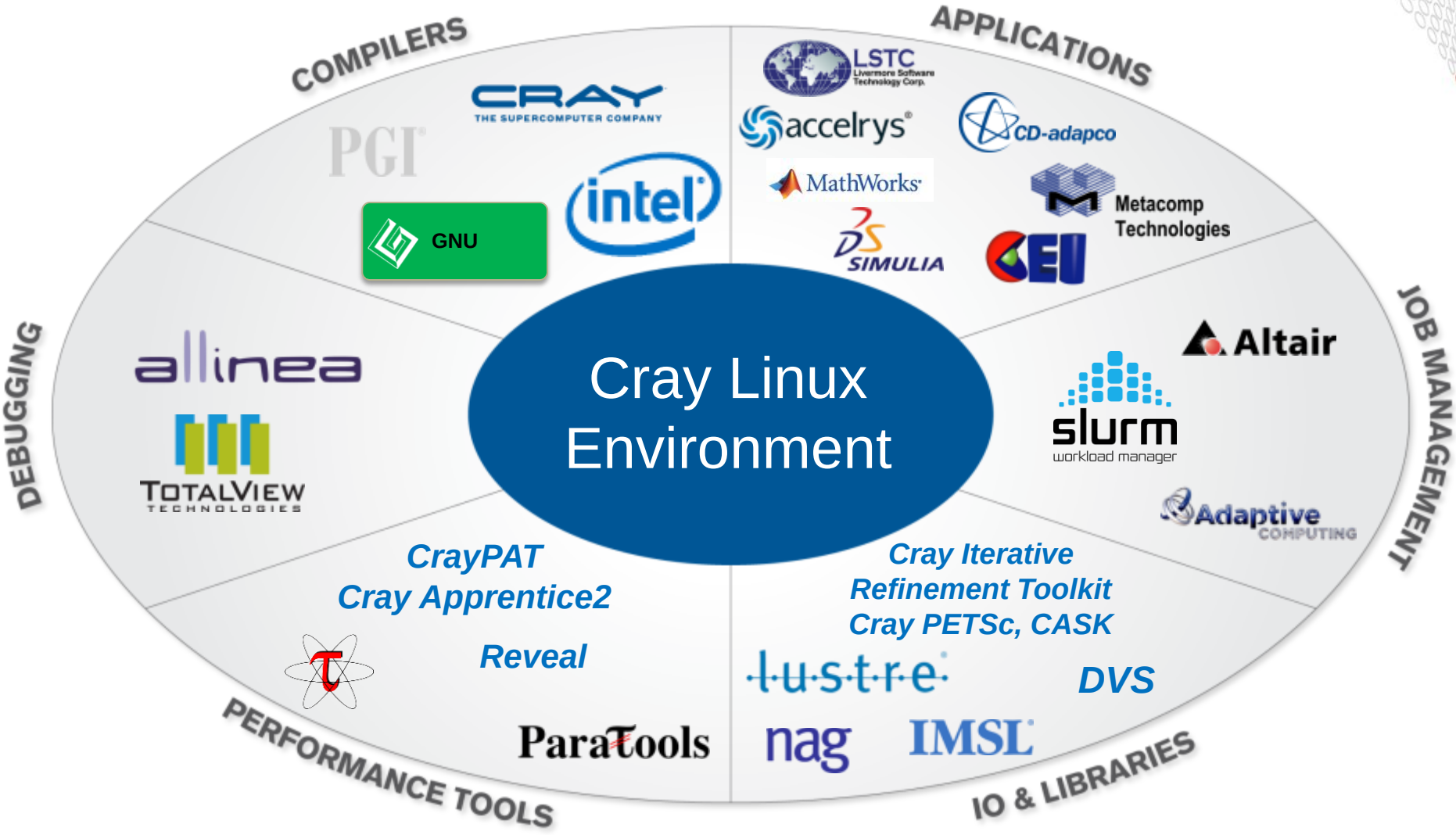
Moving from the: Cray XE6 to the Cray XC30

Continuity

The High Productivity Vision

- **Cray systems are designed to be High Productivity as well as High Performance Computers**
- **The Cray Programming Environment (PE) continues to provide a simple and consistent interface to users and developers.**
  - Renewed focus on improving scalability and reducing complexity
- **The default Programming Environment provides:**
  - the highest levels of application performance
  - a rich variety of commonly used tools and libraries
  - a consistent interface to multiple compilers and libraries
  - an increased automation of routine tasks
- **Cray is committed to extending, developing and refining the PE.**
  - Frequent communication and feedback to/from users
  - Strong collaborations with third-party developers

