CRAY
THE SUPERCOMPUTER COMPANY

Cray Programming Environment

Overview

Luiz DeRose
Programming Environments Director
Cray Inc.

HP2C

CSCS
Swiss National Supercomputing Centre

The Cray Programming Environment Vision

- It is the role of the Programming Environment to **close the gap** between observed performance and peak performance
  - Help users achieve ***highest possible performance*** from the hardware

- The Cray Programming Environment is addressing the issues of scale and complexity of high end HPC systems with:
  - Increased **automation**
  - **Ease of use**
    - Hiding the system complexity
  - Extended **functionality**
    - Focus on scalability
  - Improved **Reliability**
  - Strong academic collaborations
  - Close **interaction with users**
    - For feedback targeting functionality enhancements

Cray Programming Environment Distribution
Focus on Differentiation and Productivity

Programming Languages

- Fortran
- C
- C++
- Chapel
- Python

Programming models

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

Compilers

- Cray Compiling Environment (CCE)
- PGI

Tools

Environment setup
- Modules

Debuggers
- TOTALVIEW
- DDT
- Igdb

Debugging Support Tools
- Fast Track Debugger (CCE w/ DDT
- Abnormal Termination Processing
- STAT
- Cray Comparative Debugger#

Performance Analysis
- CrayPat
- Cray Apprentice²

Optimized Scientific Libraries

- LAPACK
- ScaLAPCK
- 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
#: Under development
Licensed ISV SW
3rd party packaging
Cray added value to 3rd party

The Cray Compilation Environment

- Cray technology focused on scientific applications
  - Takes advantage of **automatic vectorization**
  - Takes advantage of **automatic shared memory parallelization**
- Standard conforming languages and programming models
  - **Fortran 2003 standard compliant** with F2008 features already available
  - C++98/2003 compliant
  - **OpenMP 3.0 compliant**, working on OpenMP 3.1 and OpenMP 4.0
- OpenMP and automatic multithreading fully integrated
  - Share the same runtime and resource pool
  - Aggressive loop restructuring and scalar optimization done in the presence of OpenMP
  - Consistent interface for managing OpenMP and automatic multithreading
- 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

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 XT implementation close to the T3E model

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

Debuggers on Cray Systems

- Systems with hundreds of thousands of threads of execution need a new debugging paradigm
  - Innovative techniques for productivity and scalability
    - Scalable Solutions based on MRNet from University of Wisconsin

      STAT - Stack Trace Analysis Tool
      - » Scalable generation of a single, merged, stack backtrace tree
        - ☞ running at 216K back-end processes

      ATP - Abnormal Termination Processing
      - » Scalable analysis of a sick application, delivering a STAT tree and a minimal, comprehensive, core file set.

    - Fast Track Debugging
      - o Debugging optimized applications
      - o Added to Allinea's DDT 2.6 (June 2010)

    - Comparative debugging
      - o A **data-centric paradigm** instead of the traditional control-centric paradigm
      - o Collaboration with Monash University and University of Wisconsin for scalability

  - Support for traditional debugging mechanism
    - TotalView, DDT, and gdb

Cray Scientific Libraries

IRT – Iterative Refinement Toolkit
CASK – Cray Adaptive Sparse Kernels
CRAFFT – Cray Adaptive FFT
CASE – Cray Adaptive Simplified Eigensolver

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

```
ldr@todi1:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.6
  2) nodestat/2.2-1.0400.29866.4.3.gem
  3) sdb/1.0-1.0400.30000.6.18.gem
  4) MySQL/5.0.64-1.0000.4667.20.1
  5) lustre-cray_gem_s/1.8.4_2.6.32.45_0.3.2_1.0400.6221.1.1-1.0400.30252.1.29
  6) udreg/2.3.1-1.0400.3911.5.6.gem
  7) ugni/2.3-1.0400.3912.4.29.gem
  8) gni-headers/2.1-1.0400.3906.5.1.gem
  9) dmapp/3.2.1-1.0400.3965.10.12.gem
 10) xpmem/0.1-2.0400.29883.4.6.gem
 11) hss-llm/6.0.0
 12) Base-opts/1.0.2-1.0400.29823.8.1.gem
 13) xtpe-network-gemini
 14) cce/8.0.0.135
 15) totalview-support/1.1.2
 16) xt-totalview/8.9.2
 17) acml/4.4.0
 18) xt-libsci/11.0.03
 19) pmi/2.1.4-1.0000.8596.8.9.gem
 20) rca/1.0.0-2.0400.30002.5.19.gem
 21) xt-asyncpe/5.03
 22) PrgEnv-cray/4.0.30
 23) xtpe-interlagos
 24) xt-mpich2/5.3.4
 25) slurm
```

