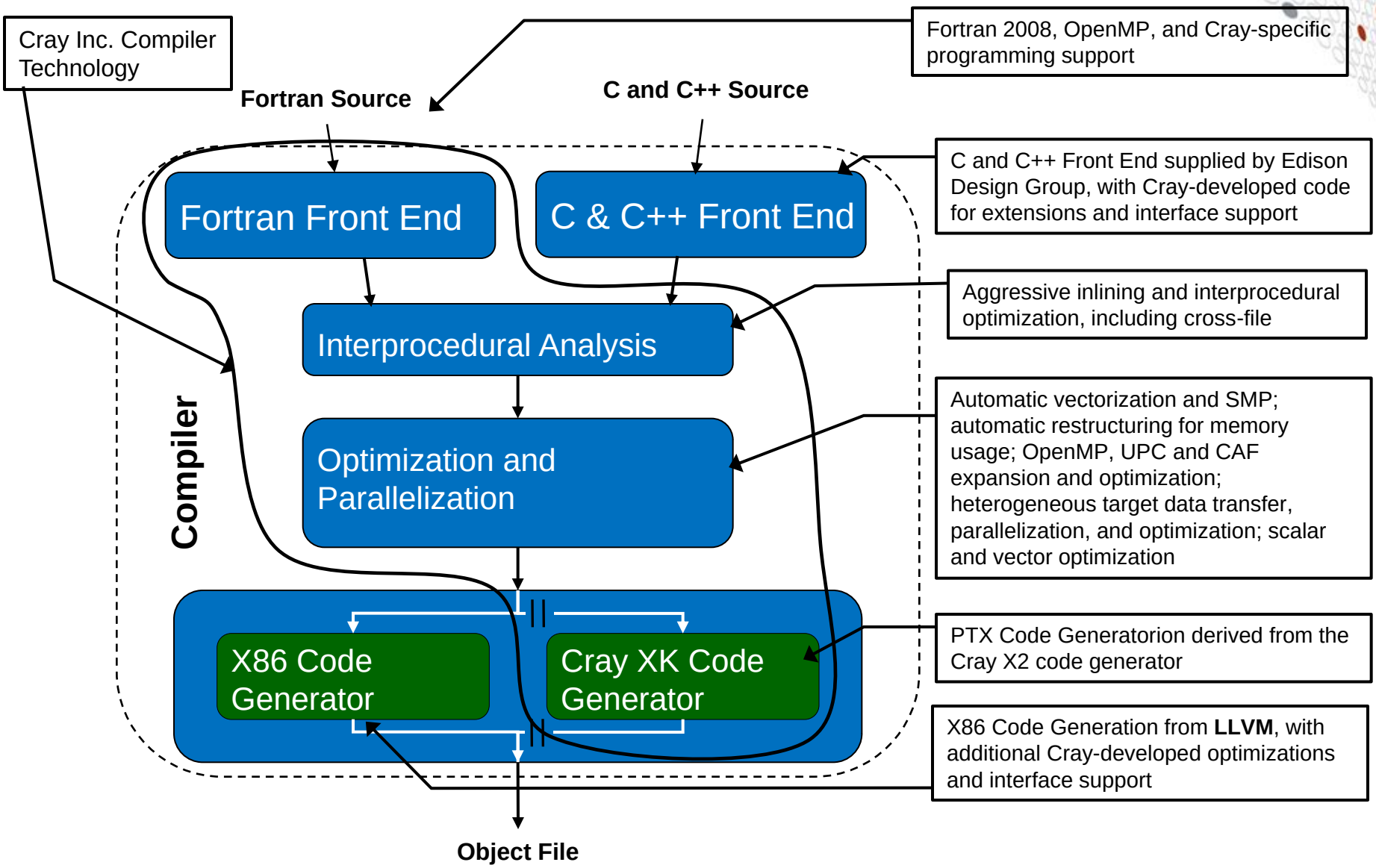
- **For libraries and include files being triggered by module files, you should NOT add anything to your Makefile**
  - No –lmpich is needed, nor should it be used
  - no –L is needed
  - Same is true for all Cray provided libraries
  - You don't need to deal with threaded vs non threaded math libs