Cray Software Ecosystem



An Adaptive Linux OS optimized specifically for HPC

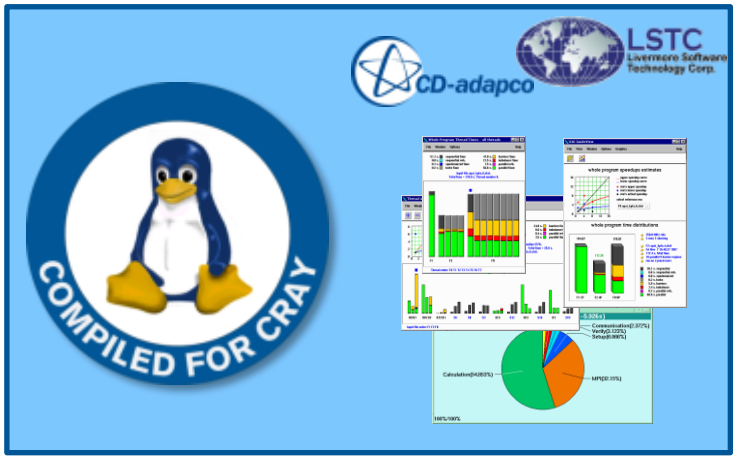
CLE
CRAY LINUX ENVIRONMENT

ESM – *Extreme Scalability Mode*

- No compromise *scalability*
- Low-Noise Kernel for scalability
- Native Comm. & Optimized MPI
- Application-specific performance tuning and scaling

CCM –*Cluster Compatibility Mode*

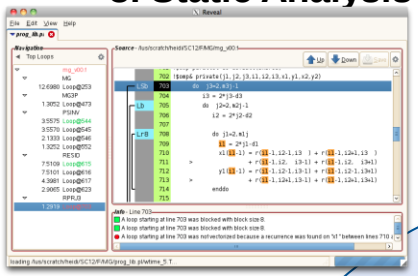
- No compromise *compatibility*
- Fully standard x86/Linux
- Standardized Communication Layer
- Out-of-the-box ISV Installation
- ISV applications simply install and run



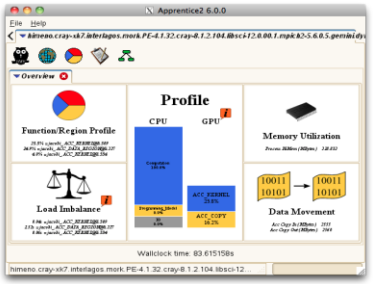
CLE run mode is set by the user on a job-by-job basis to provide full flexibility

Cray Integrated Programming Environment

3. Static Analysis



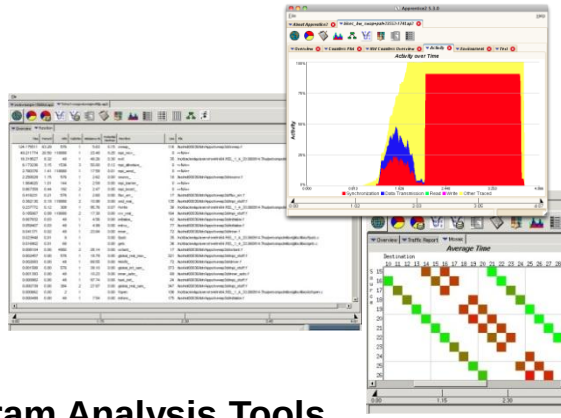
Performance Analysis Overview



Executing Application

Runtime Information

High Level Profile/ Tracing
Performance Problem Analyzer



2. Compiler

Export/Import
Program Analyses

Performance Analysis
Results

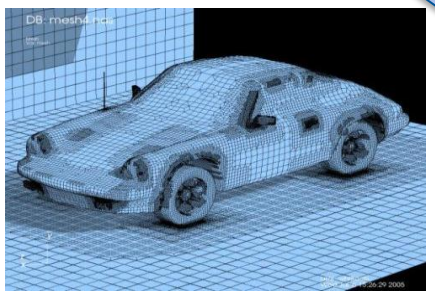
Database containing application information