Environment Setup

- 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
- The PrgEnv-cray is the default on Todi

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

```
ldr@todi1:~> module avail -U

--------------------------- /opt/cray/gem/modulefiles ---------------------------
blcr/0.8.2-1.0400.350.6.4.gem(default)
dmapp/3.2.1-1.0400.3965.10.12.gem(default)
gni-headers/2.1-1.0400.3906.5.1.gem(default)
pmi/2.1.4-1.0000.8596.8.9.gem(default)
rca/1.0.0-2.0400.30002.5.19.gem(default)
udreg/2.3.1-1.0400.3911.5.6.gem(default)
ugni/2.3-1.0400.3912.4.29.gem(default)
xpmem/0.1-2.0400.29883.4.6.gem(default)

---------------------------- /opt/cray/modulefiles ----------------------------
atp/1.3.0                libfast/1.0.9           tpsl/1.1.01
fftw/2.1.5.3             netcdf/4.1.2            trilinos/10.6.4.0
fftw/3.3.0.0(default)    ntk/1.3.0(default)      xt-lgdb/1.4
ga/4.3.5(default)        onesided/1.3.0(default) xt-libsci/11.0.03
hdf5/1.8.6               stat/1.1.3(default)     xt-mpt/5.3.4(default)

------------------------------ /opt/modulefiles -------------------------------
PrgEnv-cray/4.0.30(default)      gcc/4.5.3
PrgEnv-gnu/4.0.30(default)       gcc/4.6.1(default)
PrgEnv-pathscale/4.0.30(default) intel/12.0.5.220(default)
PrgEnv-pgi/4.0.30(default)       mrnet/3.0.0(default)
acml/4.4.0(default)              pathscale/4.0.11(default)
cce/7.4.4                        pathscale/4.0.9
cce/8.0.0.129                    petsc/3.1.09
cce/8.0.0.135(default)           petsc-complex/3.1.09
fftw/2.1.5.3                     pgi/11.9.0(default)
fftw/3.3.0.0(default)            xt-asyncpe/5.03(default)
gcc/4.4.4                        xt-totalview/8.9.2(default)

--------------------------- /apps/todi/modulefiles ----------------------------
intel/12.0.3
```

Which Software Versions Are Available?

```
ldr@todi1:~> module avail perftools

--------------------------- /opt/cray/modulefiles ---------------------------
perftools/5.2.3(default) perftools/5.3.0.8330


ldr@todi1:~> module avail cce

------------------------------ /opt/modulefiles ------------------------------
cce/7.4.4               cce/8.0.0.129           cce/8.0.0.135(default)
```

