Cray XE6 architecture

Cray XE6 Performance Workshop
11-12 July

Recipe for a good MPP

- **Select Best Microprocessor**
  - Function of time
- **Surround it with a balanced or “bandwidth rich” environment**
  - Interconnection network
  - Local memory
- **“Scale” the System**
  - Eliminate Operating System Interference (OS Jitter)
  - Design in Reliability and Resiliency
  - Provide Scaleable System Management
  - Provide Scaleable I/O
  - Provide Scaleable Programming and Performance Tools
  - System Service Life (provide an upgrade path)

The Cray XE6 node

We are concentrating on the Cray XE6 installed here in Edinburgh, which is called HECToR

There are other XE6 models using different processors and different interconnects topology which we don't cover in this workshop

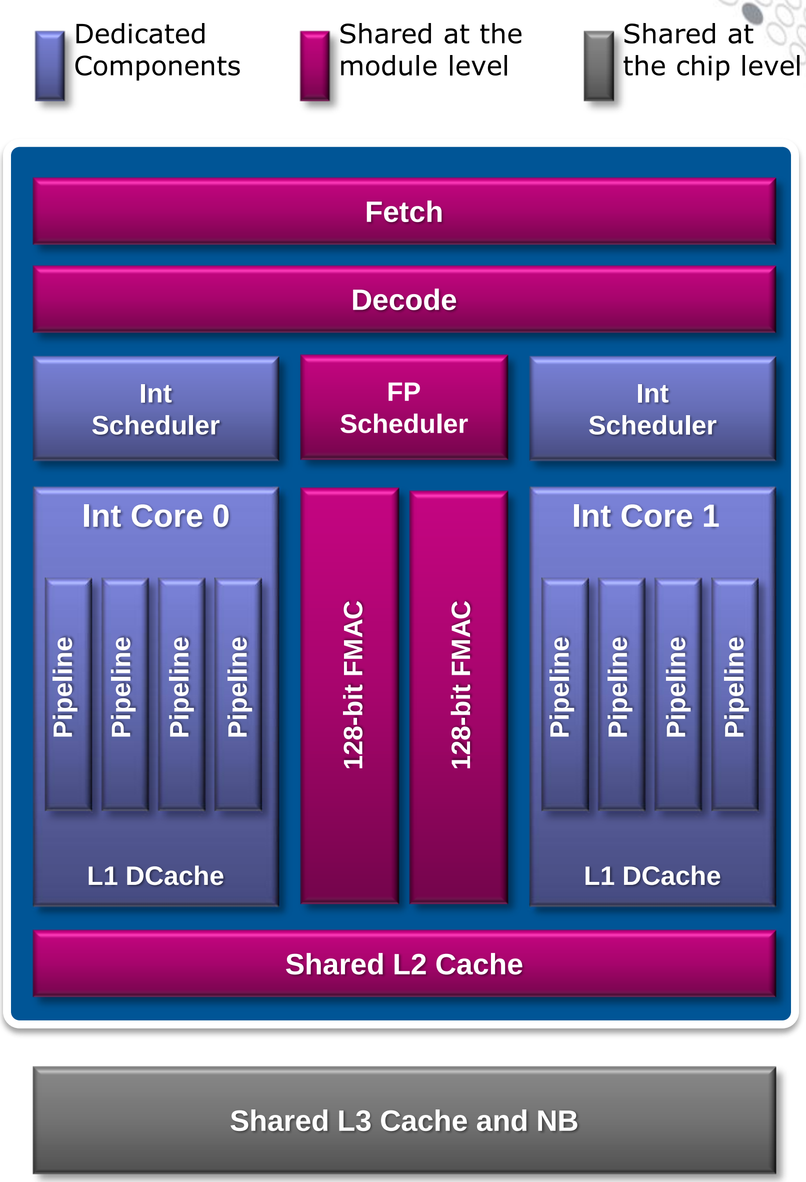
We start by introducing the node parts (processors used, interconnect, …) and shows how they are packaged

Processor

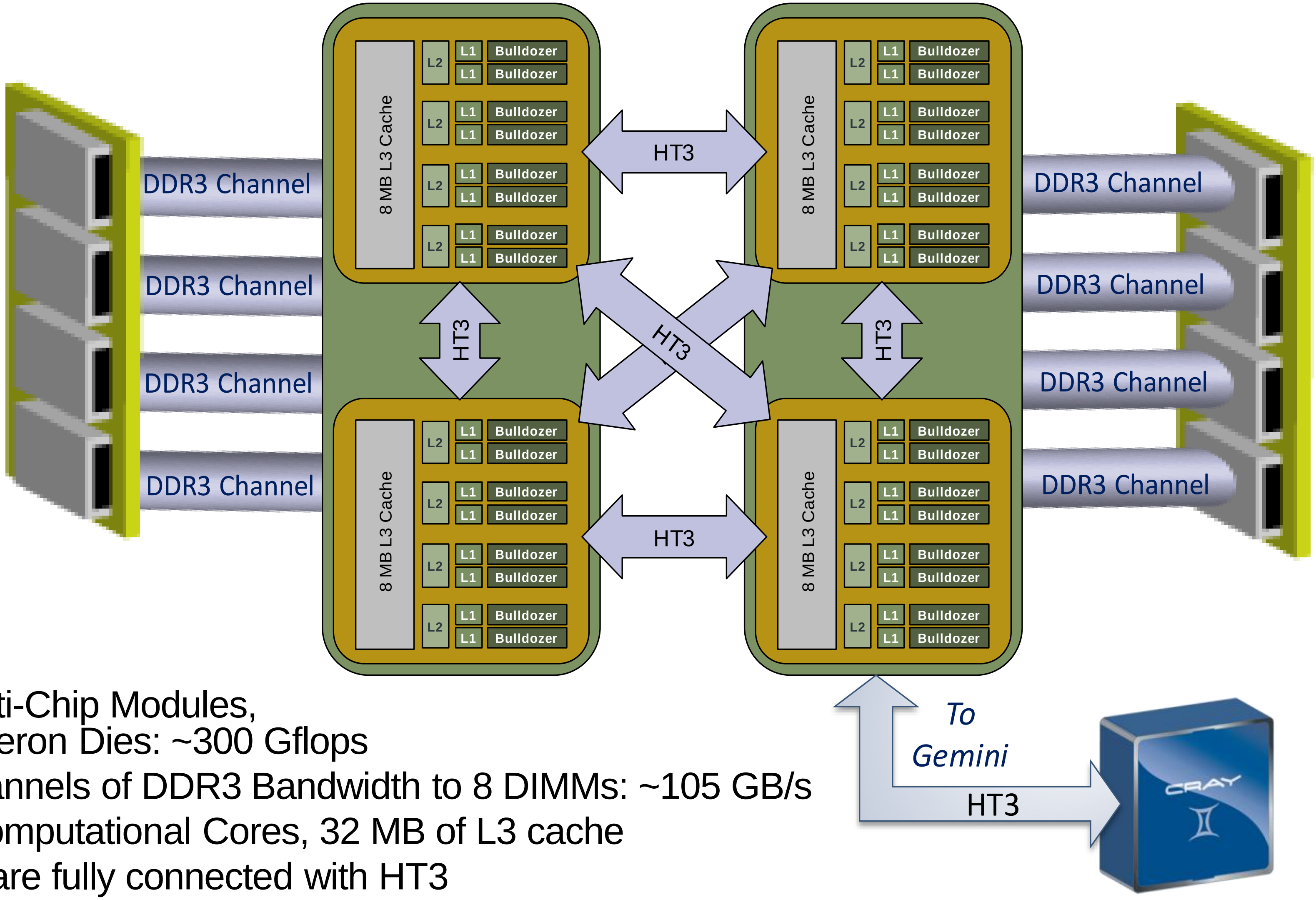
The new Opteron 6200 Series (Interlagos)
HECToR uses the Model 6276 (2.3 GHz)

Interlagos Processor Architecture

- Interlagos is composed of a number of Bulldozer core "modules"
- A core module has shared and dedicated components
- There are two independent integer cores and a *shared*, 256-bit FP resource
- A single Integer Core can make use of the entire FP resource with 256-bit AVX instructions
- This architecture is very flexible, and can be applied effectively to a variety of workloads and problems
- **DL1 is 16 KB, L2 is 2 MB and L3 is 8MB**



Cray XE6 Node Details – 32-core Interlagos



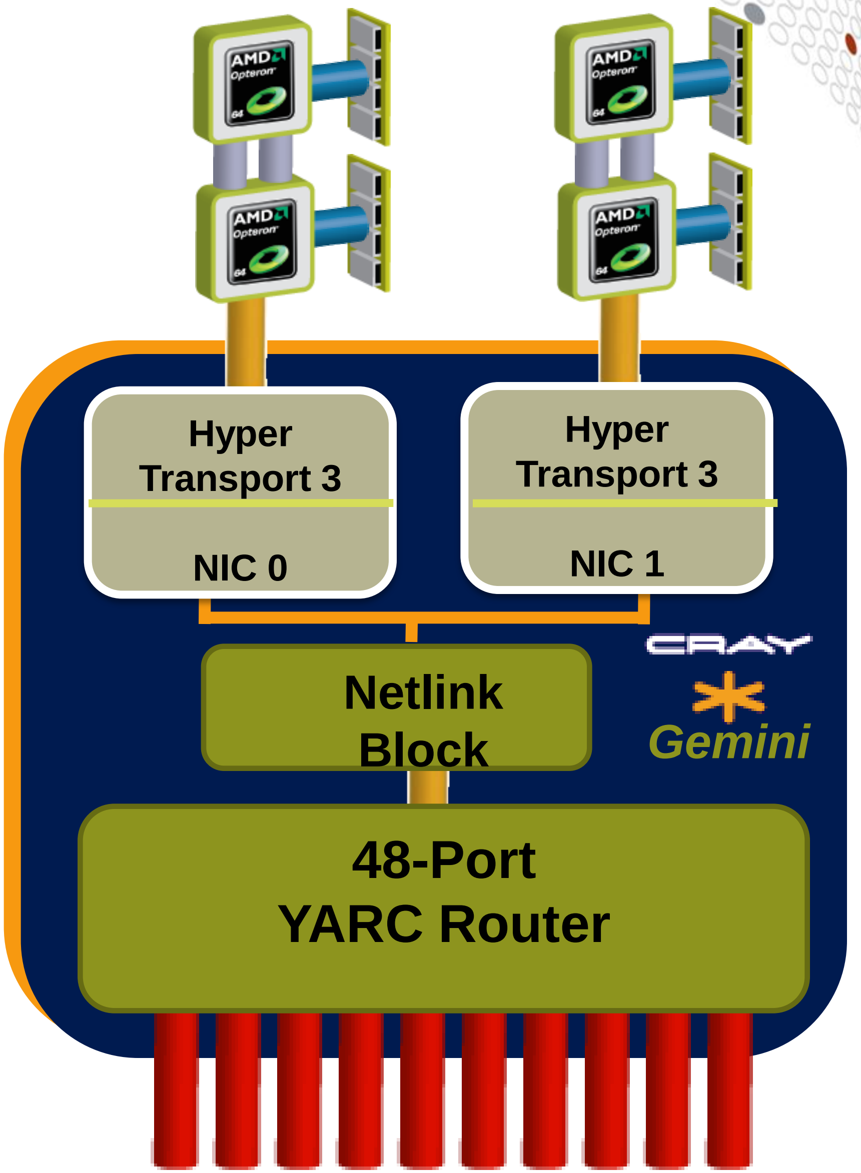
- 2 Multi-Chip Modules,
  4 Opteron Dies: ~300 Gflops
- 8 Channels of DDR3 Bandwidth to 8 DIMMs: ~105 GB/s
- 32 Computational Cores, 32 MB of L3 cache
- Dies are fully connected with HT3