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

The Cray Compiling Environment



CCE Technology Sources



CCE Main Features

- AMD Interlagos support, including AVX, FMA, and XOP instructions
- X86/NVIDIA compiler and library support
- OpenACC Version 1.0 support
- Full OpenMP 3.1 support
- Support for hybrid programming using MPI across node and OpenMP within the node
- Support for IEEE floating-point arithmetic and IEEE file formats
- Cray performance tools and debugger support
- Program Library
- CCE 8.1.0 was released on September 20, 2012
  - The full release overview can be found at: http://docs.cray.com/books/S-5212-81/

UPC and Fortran Coarray Features

- **C-based UPC and Fortran Coarray are PGAS language extensions, not stand-alone languages**
- **A subset of Fortran coarray collectives were added for CCE**
  - Although they are not yet part of the official language – they are too useful to be delayed
- **Significant improvements were made to the automatic use of blocked network transfers, including:**
  - Automatic conversion of multiple single-word accesses into blocked accesses
  - Improved capabilities for pattern matching to hand-optimized library routines, including messages stating what might be inhibiting the conversion
- **UPC and Fortran coarrays support up to 2,147,483,647 threads within a single application**
  - We actually did hit the previous limit of 65,535!

CCE Compiler Testing

- **Roughly 35,000 nightly regression tests run for Fortran (14,000), C (7,000), and C++ (14,000)**
  - Default optimization, but for multiple targets (X86, X86+AVX+FMA, X2, X86+NVIDIA), plus "debug" and "production" compiler versions
  - Additionally, cycle through "options testing" with the same test base
    - Fortran: -G0, -G1, -G2, -O0, -Oipa0, -Oipa5 -hpic, "-O3,fp3" –e0
    - C and C++: -Gn, -O0, -hipa0, -hipa5, -hpic, "-O3 –hfp3" -hzero
    - Additional tests and suites have been added for GPU testing
    - And some "stress test" option sets to create worse-case scenarios for the compiler
    - Other combinations as necessary and by request
- **Performance regression testing done weekly using important applications and benchmarks**
- **Functional and performance regressions typically use an automated system that isolates the change to a specific compiler or library mod**
- **Issues that are found as a result of testing but not immediately addressed have bugs opened to track them**

Inlining with CCE

- **Inlining is enabled by default**
  - Command line option –Oipa*n* (ftn) –hipa*n* (cc/CC) where n=0..4, provides a set of choices for inlining behavior
    - 0 - **All inlining and cloning** compiler directives **are ignored**
    - 1 - **Directive inlining**. Inlining is attempted for call sites and routines that are under the control of an inlining compiler directive.
      - **Cloning disabled and cloning directives are ignored**
    - 2 - **Call nest inlining**. Inline a call nest to an arbitrary depth as long as the nest does not exceed some compiler-determined threshold.
      - The expansion of the call nest must yield straight-line code (code containing no external calls) for any expansion to occur.
      - The call site must reside within the body of a loop for expansion to be attempted
      - **Cloning disabled and cloning directives are ignored**
    - **Constant actual argument inlining and tiny routine inlining**. for inlining
      - Includes levels 1 and 2, plus any call site that contains a constant actual argument
      - **Cloning disabled and cloning directives are ignored**
    - 4 - This includes levels 1, 2, and 3, plus routine cloning is attempted if inlining fails at a given call site. **Cloning directives are enabled**
- **Cross language inlining is not supported**

Some Cray Compilation Environment Basics

- **CCE-specific features:**
  - Optimization: **-O2** is the default and you should usually use this
  - OpenMP is supported by default (no flag needed to enable)
    - if you don't want it, use either **-hnoomp** or **-xomp** compiler flags
  - OpenACC is supported by default if GPU targeting module (craype-accel-nvidia*) is loaded
  - CCE only gives minimal information to stderr when compiling
    - To see more information, you should request a compiler listing file
      - flags **-ra** for ftn or **-hlist=a** for cc
      - writes a file with extension .lst
      - contains annotated source listing, followed by explanatory messages
    - Each message is tagged with an identifier, e.g.: **ftn-6430**
      - to get more information on this, type: **explain <identifier>**
    - Cray Reveal can display all this information (and more)

Recommended CCE Compilation Options

- **Use default optimization levels**
  - It's the equivalent of most other compilers –O3 or –fast
  - It is also our most thoroughly tested configuration
- **Using –O3,fp3 (or –O3 –hfp3, or some variation)**
  - -O3 only gives you slightly more than –O2
  - We also test this thoroughly
  - -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
  - Interlagos FMA usage is aggressive at –hfp2 and –hfp3; limited at –hfp1, and disabled at –hfp0
- **Do not use –Oipa5, -Oaggress, and so on – higher numbers are not always correlated with better performance**

What Exactly Does –hfp3 Do?