What Happens When I Load a Module?

```
ldr@todi1:~> module show perftools
-------------------------------------------------------------------
/opt/cray/modulefiles/perftools/5.2.3:

setenv           PERFTOOLS_VERSION 5.2.3
conflict         x2-craypat
conflict         craypat
conflict         xt-craypat
conflict         apprentice2
module           load rca
setenv           CHPL_CG_CPP_LINES 1
setenv           PDGCS_LLVM_DISABLE_FP_ELIM 1
setenv           PAT_REPORT_PRUNE_NAME
__cray_hwpc_,f_cray_hwpc_,cstart,__pat_,pat_region_,PAT_,OMP.slave_loop,slave_entry,_new_slave_entry,__libc_st
art_main,_start,__start,start_thread,__wrap_,UPC_ADIO_,_upc_,upc_,__caf_,__pgas_
module-whatis    Perftools - the Performance Tools module sets up environments for CrayPat, Apprentice2 and
PAPI
prepend-path     PATH /opt/cray/perftools/5.2.3/bin
prepend-path     MANPATH /opt/cray/perftools/5.2.3/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/5.2.3/cpatx
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/$PERFTOOLS_VERSION/cpatx
module           load papi/4.1
setenv           APP2_STATE 5.2.3
setenv           JH_HELPSET /opt/cray/perftools/5.2.3/help/app2help.jar
setenv           JH_VIEWER /opt/cray/perftools/5.2.3/help/jh2_0_05/demos/bin/hsviewer.jar
prepend-path     LD_LIBRARY_PATH /opt/cray/perftools/5.2.3/cpatx/lib
append-path      CLASSPATH /opt/cray/perftools/5.2.3/help/jh2_0_05/javahelp
append-path      PE_PRODUCT_LIST PERFTOOLS
append-path      PE_PRODUCT_LIST CRAYPAT
-------------------------------------------------------------------
```

What is xtpe-arch?

```
ldr@todi1:~> 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-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
setenv          PATHSCALE_PRE_COMPILE_OPTS  -march=barcelona
-------------------------------------------------------------------
```

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

I should build for the right compute-node architecture.

Oh yeah, let's link in the tuned math libraries for this architecture too.

Release Notes

```
users/ldr> module help cce/7.4.0

----------- Module Specific Help for 'cce/7.4.0' -----------------

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 7.4.0
================================

Release Date: June 16, 2011

  CCE 7.4.0
  =========
  Purpose:
  --------
    The Cray Compiling Environment 7.4 consists of the Cray Fortran compiler,
    Cray C compiler, Cray C++ compiler, and associated supporting libraries
    and utilities. CCE is used on Cray XE systems that run on the Cray Linux
    Environment (CLE) operating system, version 3.1 UP02 and later, and on
    Cray XT systems that run CLE 2.2 or CLE 3.1 UP02 and later.

  Productivity improvements in CCE 7.4
  ------------------------------------
    The Cray Compiling Environment release provides the following productivity
    enhancements:
     -  Performance enhancements for Cray XE and Cray XT systems
     -  Performance enhancements for PGAS applications on Cray XE systems
     -  New features as specified by the 2008 Fortran standard
     -  New extensions to UPC including privatizability functions, topology
        functions, and non-blocking versions of the upc_mem* family of functions
```

Release Notes (cont.)