Gemini

The Cray interconnect

Cray Gemini ASIC (application-specific integrated circuit)

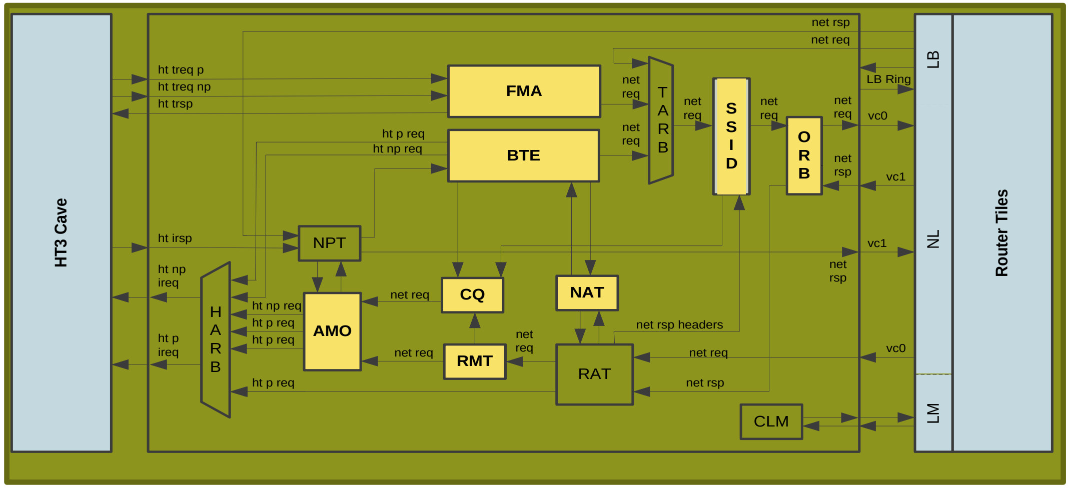
- **Supports 2 Nodes per ASIC**
- **3D Torus network**
  - XT5/XT6 systems field upgradable
- **Scales to over 100,000 network endpoints**
  - Link Level Reliability and Adaptive Routing
  - Advanced Resiliency Features
- **Advanced features**
  - MPI – millions of messages / second
  - One-sided MPI
  - UPC, Coarray FORTRAN, Shmem, Global Arrays
  - Atomic memory operations



Gemini NIC Design

- **Fast memory access (FMA)**
  - Mechanism for most MPI transfers, involves processor
  - Supports tens of millions of MPI requests per second
- **Block transfer engine (BTE)**
  - Supports asynchronous block transfers between local and remote memory, in either direction
  - For large MPI transfers that happen in the background

- Hardware pipeline maximizes issue rate
- HyperTransport 3 host interface
- Hardware translation of user ranks and addresses
- AMO cache
- Network bandwidth dynamically shared between NICs



Gemini Software

- **Cray MPI uses MPICH2 distribution from Argonne**
  - CH3 device Nemesis: multi-method device with a highly optimized shared memory sub-method
- **MPI device for Gemini based on**
  - User level Gemini Network Interface (uGNI)
  - Distributed Memory Applications (DMAPP) library
- **FMA (Fast Memory Access)**
  - In general used for small transfers
  - FMA transfers are lower latency
- **BTE (Block Transfer Engine)**
  - BTE transfers take longer to start but can transfer large amount of data without CPU involvement

Gemini MPI Features

- **FMA provides low-overhead OS-bypass**
  - Lightweight issue of small transfers
- **DMA offload engine**
  - Allows large transfers to proceed asynchronously of the application
- **Designed to minimize memory usage on large jobs**
  - Typically 20MB/process including 4MB buffer for unexpected messages
- **Adaptive routing:**
  - Reduces network contention
  - Automatically routes around link failures
- **AMOs provide a fast synchronization method for collectives**
  - AMO=Atomic Memory Operations

Gemini PGAS Features

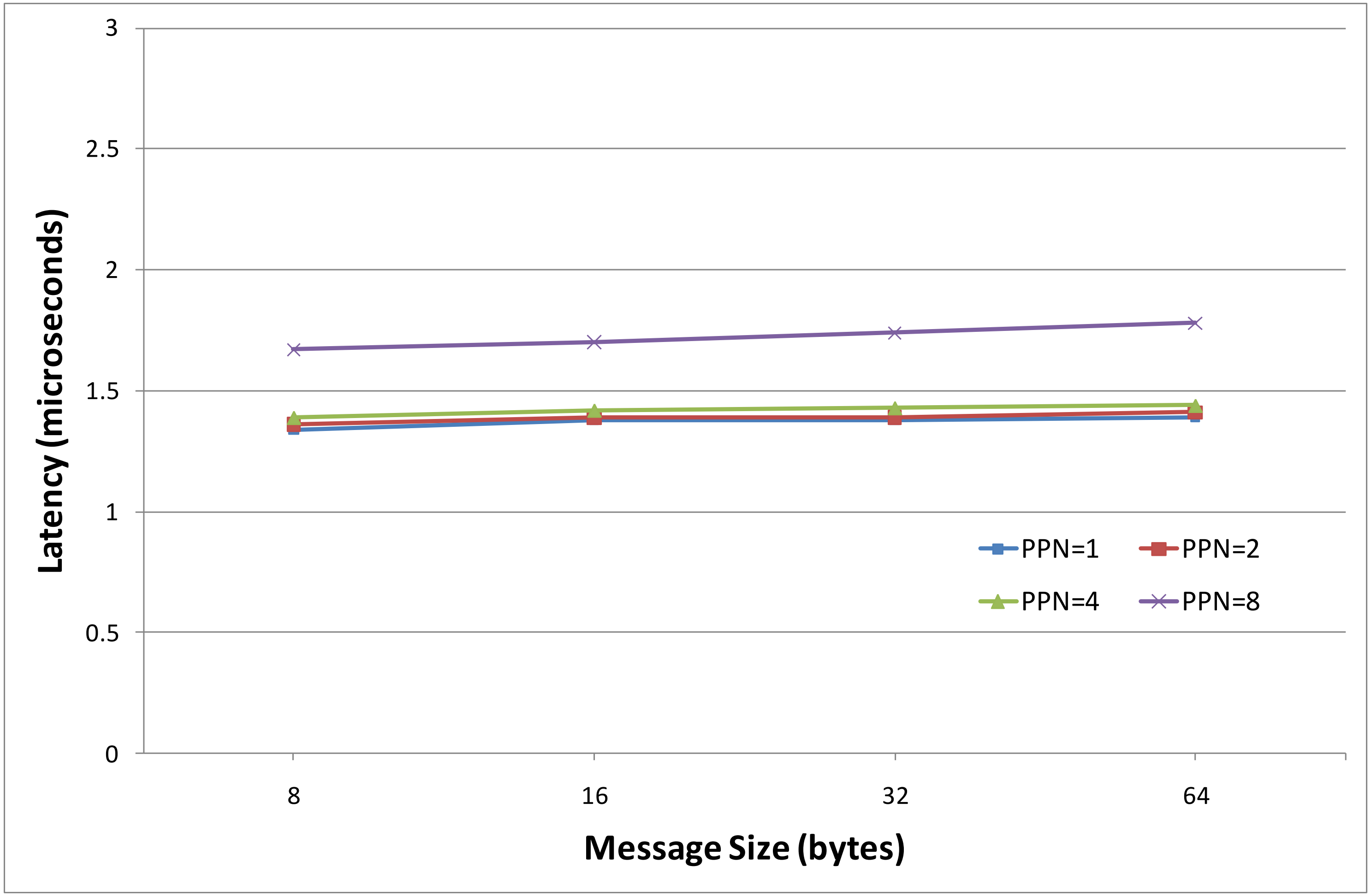
- **PGAS= Partitioned Global Address Space**
- **Globally addressable memory provides efficient support for**
  - UPC, Co-array FORTRAN, SHMEM
- **Pipelined global loads and stores**
  - Allows for fast execution of irregular communication patterns
- **Atomic memory operations**
  - Provides fast synchronization method for one-sided communication
- **Cray DMAPP application interface**
  - Cray Programming Environment targets this directly
  - Available for 3rd party tools

Gemini Performance Highlights

- **MPI latency of 1.4 µs**
  - 3X improvement on Seastar
- **MPI message rate of 9M/sec**
  - 20X improvement on Seastar
- **Injection bandwidths in excess of 6 GB/sec**
  - 3X improvement on Seastar
- **Cray SHMEM put rate of 25M/sec**
- **Scattered/indexed put rates of 60-90M/sec**