Performance
Feedback

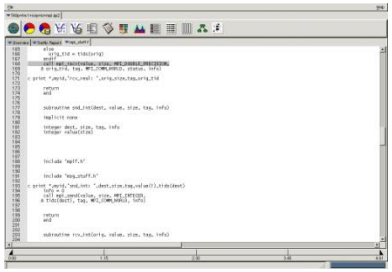
Queries for
Application
Optimization

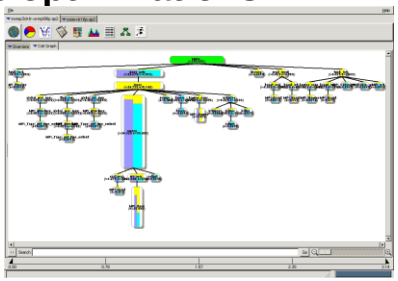
Application
Source code

1. Applications



5. Program Analysis Tools
Source-to-Source optimizations





Cray Programming Environment Distribution 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

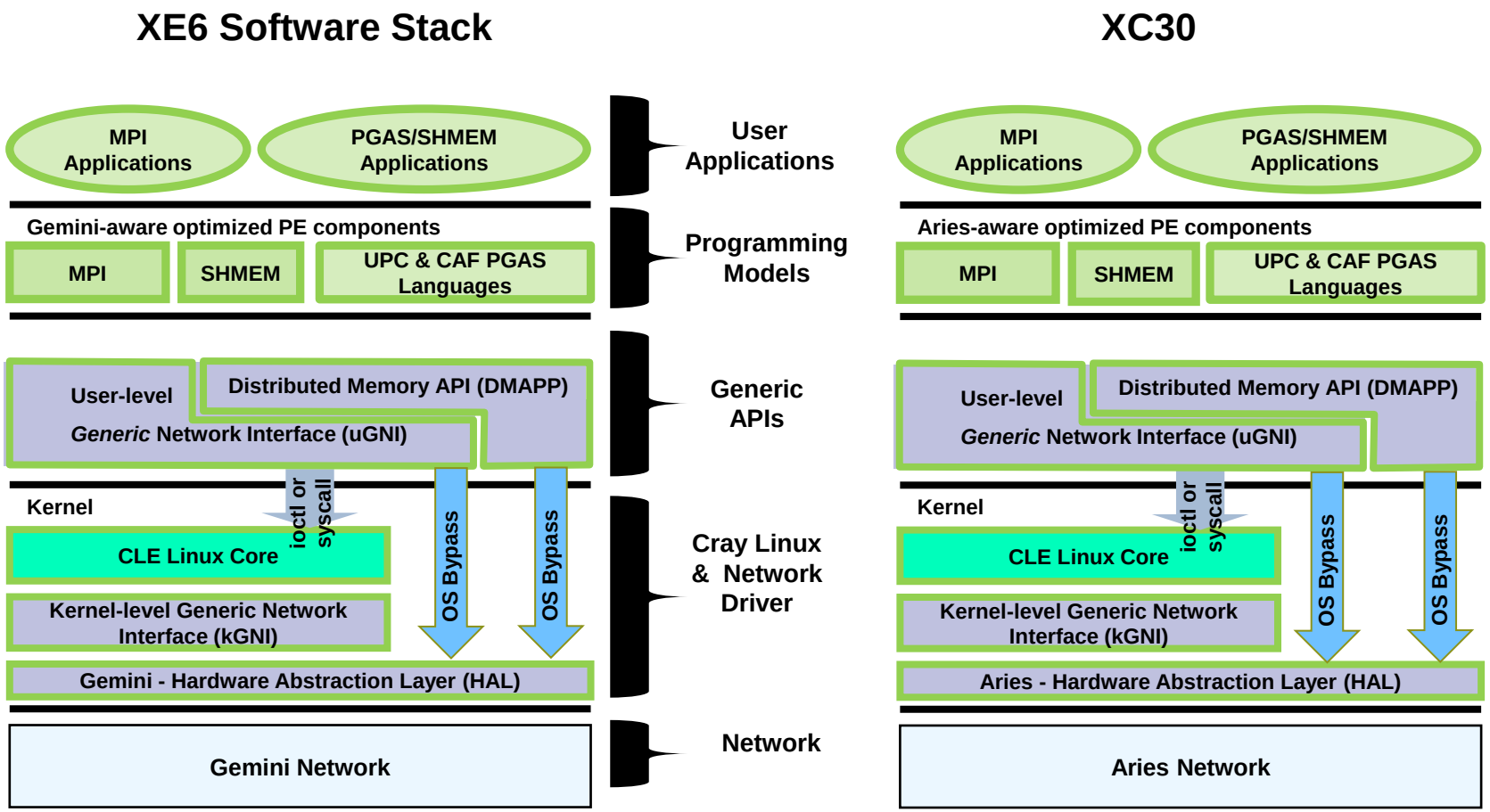
- LAPACK
- ScaLAPACK
- BLAS (libgoto)
- Iterative Refinement Toolkit
- Cray Adaptive FFTs (CRAFFT)
- FFTW
- Cray PETSc (with CASK)
- Cray Trilinos (with CASK)