- **We recommend using –O3 –hfp3 if the application runs cleanly with these options**
- **-hfp3 primarily improves 32-bit floating point performance on the X86**
- **A partial list of what happens at –hfp3 is:**
  - Use of fast 32-bit inline division, reciprocal, square root, and reciprocal square root algorithms (with some loss of precision)
  - Use of a fast 32-bit inline complex absolute value algorithm
  - Starting with CCE 8.0, more aggressive reassociation (pre-8.0 –hfp2 behavior)
  - Various assumptions about floating point trap safety
  - Somewhat more aggressive about NaN assumptions
  - Assumes standard-compliant Fortran exponentiation (x**y)

Loopmark: Compiler Feedback

- **Compiler can generate an filename.lst file.**
  - Contains annotated listing of your source code with letter indicating important optimizations

```
%%%    L o o p m a r k   L e g e n d    %%%
Primary Loop Type      Modifiers
------- ---- ----      ---------
                       a - vector atomic memory operation
A  - Pattern matched   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           w - unwound
```

Example: Cray loopmark Messages

- **ftn –rm …** or **cc –hlist=m …**

```
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
```

Example: Cray loopmark messages (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.
```

Example of Explain Utility

derose@jyc1:~> 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
```

OpenMP

- **OpenMP is ON by default**
  - Optimizations controlled by **–hthread#**
- **Autothreading is NOT on by default;**
  - -hautothread to turn on
  - Modernized version of Cray X1 streaming capability
  - **Interacts with OpenMP directives**
- **If you do not want to use OpenMP and have OMP directives in the code, make sure to shut off OpenMP at compile time**
  - **To shut off** use **–hthread0** or **–xomp** or **–hnoomp**

Why Are CCE's Results Sometimes Different?

- **We do 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
- **We are also fairly aggressive at floating point optimizations that violate IEEE requirements**
  - -hfp[0-3] can control this, -hfp2 is the default, -hfp0 is close to IEEE conformance, but has significant performance implications
  - -hfp2 allows things like rewriting divisions as multiplication by reciprocal, floating point parallel reductions, simplified complex division algorithms, and so on
  - -hfp3 can be used for most applications and is tested often

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
- **GNU**
  - -O3 –ffast-math –funroll-loops
  - Compiler feedback: -ftree-vectorizer-verbose=2
  - man gfortran; man gcc; man g++

Compiler man Pages

- The cc(1), CC(1), and ftn(1) man pages contain information about the compiler driver commands
- The ***craycc(1), crayCC(1), and crayftn(1) man pages contain descriptions of the Cray compiler command options***
- The pgcc(1), pgCC(1), and pgf95(1) man pages contain descriptions of the PGI compiler command options
- The gcc(1), g++(1), and gfortran(1) man pages contain descriptions of the GNU compiler command options
- To verify that you are using the correct version of a compiler, use:
  - -V option on a cc, CC, or ftn command with PGI and CCE
  - --version option on a cc, CC, or ftn command with GNU