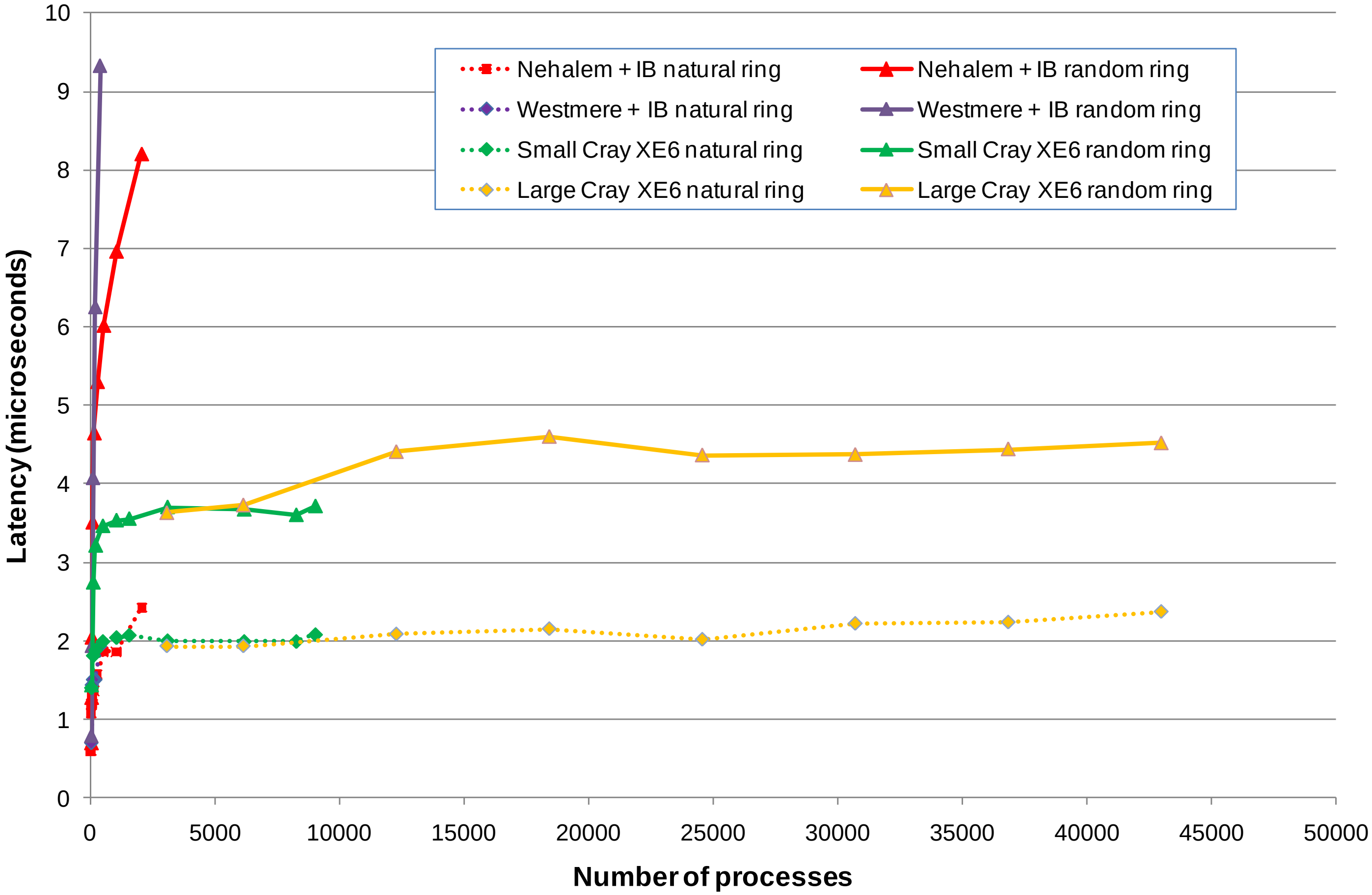
Gemini MPI Latency

- Small message latencies of 1.4 µs (best case)



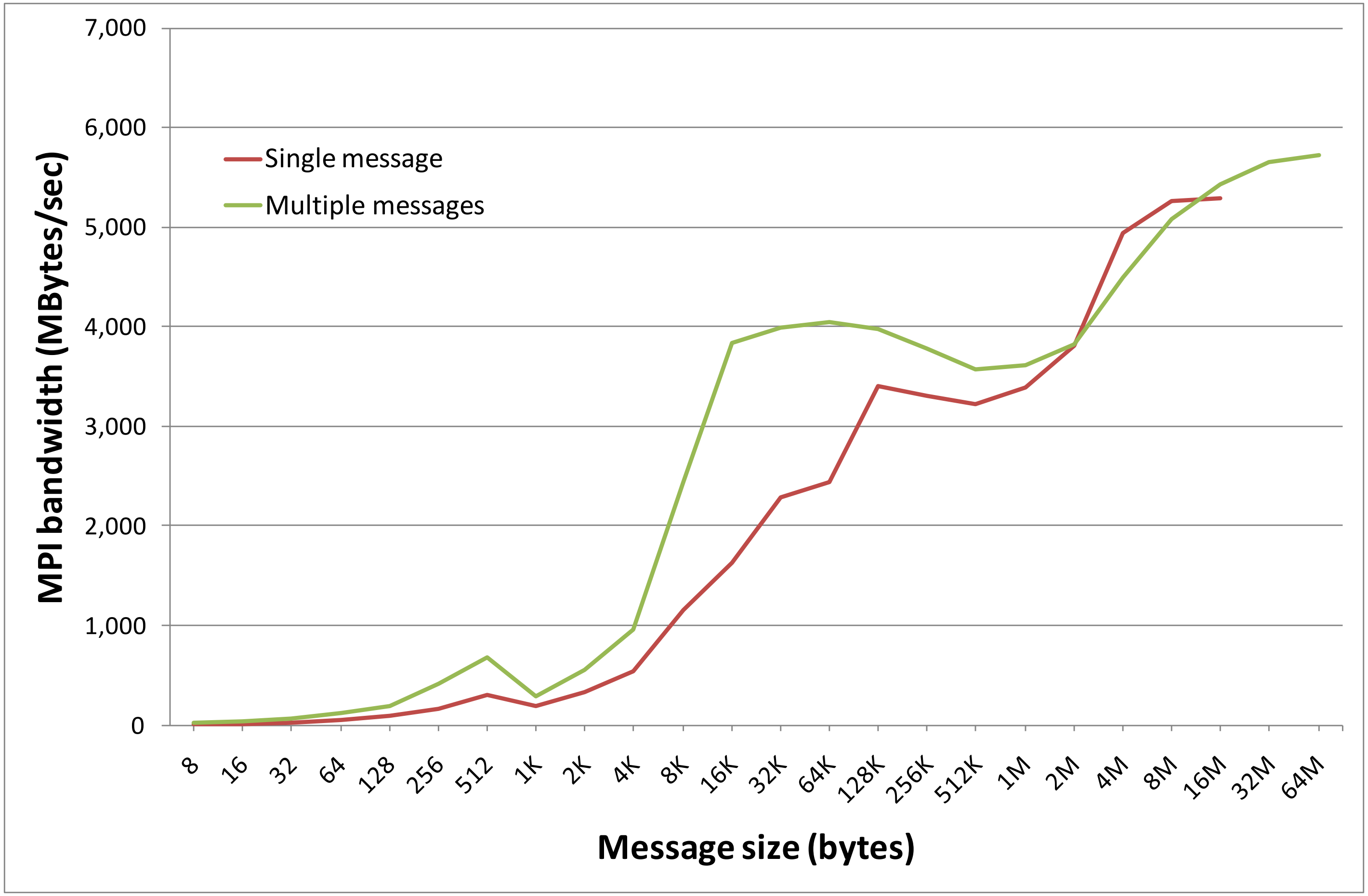
MPI Latency at Scale

- Low latencies are maintained across the whole system with cores sending non-local messages (HPCC natural+random ring)



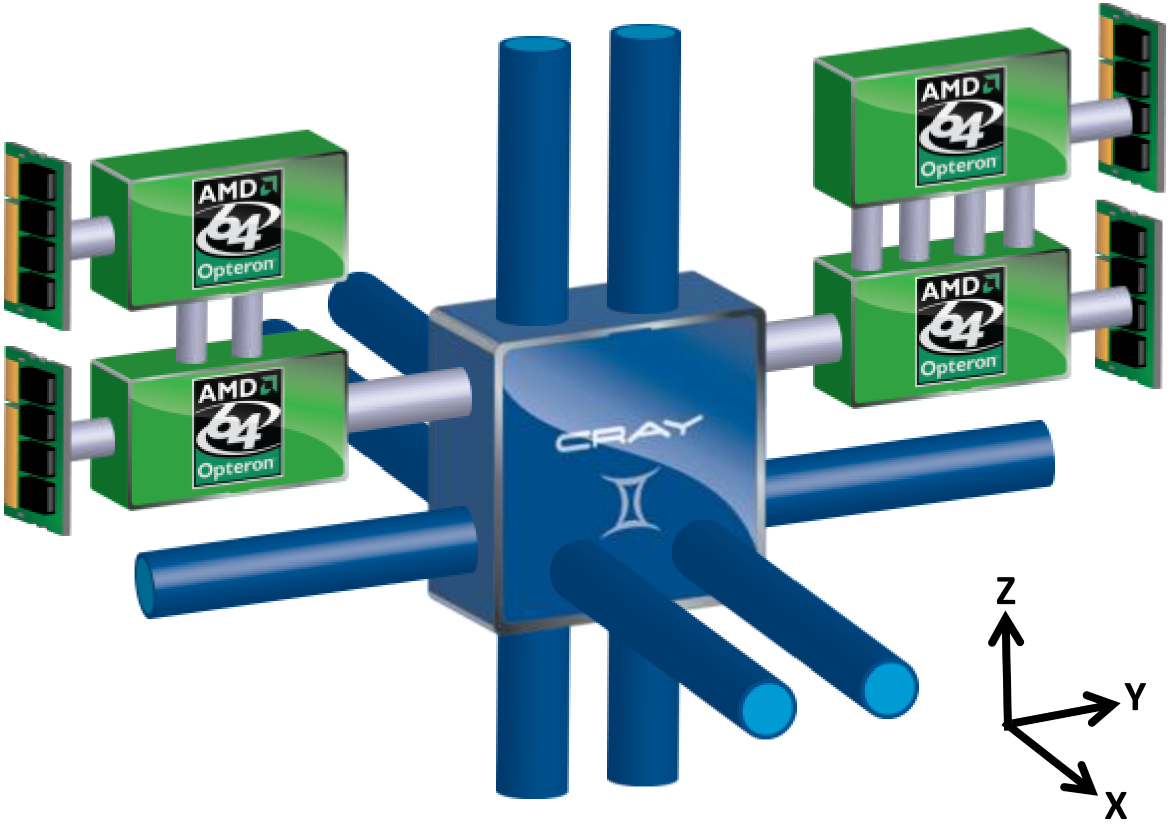
MPI Bandwidth

- Gemini MPI bandwidth exceeds 5 GB/sec – nearly twice that of QDR Infiniband (not limited by Gen2 PCI-Express)



Cray XE6 Compute Node

- Built around the Gemini Interconnect
- Each Gemini ASIC provides 2 NICs enabling it to connect 2 dual-socket nodes