I/O Libraries

- NetCDF
- HDF5

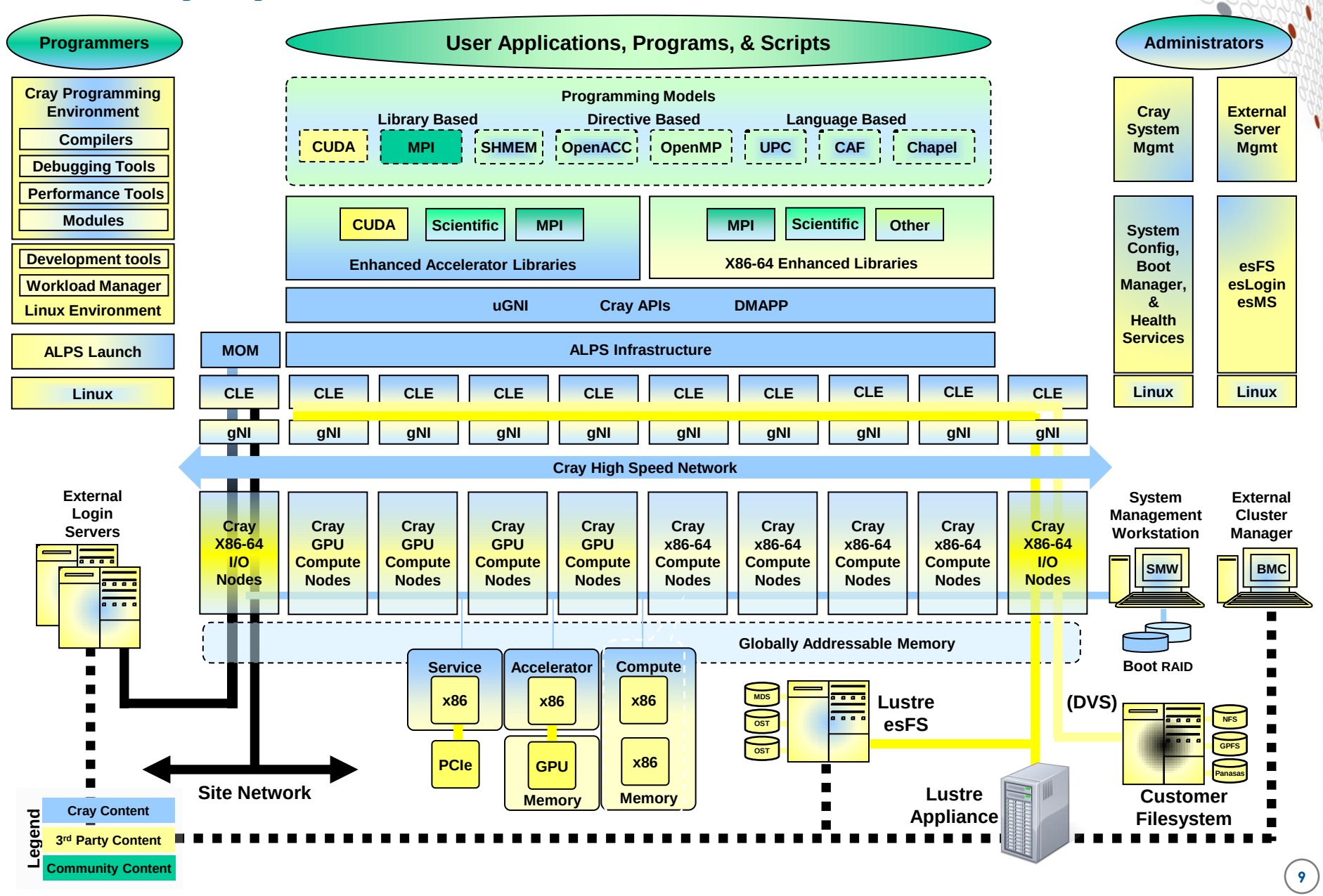
Cray developed
Licensed ISV SW
3rd party packaging
Cray added value to 3rd party

XE6 and XC30 Software Components Generational Commonality



Common Base

The Cray System



Transition

Compilers

Like HECToR will be three compilers installed, however Intel replaces the PGI compiler.