Cray Scientific Libraries Overview

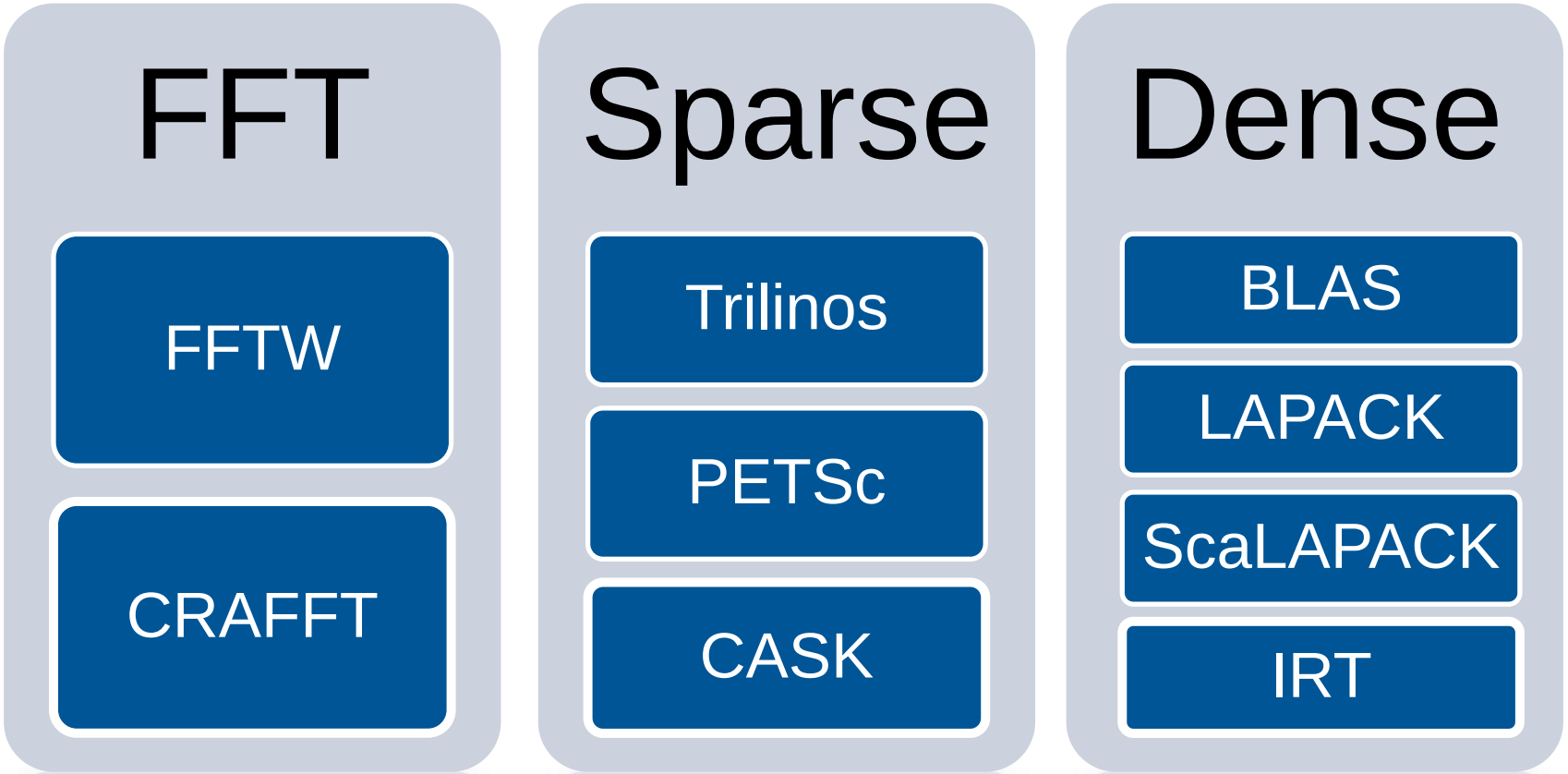
What are libraries for?

- Building blocks for writing scientific applications
- Historically – allowed the first forms of code re-use
- Later – became ways of running optimized code
- These days the complexity of the hardware is very high
- Cray PE insulates the user from that complexity
  - Cray module environment
  - CCE
  - Performance tools
  - Tuned MPI libraries (+PGAS)
  - Optimized Scientific libraries
- Cray scientific libraries are designed to give maximum possible performance from Cray systems with minimum effort

What makes Cray libraries special

1. **Node performance**
   - Highly tuned BLAS etc at the low-level
2. **Network performance**
   - Optimize for network performance
   - Overlap between communication and computation
   - Use the best available low-level mechanism
   - Use adaptive parallel algorithms
3. **Highly adaptive software**
   - Using auto-tuning and adaptation, give the user the known best (or very good) codes at runtime
4. **Productivity features**
   - Simpler interfaces into complex software

Cray Scientific Libraries – Functional View



Libsci Usage

- **LIbSci**
  - The drivers should do it all for you. Don't explicitly link.
  - For threads, set OMP_NUM_THREADS
    - Threading is used within libsci.
    - If you call within parallel region, single thread used
    - -Wl, -ydgemm_ reveals where the link was resolved
- **FFTW**
  - Module load fftw (there are also wisdom files you can pick up)
- **PETSc**
  - Module load petsc (or module load petsc-complex)
  - Use as you would your normal petsc build
- **Trilinos**
  - Module load trilinos
- **CASK – no need to do anything you get free optimization**