The Cray Programming Environment Overview

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®
- GNU

Tools

Environment setup

- Modules

Debuggers

- TOTALVIEW TECHNOLOGIES
- DDT
- gdb

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

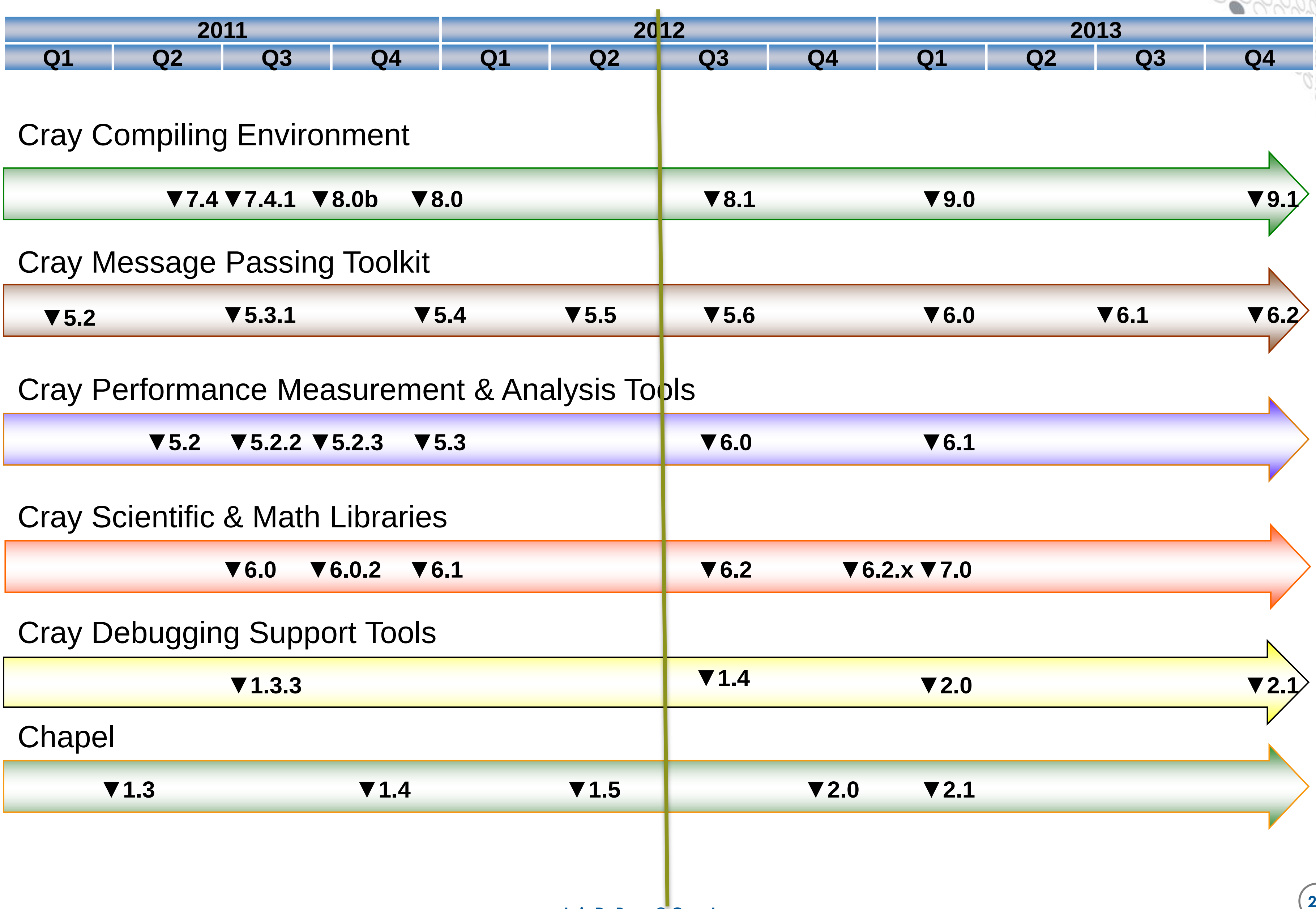
#: Under development

Licensed ISV SW

3rd party packaging

Cray added value to 3rd party

Cray Programming Environment Releases



2011 2012 2013

Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4

Cray Compiling Environment

▼7.4 ▼7.4.1 ▼8.0b ▼8.0 ▼8.1 ▼9.0 ▼9.1

Cray Message Passing Toolkit

▼5.2 ▼5.3.1 ▼5.4 ▼5.5 ▼5.6 ▼6.0 ▼6.1 ▼6.2

Cray Performance Measurement & Analysis Tools

▼5.2 ▼5.2.2 ▼5.2.3 ▼5.3 ▼6.0 ▼6.1

Cray Scientific & Math Libraries

▼6.0 ▼6.0.2 ▼6.1 ▼6.2 ▼6.2.x ▼7.0

Cray Debugging Support Tools

▼1.3.3 ▼1.4 ▼2.0 ▼2.1

Chapel

▼1.3 ▼1.4 ▼1.5 ▼2.0 ▼2.1

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 XE implementation close to the T3E model

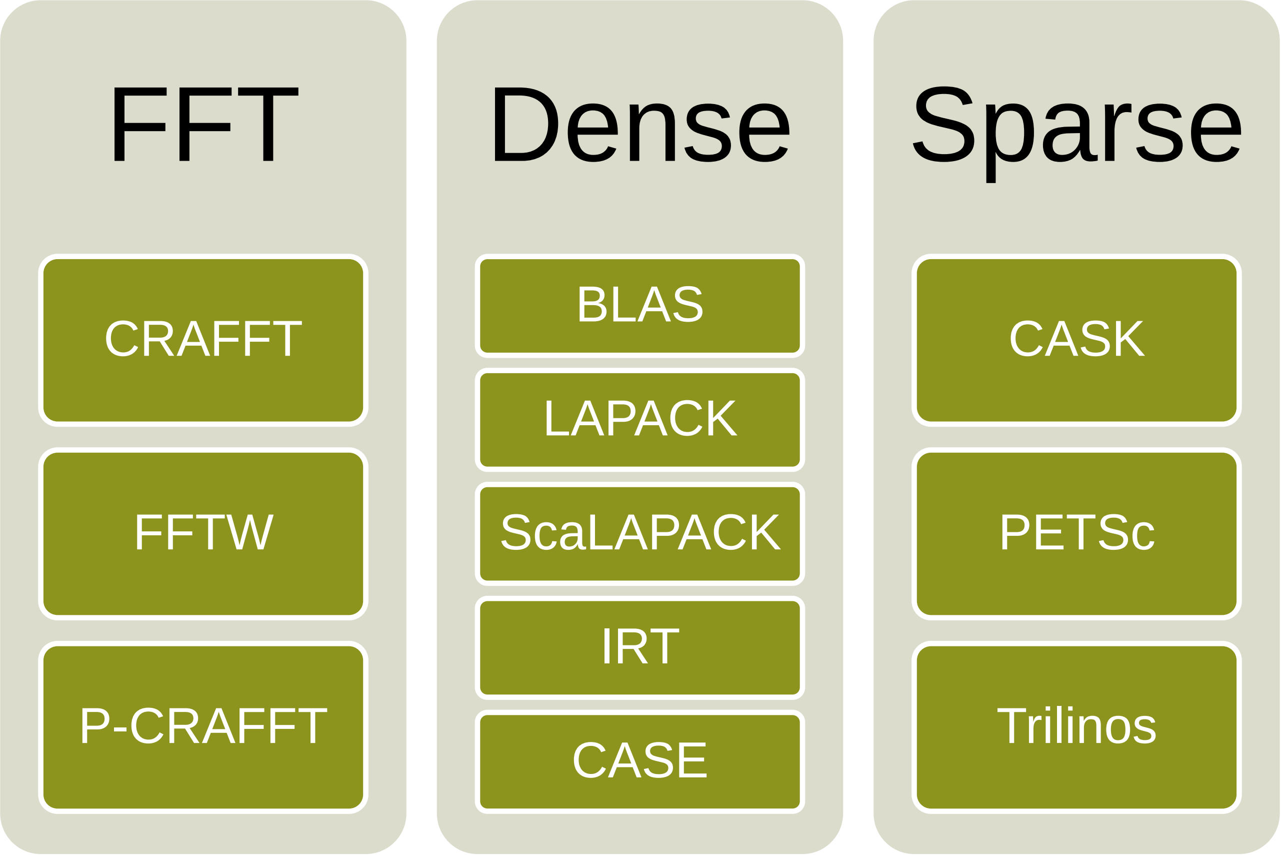
Cray Performance Analysis Tools

- **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
      - Debugging optimized applications
      - Added to Allinea's DDT 2.6 (June 2010)
    - Comparative debugging
      - A data-centric paradigm instead of the traditional control-centric paradigm
      - Collaboration with Monash University and University of Wisconsin for scalability
  - Support for traditional debugging mechanism
    - TotalView, DDT, and gdb

Cray Scientific Libraries



FFT

CRAFFT

FFTW

P-CRAFFT

Dense

BLAS

LAPACK

ScaLAPACK

IRT

CASE

Sparse

CASK

PETSc

Trilinos

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

Environment Setup

Environment Setup

- **The Cray XE system uses 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
  - You can use the default version of an application, or you can choose another version by using Modules system commands

The module tool on the Cray XE

- **How can we get appropriate Compiler, Tools, and Libraries?**
  - The modules tool is used to handle different versions of packages
    - e.g.: module load compiler_v1
    - e.g.: module swap compiler_v1 compiler_v2
    - e.g.: module load perftools

- **Taking care of changing of PATH, MANPATH, LM_LICENSE_FILE,.... environment**
  - 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**

Useful module commands

- **Which modules are loaded?**
  - module list
- **Load software**
  - module load perftools
- **Change programming environment**
  - module swap PrgEnv-cray PrgEnv-gnu
- **Change software version**
  - module swap cce/8.0.2 cce/7.4.4

module list

```
eslogin002:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.6                          13) xe-sysroot/4.0.36
  2) xtpe-network-gemini                      14) rca/1.0.0-2.0400.30002.5.75.gem
  3) xtpe-interlagos                          15) xt-asyncpe/5.07
  4) cce/8.0.2                                16) atp/1.4.2
  5) acml/4.4.0                               17) PrgEnv-cray/4.0.36
  6) xt-libsci/11.0.05                        18) xt-mpich2/5.4.3
  7) udreg/2.3.1-1.0400.3911.5.13.gem         19) eswrap/1.0.9
  8) ugni/2.3-1.0400.4127.5.20.gem            20) torque/2.5.9
  9) pmi/3.0.0-1.0000.8661.28.2807.gem        21) moab/6.1.5.s1992
 10) dmapp/3.2.1-1.0400.3965.10.63.gem        22) system/ws_tools
 11) gni-headers/2.1-1.0400.4156.6.1.gem      23) system/hlrs-defaults
 12) xpmem/0.1-2.0400.30792.5.6.gem
```

PrgEnv-cray is the default

What is xtpe-interlagos?

```
@eslogin002:~> 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
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.

Which SW Products and Versions Are Available

- **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

Which Software Versions Are Available?

```
hpcnicho@eslogin002:~> module avail perftools

--------------------------- /opt/cray/modulefiles --------------------------
perftools/5.2.0          perftools/5.2.3          perftools/5.3.0(default)




hpcnicho@eslogin002:~> module avail cce

---------------------------- /opt/modulefiles --------------------------------

cce/7.3.3          cce/7.4.2          cce/8.0.0          cce/8.0.0.137
cce/8.0.2(default) cce/7.3.4          cce/7.4.4          cce/8.0.0.129
cce/8.0.1
```

What Happens When I Load a Module?