Only the most recent versions will be available (e.g. those released over the last few months)

- **Cray Compilation Environment (CCE)**
  - Coincides with new 8.2 release
- **Intel Composer Suite**
  - New compiler over HECToR
- **GNU Compiler Collection**
  - The standard set of expected compilers and tools.

CCE Overview

- **Cray technology focused on scientific applications**
  - Takes advantage of automatic vectorization
  - Takes advantage of automatic shared memory parallelization
- **Standards conforming languages and programming models**
  - ANSI/ISO Fortran 2003 and Fortran 2008 standards compliant
  - ANSI/ISO C99 and C++2003 compliant
  - OpenMP 3.1 compliant, working on OpenMP 4.0
  - OpenACC 1.0
- **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

Improvements for CCE/8.2.0 on XC30

- Full support for optimising for “ivybridge” processors.
- Improved performance of performance-critical maths intrinsics.
- New Coarray C++ template library that implements coarray concepts to C++.
- GNU and CCE OpenMP libraries are now compatible. Linking must be performed by CCE with PrgEnv-cray.

CCE – GNU – Intel compilers

- **More or less all optimizations and features provided by CCE are available in Intel and GNU compilers**
  - GNU compiler serves a wide range of users & needs
    - Default compiler with Linux, some people only test with GNU
    - Defaults are conservative (e.g. -O1)
      - -O3 includes vectorization and most inlining
    - Performance users set additional options
  - Intel compiler is typically more aggressive in the optimizations
    - Defaults are more aggressive (e.g -O2), to give better performance "out-of-the-box"
      - Includes vectorization; some loop transformations such as unrolling; inlining within source file
    - Options to scale back optimizations for better floating-point reproducibility, easier debugging, etc.
    - Additional options for optimizations less sure to benefit all applications
  - CCE is even more aggressive in the optimizations by default
    - Better inlining and vectorization
    - Aggressive floating-point optimizations
    - OpenMP enabled by default

Cray, Intel and GNU compiler flags

Feature	Cray	Intel	GNU
Listing	-hlist=a	-opt-report3	-fdump-tree-all
Free format (ftn)	-f free	-free	-ffree-form
Vectorization	By default at -O1 and above	By default at -O2 and above	By default at -O3 or using -ftree-vectorize
Inter-Procedural Optimization	-hwp	-ipo	-flto (note: link-time optimization)
Floating-point optimizations	-hfpN, N=0...4	-fp-model [fast|fast=2|precise| except|strict]	-f[no-]fast-math or -funsafe-math-optimizations
Suggested Optimization	(default)	-O2 -xAVX	-O2 -mavx -ftree-vectorize -ffast-math -funroll-loops
Aggressive Optimization	-O3 -hfp3	-fast	-Ofast -mavx -funroll-loops
OpenMP recognition	(default)	-fopenmp	-fopenmp
Variables size (ftn)	-s real64 -s integer64	-real-size 64 -integer-size 64	-freal-4-real-8 -finteger-4-integer-8