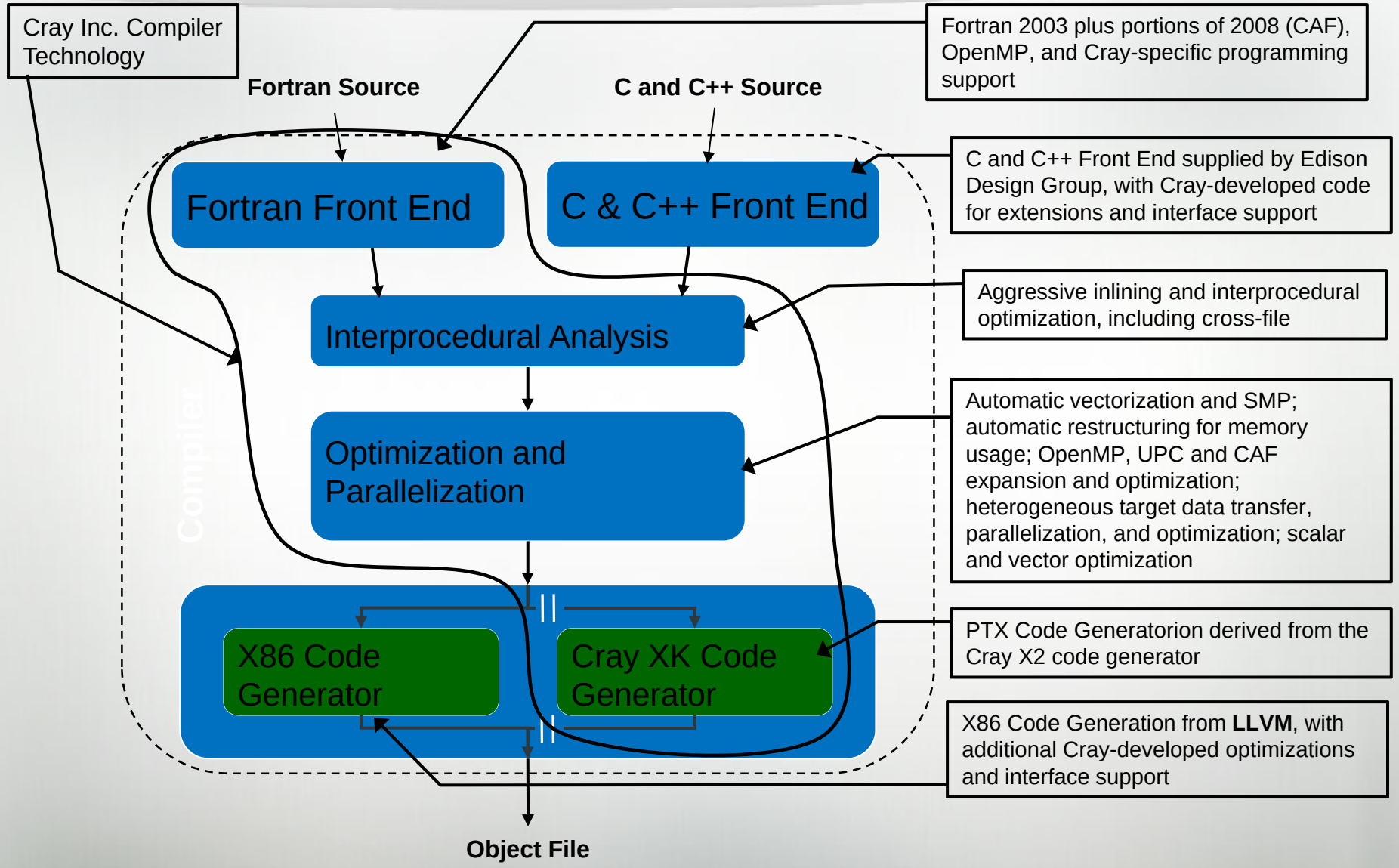
```
Feature information and overview:
  ---------------------------------
    CCE 7.4 contains
      performance enhancements
      Fortran, OpneMP and ISO C++ standards compliance improvements
      Atomic Memory Operations Intrinsics
      Cloning Directives
      Topology Functions
      Non-blocking Versions of the upc_mem* Family of Functions
      Privatizability Functions in UPC
      New PGAS Environment Variable
      New Command Line Options

    See the Cray Compiling Environment 7.4 Release Overview and Installation
    Guide (S-5212-74) for a more complete list of enhancements.

  Bugs fixed in the CCE 7.4.0 release:
  ------------------------------------
. . .
  Known Limitations:
  ------------------
. . .
  Dependencies:
  -------------
. . .
  Installation instructions:
  --------------------------
```

Release Notes (pre-release)

```
ldr@todi1:~> module help cce

----------- Module Specific Help for 'cce/8.0.0.135' --------------

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 8.0.0 Customer test package
======================================================
  Purpose:
  --------
    This CCE 8.0.0 package is intended for CCE 8.0.0 customer test.
    This CCE 8.0.0 provides Fortran, C, and C++ compilers for Cray XE and Cray XK6 systems.

    CCE 8.0 highlights:
      Support for Cray XK systems
      Support for Advanced Vector Extensions (AVX)
      128-bit (quad-precision) intrinsic
      Performance enhancements for PGAS
      New features as specified by the 2008 Fortran standard

    The Cray XK support includes a set of accelerator directives that are
    based on proposed OpenMP accelerator directives. This enables the
    development of applications for the Cray XK6 using a directives based
    model.  Please see the intro_openmp_acc man page for more information
    about these directives.

    Additional details can be found in the:
      Cray Compiling Environment 8.0 Release Overview  S-5212-80

    This release also includes many bug fixes.
```