Threading

- **LibSci is compatible with OpenMP**
  - Control the number of threads to be used in your program using OMP_NUM_THREADS
  - e.g. in job script
  - setenv OMP_NUM_THREADS 16
  - Then run with aprun –n1 –d16
- **What behavior you get from the library depends on your code**
  - No threading in code
    - The BLAS call will use OMP_NUM_THREADS threads
  - Threaded code, outside parallel region
    - The BLAS call will use OMP_NUM_THREADS threads
  - Threaded code, inside parallel region
    - The BLAS call will use a single thread
- **Threaded LAPACK works exactly the same as threaded BLAS**
  - Anywhere LAPACK uses BLAS, those BLAS can be threaded
  - Some LAPACK routines are threaded at the higher level
  - No special instructions

Tuning Requests

- **CrayBLAS is an auto-tuned library**
  - Generally, excellent performance is possible for all shapes and sizes
- **However, even the adaptive CrayBLAS can be improved by tuning for exact sizes and shapes**
- **Send your specific tuning requirements to**

  crayblas@cray.com
- **Just send the routine name, and the list of calling sequences**

ScaLAPACK

- **ScaLAPACK in libsci is optimized for the Gemini interconnect**
  - New collective communication procedures are added
  - Default topologies are changed to use the new optimizations
  - Much better strong scaling
- **It also benefits from the optimizations in CrayBLAS**
- **IRT can provide further improvements**

Iterative Refinement Toolkit

- **Mixed precision can yield a big win on x86 machines.**
- **SSE (and AVX) units issue double the number of single precision operations per cycle.**
- **On CPU, single precision is always 2x as fast as double**
- **Accelerators sometimes have a bigger ratio**
  - Cell – 10x
  - Older NVIDIA cards – 7x
  - New NVIDIA cards (2x)
  - Newer AMD cards (> 2x)
- **IRT is a suite of tools to help exploit single precision**
  - A library for direct solvers
  - An automatic framework to use mixed precision under the covers

Iterative Refinement Toolkit - Library

- Various tools for solves linear systems in mixed precision
- Obtaining solutions accurate to double precision
  - For well conditioned problems
- Serial and Parallel versions of LU, Cholesky, and QR
- 2 usage methods
  - **IRT Benchmark routines**
    - Uses IRT 'under-the-covers' without changing your code
      - Simply set an environment variable
      - Useful when you cannot alter source code
  - **Advanced IRT API**
    - If greater control of the iterative refinement process is required
      - Allows
        - condition number estimation
        - error bounds return
        - minimization of either forward or backward error
        - 'fall back' to full precision if the condition number is too high
        - max number of iterations can be altered by users

IRT library usage

Decide if you want to use advanced API or benchmark API

benchmark API :