```
hpcnicho@eslogin002:~> module show perftool
-------------------------------------------------------------------
/opt/cray/modulefiles/perftools/5.3.0:

setenv           PERFTOOLS_VERSION 5.3.0
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$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_
module-whatis    Perftools - the Performance Tools module sets up environments for CrayPat, Apprentice2 and
PAPI
prepend-path     PATH /opt/cray/perftools/5.3.0/bin
prepend-path     MANPATH /opt/cray/perftools/5.3.0/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/5.3.0
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
module           load papi/4.2.0
setenv           APP2_STATE 5.3.0
setenv           JH_HELPSET /opt/cray/perftools/5.3.0/help/app2help.jar
setenv           JH_VIEWER /opt/cray/perftools/5.3.0/help/jh2_0_05/demos/bin/hsviewer.jar
prepend-path     CRAY_LD_LIBRARY_PATH /opt/cray/perftools/5.3.0/lib
append-path      CLASSPATH /opt/cray/perftools/5.3.0/help/jh2_0_05/javahelp
append-path      PE_PRODUCT_LIST PERFTOOLS
append-path      PE_PRODUCT_LIST CRAYPAT
-------------------------------------------------------------------
```

Release Notes

```
hpcnicho@eslogin002:~> module help cce/8.0.2

----------- Module Specific Help for 'cce/8.0.2' -------------------

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 8.0.2 (CCE 8.0.2)
============================================
  Purpose:
  --------
    The CCE 8.0.2 update provides bugfixes to the CCE 8.0.1 release for Cray XE
    systems.

    Bugs fixed in 8.0.2 are:
      779483  Runtime error with Cray Fortran compiler cce/7.4.4
      780053  Illegal folding of optional argument test into a merge
      780346  Internal compiler error with crayftn when enabling full debugging
      779573  Fortran function pointer issue

    Note:
    -----
      Support for CCE on Cray XT systems will continue to be provided with
      updates to the CCE 7.4 release. The CCE 8.0 release branch is
      supported on the Cray XE and XK systems only.

    Dependencies:
    -------------
      The CCE 8.0.2 release is supported on Cray XE systems that run on the Cray
      Linux Environment (CLE) operating system, version 3.1 and later and on the
      Cray XK systems that run the Cray Linux Environment 4.0 UP01 and later.
```

Release Notes (cont.)

```
CCE 8.0.2 requires that gcc/4.4.4 be installed. GCC 4.4.4 does not need to
     be a default GNU environment.

     Cray Performance Measurement and Analysis Tools dependency:
       - cce/8.0.0 or later compiles using -h profile_generate require
         perftools/5.3.0 in order to provide loop work estimates.
       - perftools/5.3.0 is required to support the PGAS (UPC, CAF) runtime
         library changes made in CCE 8.0

     The Cray Compiling Environment 8.0.2 update requires the following supporting
     asynchronous software products:
       Cray Compiler Drivers (xt-asyncpe) 5.04 or later
       GNU GCC 4.4.4 must be installed, but is not required to be the default GCC
       PMI 2.1.4 or later
       Cray Scientific Libraries (LibSci) 11.0.00 or later

    The Cray Compiling Environment 8.0.2 update 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
```

Using the Compiler Driver Commands

- **You use compiler driver commands to launch all Cray XE compilers**

- **The syntax for the compiler driver is:**

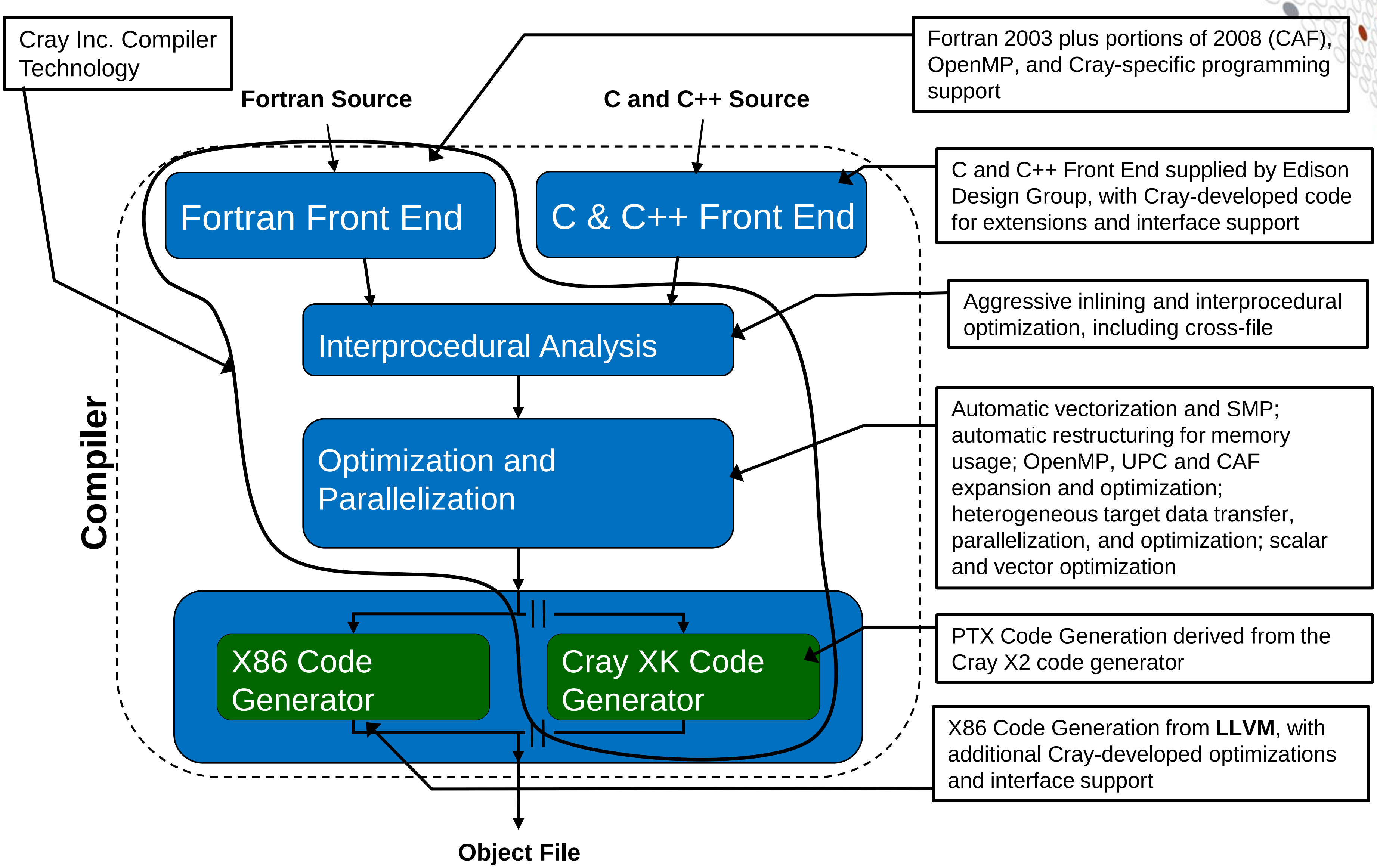
  - cc | CC | ftn [Cray_options | PGI_options | GNU_options] files [-lhugetlbfs]

- **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**

The Cray Compilation Environment

CCE Technology Sources



Fortran 2003 plus portions of 2008 (CAF), OpenMP, and Cray-specific programming support

C and C++ Front End supplied by Edison Design Group, with Cray-developed code for extensions and interface support

Aggressive inlining and interprocedural optimization, including cross-file

Automatic vectorization and SMP; automatic restructuring for memory usage; OpenMP, UPC and CAF expansion and optimization; heterogeneous target data transfer, parallelization, and optimization; scalar and vector optimization

PTX Code Generation derived from the Cray X2 code generator

X86 Code Generation from **LLVM**, with additional Cray-developed optimizations and interface support

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
    - 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 8.0 was released on December, 2011**
  - The full release overview can be found at: http://docs.cray.com/books/S-5212-74/

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 –Oipan (ftn) –hipan (cc/CC) where n=0..4, provides a set of choices for inlining behavior
    - 0 - All inlining and cloning are disabled. 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. A call nest can be a leaf routine. The expansion of the call nest must yield straight-line code (code containing no external calls) for any expansion to occur. The call site is said to "flatten" when there are no calls present in the expanded code. 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. This includes levels 1 and 2, plus any call site that contains a constant actual argument. Additionally, any call nest (regardless of location) that is below some small compiler-determined threshold will be inlined provided that call nest completely flattens. 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.

Inlining with CCE (cont)

- By default, all inlining candidates come from the current source file
- 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
  - “ftn –Oipafrom=b.f a.f” tells the compiler to look for inlining candidates within b.f when compiling a.f
  - “cc –hipafrom=./dir src.c” tells the compiler to look for inlining candidates in all the valid source files that exist in the directory ./dir when compiling src.c
- Cross language inlining is not supported.

Whole-Program Compilation

- **The Program Library (PL) feature allows the user to specify a repository of compiler information for an application build**
  - This repository provides the framework for future productivity features such as
    - Whole program static error detection
    - Incremental recompilation
    - Provide support for the future Cray interactive whole program performance analysis and tuning assistant Reveal

- **Two command line options control the Program Library functionality**
  - -h pl = <PL_path> specifies the repository
    - ftn –hpl=./PL.1 tells the compiler to either update the Program Library "./PL.1" if it exists, or create it if it does not exist.
    - <PL_path> should specify a single location to be used for entire application build. If a makefile changes directories during a build, an absolute path might be necessary.
  - -h wp enables whole-program mode

Whole-Program Compilation (cont)

- **Whole-program mode (-hwp) requires a program library (-hpl =) and both options must be specified on all compilation command lines as well as on the link line.**
  - The compiler frontend is invoked for the compilation (-c) command lines
  - The compiler backend (inliner, optimizer, code generator) is invoked for all source files when the link line is specified.
  - While –hwp might have a negative affect on overall compile time due to increased inlining, it is most usually a compile time shift, where –c compilations become quite fast and the time spent on the link step increases.
  - Setting the environment variable "NPROC" to a number greater than 1 instructs the compiler to invoke NPROC backend processes concurrently. The backend invocations are independent of each other and setting NPROC to a level that is appropriate for the host build machine can improve compile time.

- **Whole-program mode (-hwp) allows the inliner to see all inline candidates in the application.**
  - This option makes cross file inlining automatic
    - Removes the need for –h ipafrom =
    - Inlining heuristics are still controlled by –h/-O ipan

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)

Cray compiler flags

- **Overall Options**
  - -ra creates a listing file with optimization info
  - -rm produces a source listing with loopmark information
- **Preprocessor Options**
  - -eZ runs the preprocessor on Fortran files
  - -F enables macro expansion throughout the source file
- **Optimisation Options**
  - -O2 optimal flags [enabled by default]
  - -O3 aggressive optimization
  - -O ipa<n> inlining, n=0-5

Cray compiler flags

- **Language Options**

-f free	process Fortran source using freeform
-s real64	treat REAL variables as 64-bit
-s integer64	treat INTEGER variables as 64-bit

- **Parallelization Options**

-O omp	Recognize OpenMP directives [default]
-O thread<n>	n=0-3, aggressive parallelization, default n=2

=> man crayftn

http://docs.cray.com/cgi-bin/craydoc.cgi?mode=View;id=S-3901-71;idx=books_search;this_sort=;q=3901;type=books;title=Cray%20Fortran%20Reference%20Manual

OpenMP

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

- **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**

Performing Endian Conversion on Cray XE Series Systems

- Cray Compiling Environment (CCE)
  - -hbyteswapio: forces byte-swapping of all input and output files for direct and sequential unformatted I/O

Cray programming environment: assign

- **Assigns options for library file open processing**
- **assign [assign options] assign_object**

- **Interesting assign options**
  - -R removes all assign options for assign_object
  - -N <numcon> specifies foreign numeric conversion
  - swap_endian the endianess of data is swapped during unformatted input and output.
- **assign object used to specify the object of assign options**
  - f:<filename> applies to filename
  - u:<unit> applies to Fortran unit number
  - g:su applies to all Fortran sequential unform. files

How to handle byte-swapped files with CCE

- **Explicit usage of assign**
  - Can control which files are byte-swapped