Compiler man pages and documentation

- **For more information on individual compilers**

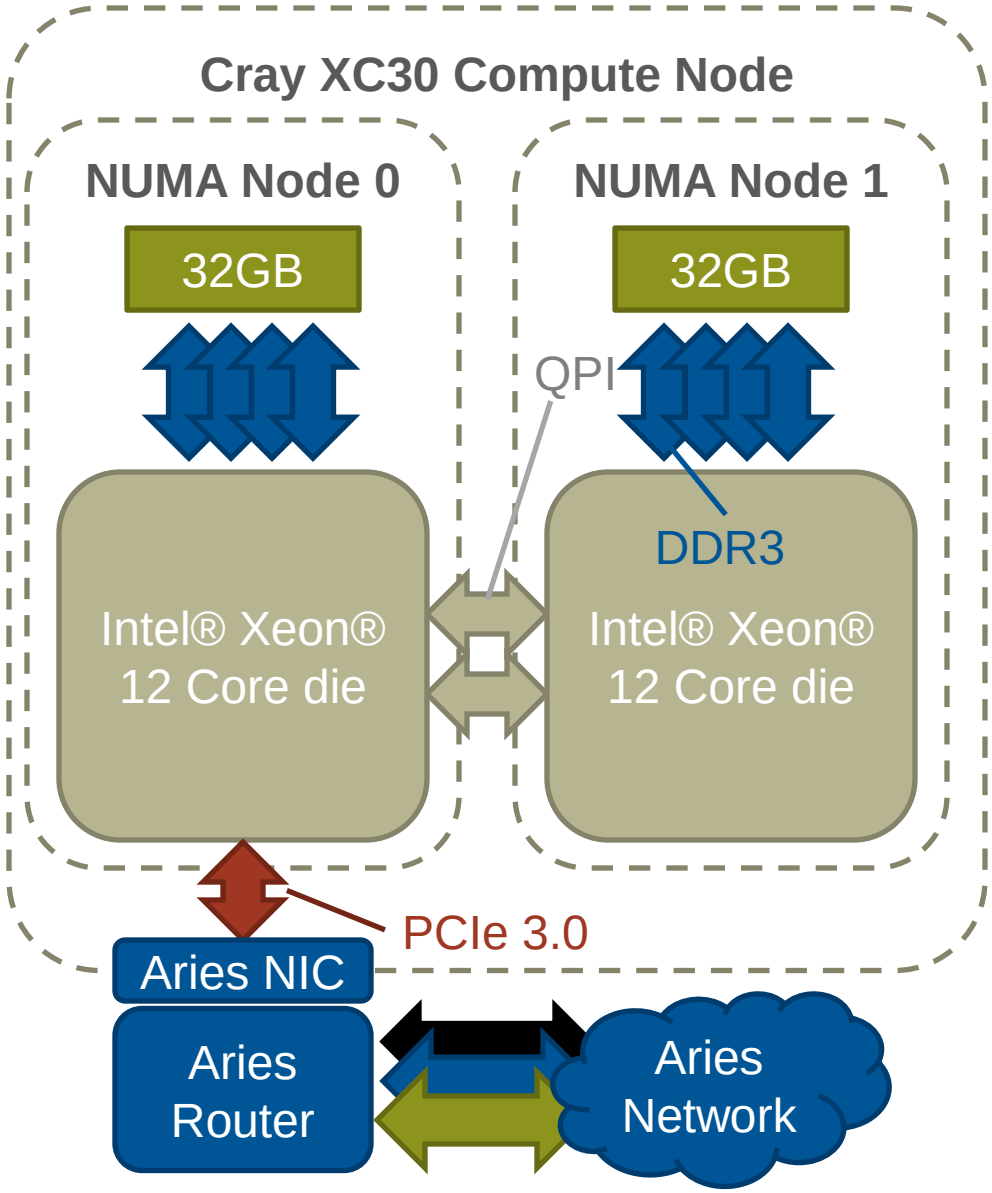
PrgEnv	C	C++	Fortran
PrgEnv-cray	man craycc	man crayCC	man crayftn
PrgEnv-intel	man icc	man icpc	man ifort
PrgEnv-gnu	man gcc	man g++	man gfortran
Wrappers	man cc	man CC	man ftn

- **To verify that you are using the correct version of a compiler, use:**
  - **-V** option on a cc, CC, or ftn command with CCE and Intel
  - **--version** option on a cc, CC, or ftn command with GNU
- **Cray Reference Manuals:**
  - C and C++: http://docs.cray.com/books/S-2179-81/
  - Fortran: http://docs.cray.com/books/S-3901-81/

Interlagos/Ivybridge Comparison

	AMD Opteron “Interlagos”	Intel Xeon “Ivybridge”
Base Clock Speed	2.3 GHz	2.7 GHz
Cores per die	6	12
Dies per node	4	2
Each cores has:		
User threads	1	2
Function group	1 SSE (vector)	1 AVX (vector)
bits wide	128 bits wide	256 bits wide
functional units	1 add and 1 multiply	1 add and 1 multiply
Cache: L1	32KB	32KB
Cache: L2	512KB	256KB
L3 Cache (per die)	6 MB	30 MB
Total Cache per core	1.5 MB	2.75 MB
Cache BW Per core (GB/s)		
L1/L2/L3	35 / 3.2 / 3.2	100 / 40 / 23
Stream TRIAD BW/node	52 Gbytes/s	100 Gbytes/s
Peak DP FLOPs per core	4 flops/clk	8 flops/clk
Peak DP FLOPs per node	294 GFlops	518 GFlops
Main memory latency	110ns	82ns