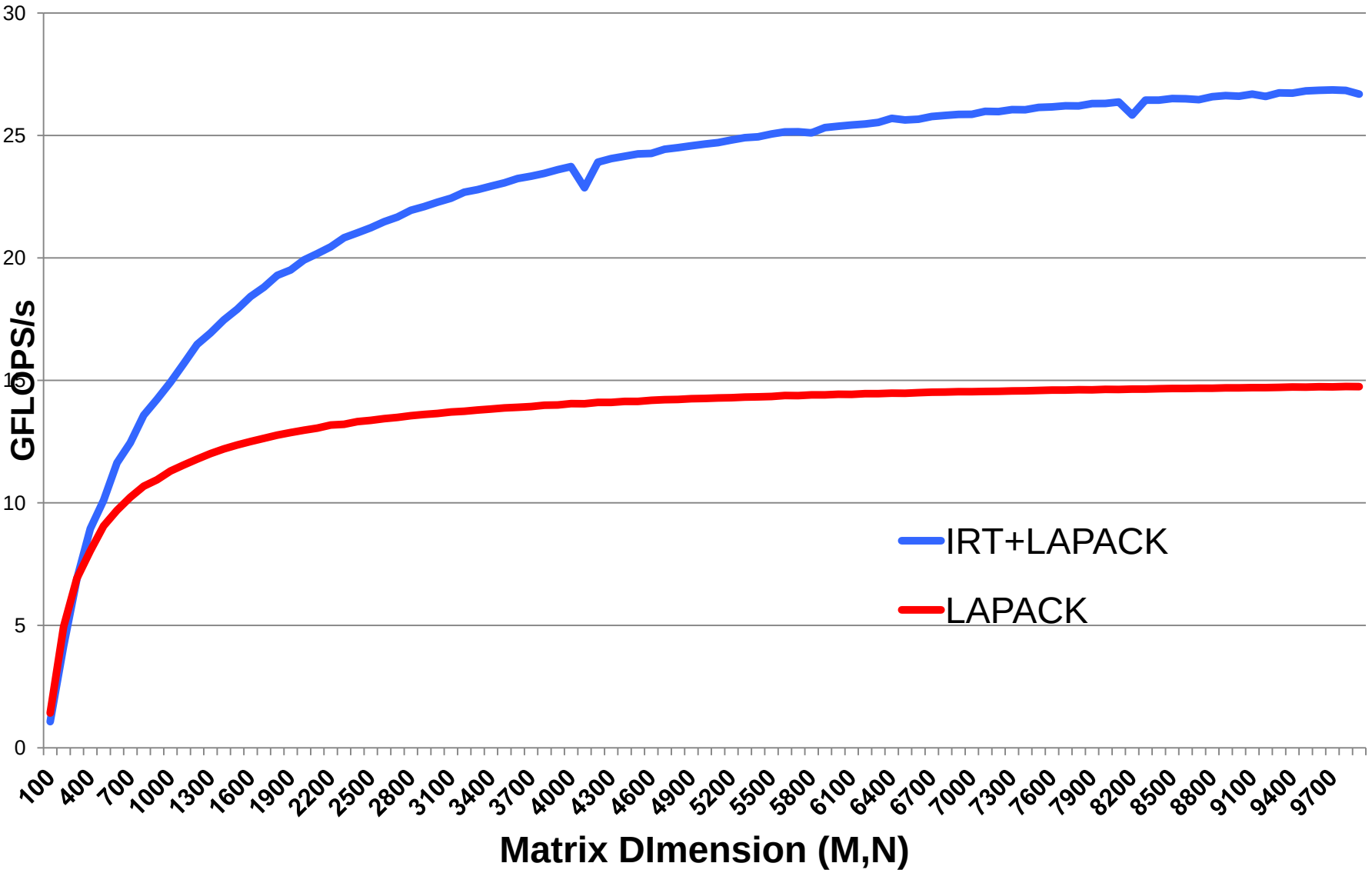
setenv IRT_USE_SOLVERS 1

advanced API :

1. **locate the factor and solve in your code (LAPACK or ScaLAPACK)**
2. **Replace factor and solve with a call to IRT routine**
   - e.g. dgesv -> irt_lu_real_serial
   - e.g. pzgesv -> irt_lu_complex_parallel
   - e.g pzposv -> irt_po_complex_parallel
3. **Set advanced arguments**
   - Forward error convergence for most accurate solution
   - Condition number estimate
   - “fall-back” to full precision if condition number too high

Note : “info” does not return zero when using IRT !!