```
export FILENV=.assign
assign -R
assign -N swap_endian f:aof
aprun a.out
```

- **Link the application with –hbyteswapio**
  - This is equivalent to set

```
assign -N swap_endian g:su ←all sequential unformatted
assign -N swap_endian g:du <-all direct unformatted
```

- **„man assign“ (when PrgEnv-cray loaded)**

CCE Directives

- **Cray compiler supports a full and growing set of directives and pragmas**

  - !dir$ concurrent
  - !dir$ ivdep
  - !dir$ interchange
  - !dir$ unroll
  - !dir$ loop_info [max_trips] [cache_na] ... Many more
  - !dir$ blockable

  - man directives
  - man loop_info

Loopmark/Compiler Feedback

- ftn –rm …   or   cc –hlist=m …
- Compiler can generate an **‘.lst’**with annotated listing of your source code with letter indicating important optimizations

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 for Resid

- 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.
```

Compiler man Pages

- **The cc(1), CC(1), and ftn(1) man pages contain information about the compiler driver commands**
- **The pgcc(1), pgCC(1), and pgf95(1) man pages contain descriptions of the PGI compiler command options**
- **The craycc(1), crayCC(1), and crayftn(1) man pages contain descriptions of the Cray 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 and PGI compiler flags

Feature	PGI	Cray
Listing	-Mlist	-ra
Diagnostic	-Minfo -Mneginfo	(produced by -ra)
Free format	-Mfree	-f free
Preprocessing	-Mpreprocess	-eZ -F
Suggested Optimization	-fast	(default)
Aggressive Optimization	-Mipa=fast,inline	-O3, fp3
Variables size	-r8 –i8	-s real64 –s integer64
Byte swap	-byteswapio	-h byteswapio
OpenMP recognition	-mp=nonuma	(default)
Automatic parallelization	-Mconcur	-h autothread

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

Other programming environments

- GNU (PrgEnv-gnu)
  - Suggested options: -O3 –ffast-math –funroll-loops
  - Compiler feedback: -ftree-vectorize -verbose=2
  - OpenMP: -fopenmp
  - Man pages: gcc, gfortran, g++

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 closes 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

Cross Compiling Environment

- **Compiling on a Linux service node**

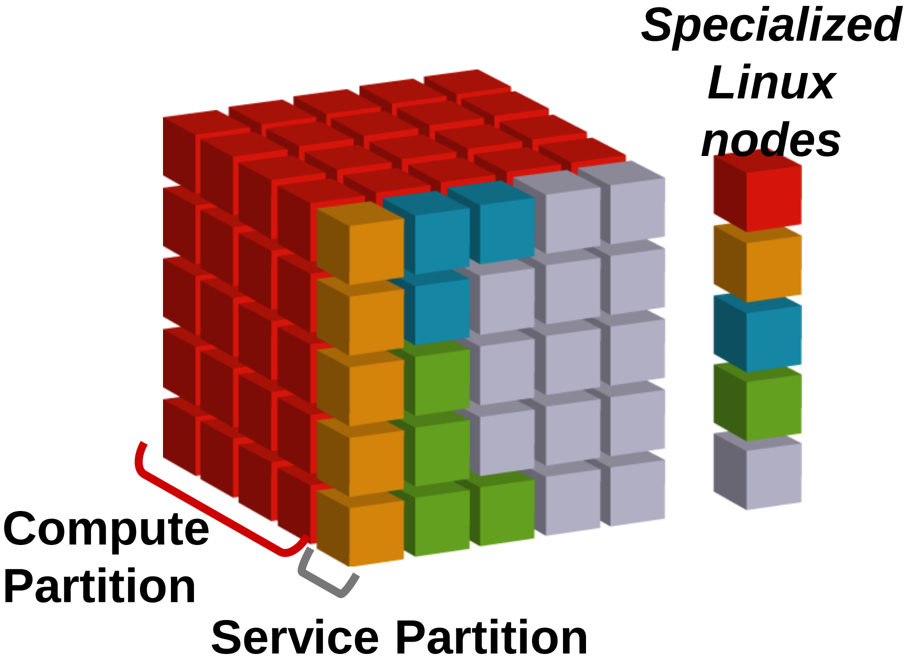
- **Generating an executable for a CLE compute node**

- **Do not use pgf90, pgcc, gcc, g++, ..., unless you want a Linux executable for the service node**
  - Use ftn, cc, or CC instead

- **Information message:**
  - ftn: INFO: linux target is being used

Scalable Software Architecture

Scalable Software Architecture: CLE



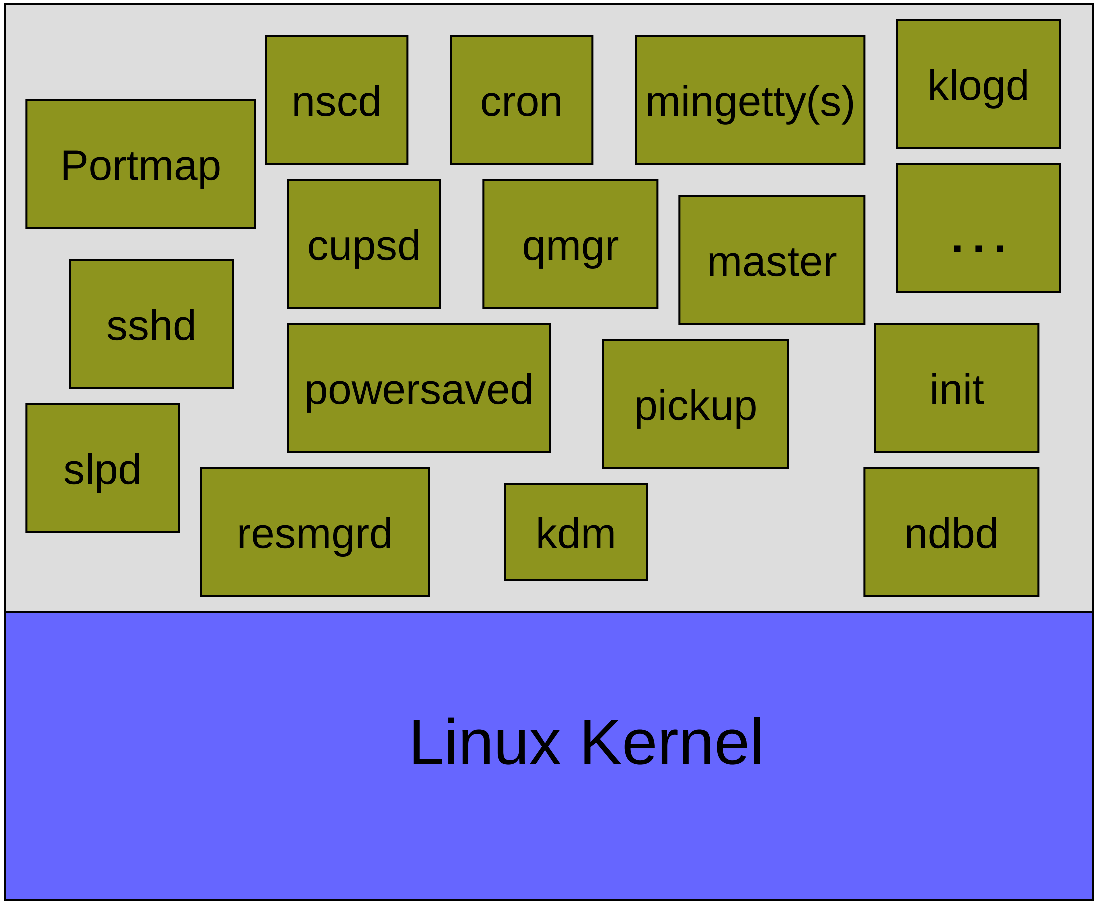
Microkernel on Compute nodes, full featured Linux on Service nodes.