The Cray Compilation Environment

CCE Technology Sources



CCE Main Features

- Compliance with ANSI/ISO FORTRAN 2003
  - Fortran 2008 (full compliance targeted for 2012)
    - **Fortran 2008 coarrays**
    - Submodules
    - Block construct
    - Contiguous Attribute
    - ALLOCATE enhancements (MOLD =, shape from SOURCE/MOLD)
    - intrinsic assignment for polymorphic variables
    - Most of the new intrinsic functions
    - ISO_Fortran_Env module enhancements

- Compliance with ANSI/ISO C99 and ANSI/ISO C++ 2003 (except the export keyword for templates)
  - Support for Kernighan & Ritchie C
  - C/C++ enhancements/changes
    - updated to GCC version 4.4.4 compatibility
    - C++ supports the ISO 1998 Standard Template Library (STL) headers
    - Upgraded the C and C++ front end to EDG Version 4.1
      - **With this update CCE can better handle modern C++ applications**
      - Periodic synchronization with the latest sources and bug fixes
      - Better support for non-standard GNU language extensions
      - The new EDG C and C++ front end more strictly enforces the standards
  - UPC 1.2 support

CCE Main Features (cont.)

- AMD Interlagos support, including AVX, FMA, and XOP instructions
- X86/NVIDIA compiler and library development (ongoing “beta” release)
- Support for MPI 2.2
- Full OpenMP 3.0 support
  - Automatic multithreading integrated with OpenMP
  - OpenMP 3.1 under development
    - Atomic construct extensions
    - *taskyield* construct
    - *firstprivate* clause accepts intend(in) and constant objects
- 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 7.4.4 was released on October 20, 2011
  - The full release overview can be found at: http://docs.cray.com/books/S-5212-74/
  - CCE 8.0 targeted to be released on December 15, 2011

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**
    - 3 - **Constant actual argument inlining and tiny routine inlining**. Default level 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**

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
- Use –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 for Resid (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.
```

OpenMP

- OpenMP is **ON** by default
  - Optimizations controlled by **–Othread#**
- 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 **–Othread0** or **–xomp** or **–hnoomp**

Why CCE Does Not Support Inline Assembly

- It would be very expensive to implement
- Almost impossible to test in any comprehensive fashion
- Non-portable across architectures
- Needs to be rewritten as new hardware features become available (like 256-bit vectors and FMAs)
- There is no standard; essentially what gcc or Intel decides to support
- We want to spend our development resources on providing the best possible automatic vectorization, rather than forcing the developer to program in assembly language
- That said, we will consider individual requests for more direct access to instructions – but typically through intrinsic functions, rather than inline assembly

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

Impact of Fused Multiply-Add (FMA) on Application Results

- One rounding for the FMA as a whole, rather than two (one for multiply and one for addition)
- That sounds like a minor difference, but these differences can accumulate
- For our internal testing, most of the differences we manually approved by examining them and deciding the FMA-based results were within an acceptable range
- Actual applications – at least some of them – appear to be less forgiving
- There is no hardware way to obtain the exact same result between FMAs and individual multiplications and additions
  - … but the performance difference means we really do need to use them
- Some level of FMA control is provided by CCE –hfp options
  - **-hfp0:** No FMA generation (but also disables a lot of other stuff)
  - **-hfp1:** Generate FMAs, but not across user parenthesis
  - **-hfp2,3:** Aggressive FMA generation

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++

Using the Compiler Driver Commands

- You use compiler driver commands to launch all Cray XE compilers
- The syntax for the compiler driver is:
  - cc | CC | ftn [Cray_options | PGI_options | GNU_options] files [-lhugetlbfs]
- For example, to use any Fortran compiler (CCE, PGI, GNU) to compile prog1.f90
  - Use this command:
    - % ftn prog1.f90
- The compiler drivers are setup by the PrgEnv-??? Module

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 '.'

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