IRT with LAPACK LU DGETRF
AMD Bulldozer 2.1 GHz
2threads :: September 2012



FFTW

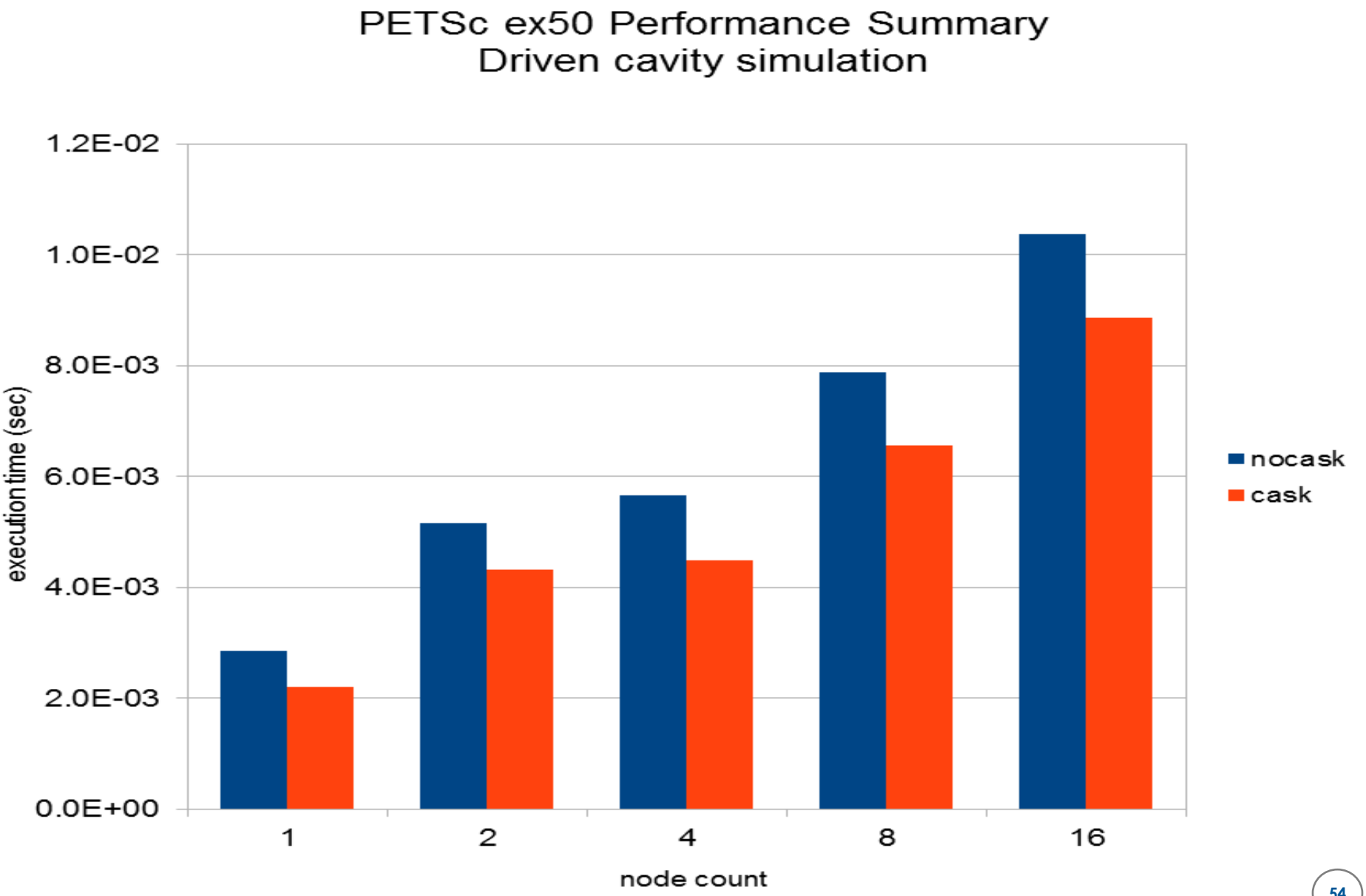
- **Cray's main FFT library is FFTW from MIT**
  - Some additional optimizations for Cray hardware
- **Usage is simple**
  - Load the module
  - In the code, call an FFTW plan
- **Cray's FFTW provides wisdom files for these systems**
  - You can use the wisdom files to skip the plan stage
  - This can be a significant performance boost
- **FFTW 3.3.0.1 includes Cray optimizations for Interlagos processors**

Cray Adaptive Sparse Kernel (CASK)

- **Sparse matrix operations in PETSc and Trilinos on Cray systems are optimized via CASK**

- **CASK is a product developed at Cray using the Cray Auto-tuning Framework**
  - Offline :
    - ATF program builds many thousands of sparse kernel
    - Testing program defines matrix categories based on density, dimension etc
    - Each kernel variant is tested against each matrix class
    - Performance table is built and adaptive library constructed
  - Runtime
    - Scan matrix at very low cost
    - Map user's calling sequence to nearest table match
    - Assign best kernel to the calling sequence
    - Optimized kernel used in iterative solver execution

CASK + PETSc AMD IL



PETSc, Linear System Solution

2D Laplacian Problem
Weak Scalability
N=262,144 --- 268M
AMD Bulldozer 2.1G :: July 2012