Service PEs specialize by function

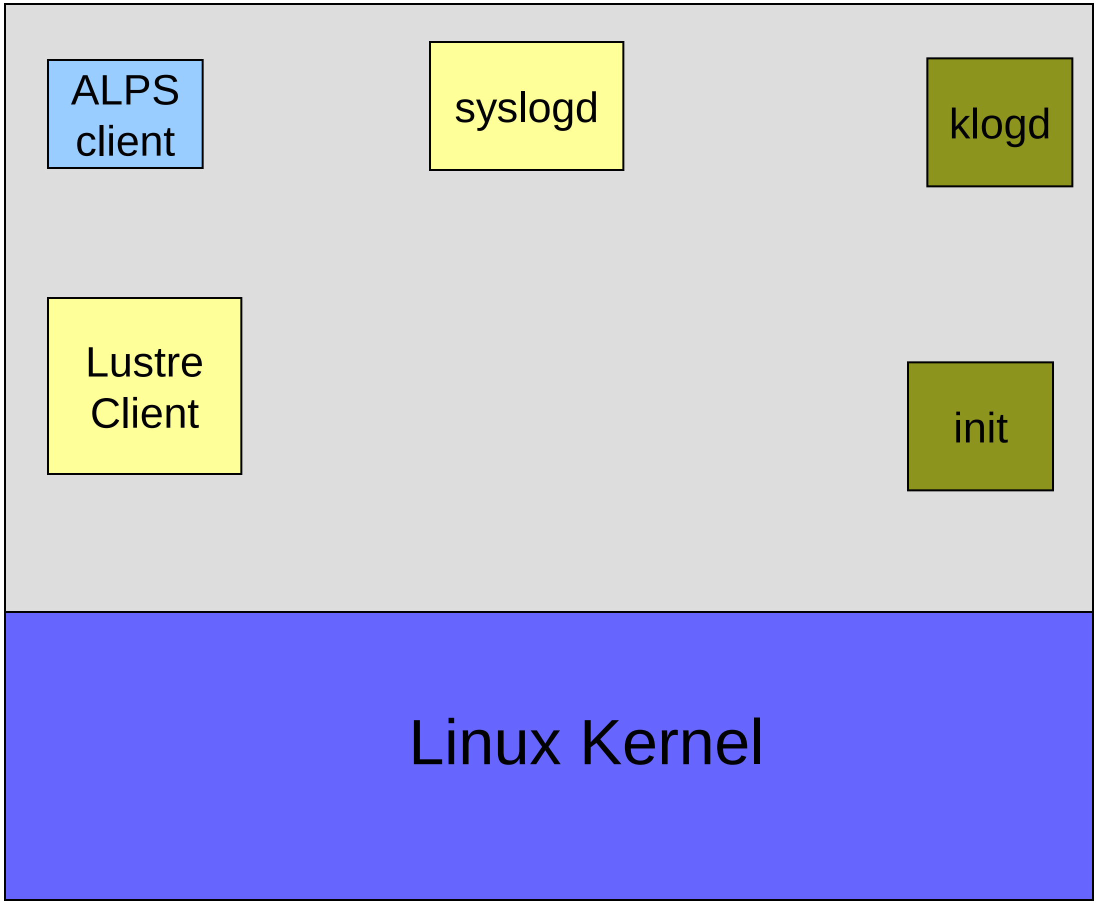
Software Architecture eliminates OS “Jitter”

Software Architecture enables reproducible run times

Trimming OS – *Standard Linux Server*



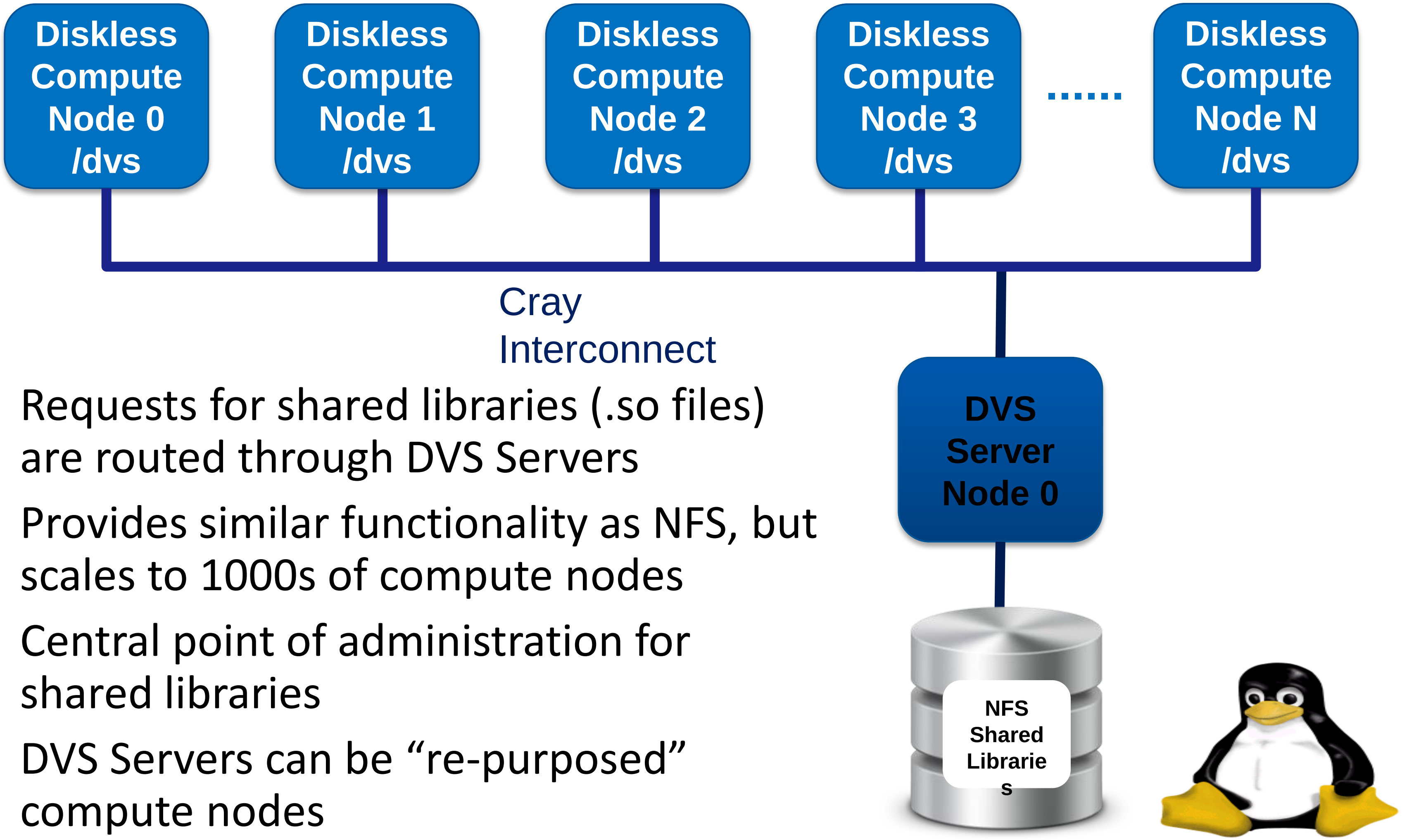
Linux on a Diet – *CNL*



Cray XE I/O architecture

- **All I/O is offloaded to service nodes**
- **Lustre**
  - High performance parallel I/O file system
  - Direct data transfer between compute nodes and files
- **DVS**
  - Virtualization service
  - Allows compute nodes to access NFS mounted on service node
  - Applications must execute on file systems mounted on compute nodes
- **No local disks**
- **/tmp is a MEMORY file system, on each login node**

Scaling Shared Libraries with DVS


Diskless Compute Node 0 /dvs
Diskless Compute Node 1 /dvs
Diskless Compute Node 2 /dvs
Diskless Compute Node 3 /dvs
......
Diskless Compute Node N /dvs
Cray Interconnect
DVS Server Node 0
NFS Shared Librarie s

- Requests for shared libraries (.so files) are routed through DVS Servers
- Provides similar functionality as NFS, but scales to 1000s of compute nodes
- Central point of administration for shared libraries
- DVS Servers can be “re-purposed” compute nodes

DSL : Dynamic shared libraries

- Benefit: root file system environment available to applications
- Shared root from SIO nodes will be available on compute nodes
- Standard libraries / tools will be in the standard places
- Able to deliver customer-provided root file system to compute nodes
- Programming environment supports static and dynamic linking
- Performance impact negligible, due to scalable implementation
- Link with “ftn –dynamic” to create a dynamically linked executable

Running an application on the Cray XE6

Running an application on the Cray XE
ALPS + aprun

- **ALPS : Application Level Placement Scheduler**
- **aprun is the ALPS application launcher**
  - It **must** be used to run application on the XE compute nodes
  - If aprun is not used, the application is launched on the Mom node (and will most likely fail)
  - aprun man page contains several useful examples
  - at least 3 important parameters to control:
    - The total number of PEs : -n
    - The number of PEs per node: -N
    - The number of OpenMP threads: -d
      More precise : The 'stride' between 2 PEs in a node

Some Definitions

- **ALPS is always used for scheduling a job on the compute nodes. It does not care about the programming model you used. So we need a few general 'definitions' :**
  - PE : Processing Elements
    Basically an Unix 'Process', can be a MPI Task, CAF image, UPC tread, …
  - numa_node
    The cores and memory on a node with 'flat' memory access, basically one of the 4 Dies on the Opteron and the direct attach memory.
  - Thread
    A thread is contained inside a process. Multiple threads can exist within the same process and share resources such as memory, while different PEs do not share these resources.
    Most likely you will use OpenMP threads.

Running an application on the Cray XE6
some basic examples

- **Assuming a XE6 IL16 system (32 cores per node)**
- **Pure MPI application, using all the available cores in a node**

```
$ aprun -n 32 ./a.out
```

- **Pure MPI application, using only 1 core per node**
  - 32 MPI tasks, 32 nodes with 32*32 core allocated
  - Can be done to increase the available memory for the MPI tasks

```
$ aprun -N 1 -n 32 -d 32./a.out
(we'll talk about the need for the -d32 later)
```

- **Hybrid MPI/OpenMP application, 4 MPI ranks per node**
  - 32 MPI tasks, 8 OpenMP threads each
  - need to set OMP_NUM_THREADS

```
$ export OMP_NUM_THREADS=8
$ aprun -n 32 -N 4 -d $OMP_NUM_THREADS
```

aprun CPU Affinity control

- **CLE can dynamically distribute work by allowing PEs and threads to migrate from one CPU to another within a node**
- **In some cases, moving PEs or threads from CPU to CPU increases cache and translation lookaside buffer (TLB) misses and therefore reduces performance**
- **CPU affinity options enable to bind a PE or thread to a particular CPU or a subset of CPUs on a node**
- **aprun CPU affinity option (see man aprun)**
  - Default settings : -cc cpu
    PEs are bound a to specific core, depended on the –d setting
  - Binding PEs to a specific numa node : -cc numa_node
    PEs are not bound to a specific core but cannot ‘leave’ their numa_node
  - No binding : -cc none
  - Own binding : -cc 0,4,3,2,1,16,18,31,9,…

Memory affinity control

- **Cray XE6 systems use dual-socket compute nodes with 4 dies**
  - Each die (8 cores) is considered a NUMA-node
- **Remote-NUMA-node memory references, can adversely affect performance.
  Even if you PE and threads are bound to a specific numa_node, the memory used does not have to be 'local'**
- **aprun memory affinity options (see man aprun)**
  - Suggested setting is –ss
    a PE can only allocate the memory local to its assigned NUMA node. If this is not possible, your application will crash.

Running an application on the Cray XE - MPMD

- aprun supports MPMD – Multiple Program Multiple Data

- Launching several executables on the same MPI_COMM_WORLD

```
$ aprun -n 128 exe1 : -n 64 exe2 : -n 64 exe3
```

- Notice : Each exacutable needs a dedicated node, exe1 and exe2 cannot share a node.

  Example : The following commands needs 3 nodes

```
$ aprun -n 1 exe1 : -n 1 exe2 : -n 1 exe3
```

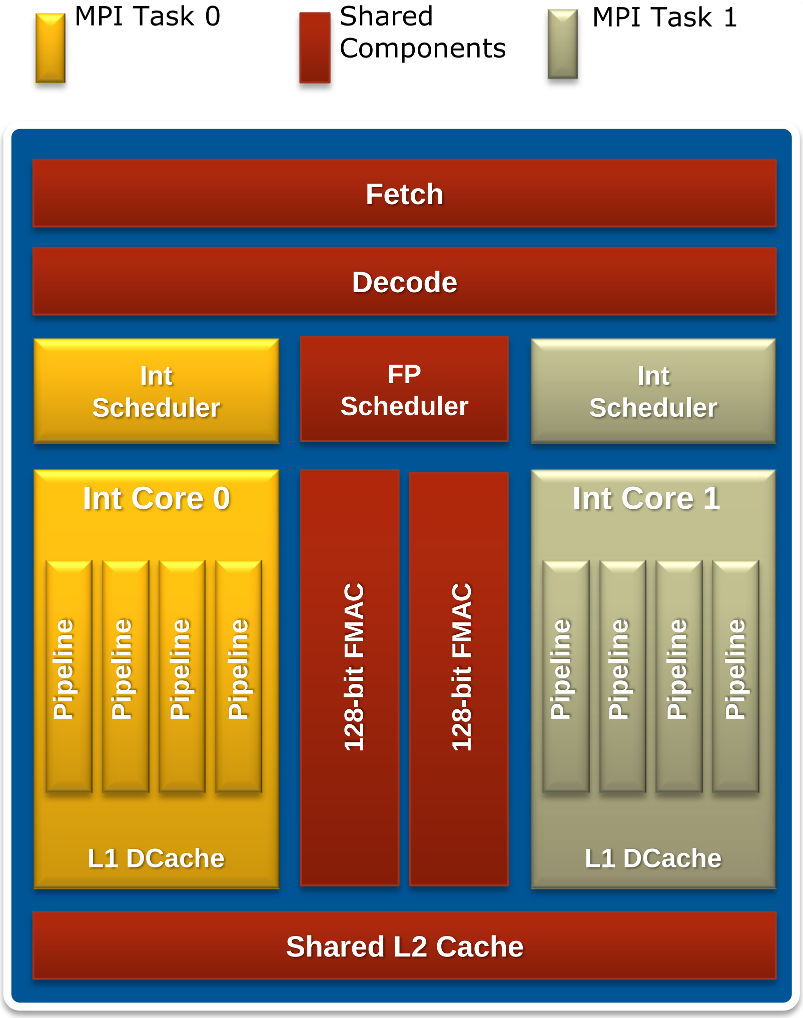
- Use a script to start several serial jobs on a node :

```
$ aprun -a xt -n 3 script.sh
```

```
>cat script.sh
./exe1&
./exe2&
./exe3&
wait
>
```

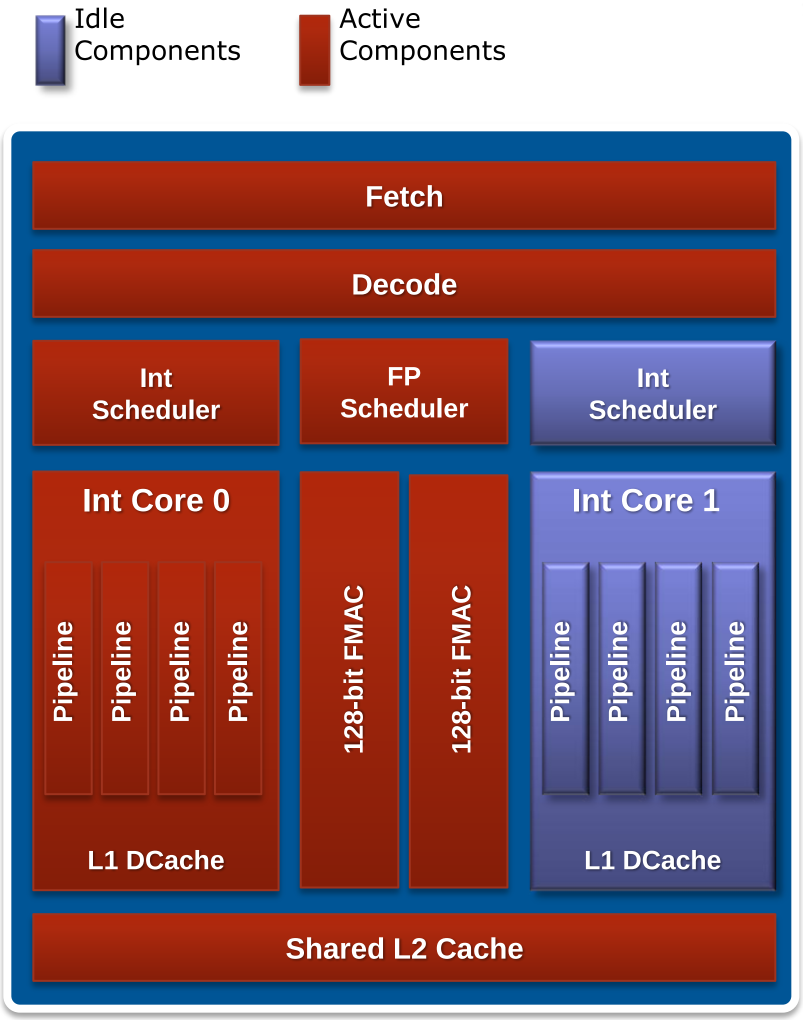
How to use the interlagos 1/3
1 MPI Rank on Each Integer Core Mode

- In this mode, an MPI task is pinned to each integer core
- Implications
  - Each core has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit and the L2 Cache is shared between the two cores
  - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- When to use
  - Code is highly scalable to a large number of MPI ranks
  - Code can run with 1 GB per core memory footprint (or 2 GB on 64 GB node)
  - Code is not well vectorized



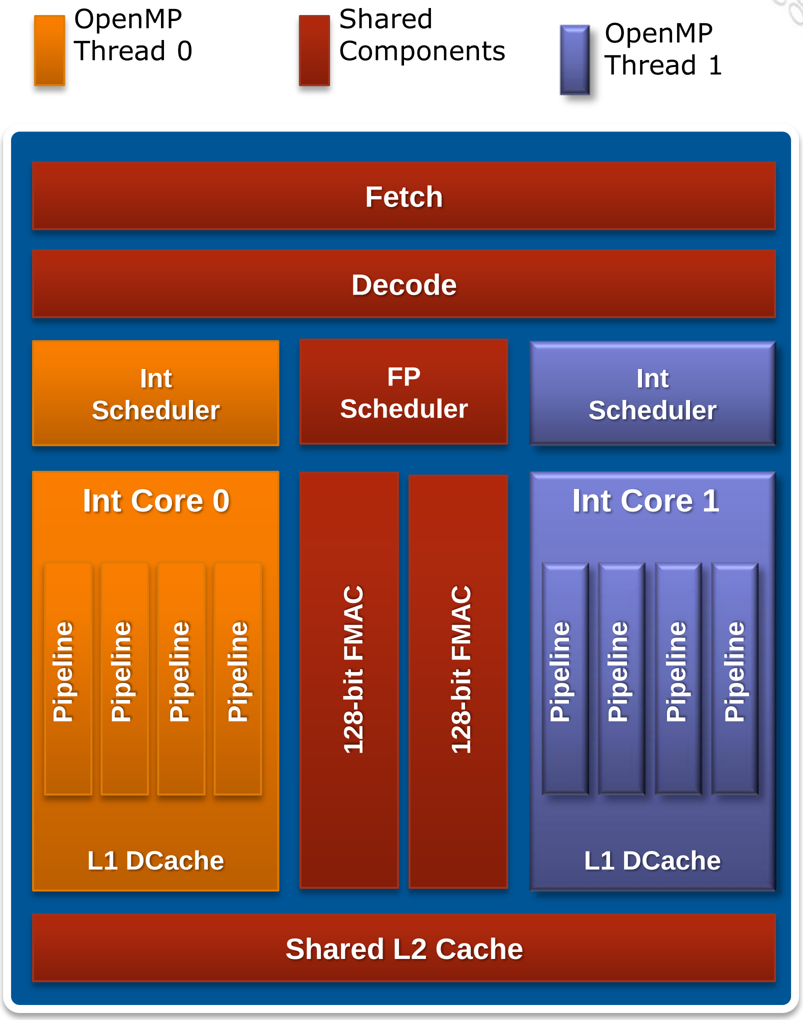
How to use the interlago 2/3
Wide AVX mode

- In this mode, only one integer core is used per core pair
- Implications
  - This core has *exclusive* access to the 256-bit FP unit and is capable of 8 FP results per clock cycle
  - The core has twice the memory capacity and memory bandwidth in this mode
  - The L2 cache is effectively twice as large
  - The peak of the chip is not reduced
- When to use
  - Code is highly vectorized and makes use of AVX instructions
  - Code needs more memory per MPI rank



How to use the interlago 3/3
2-way OpenMP Mode

- In this mode, an MPI task is pinned to a core pair
- OpenMP is used to run a thread on each integer core
- Implications
  - Each OpenMP thread has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit and the L2 Cache is shared between the two threads
  - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- When to use
  - Code needs a large amount of memory per MPI rank
  - Code has OpenMP parallelism exposed in each MPI rank



Aprun: cpu_lists for each PE

- CLE was updated to allow threads and processing elements to have more flexibility in placement. This is ideal for processor architectures whose cores share resources with which they may have to wait to utilize. Separating cpu_lists by colons (:) allows the user to specify the cores used by processing elements and their child processes or threads.
  Essentially, this provides the user more granularity to specify cpu_lists for each processing element.
  Here an example with 3 threads :
  aprun -n 4 -N 4 -cc 1,3,5:7,9,11:13,15,17:19,21,23
- Note: This feature will be modified in CLE 4.0.UP03, however this option will still be valid.

Core specialization

- **System ‘noise’ on compute nodes may significantly degrade scalability for some applications**
- **Core Specialization can mitigate this problem**
  - 1 core per node will be dedicated for system work (service core)
  - As many system interrupts as possible will be forced to execute on the service core
  - The application will not run on the service core
- **Use aprun -r to get core specialization**

```
$ aprun -r -n 100 a.out
```

- **apcount provided to compute total number of cores requirec**

```
$ qsub -l mppwidth=$(apcount -r 1 1024 16)jo
  aprun -n 1024 -r 1 a.out
```

Questions?

- Cray documentation (docs.cray.com)
- HECToR website (www.hector.ac.uk)
- Cray Centre of Excellence for HECToR (www.hector.ac.uk/coe)