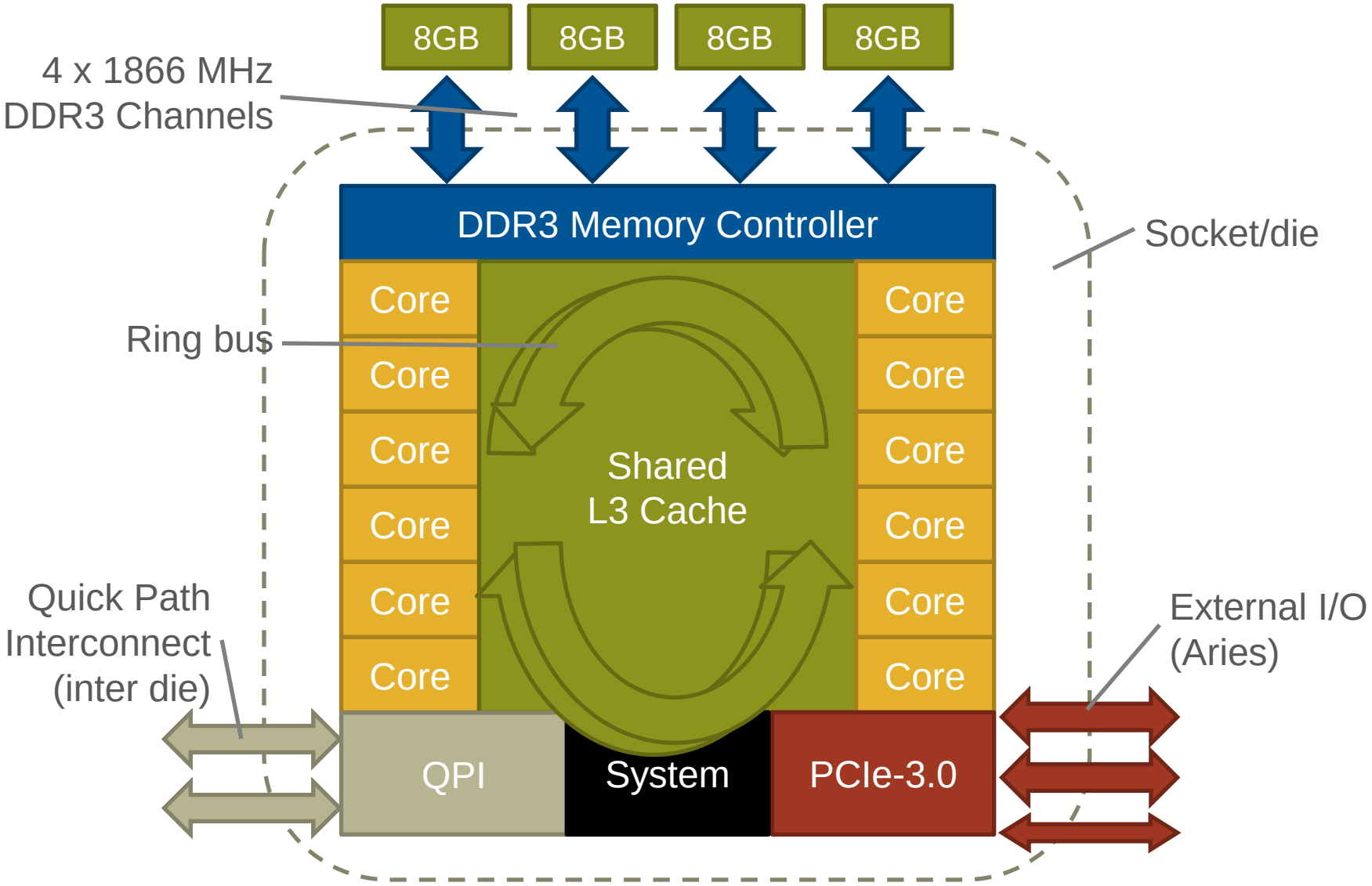
Cray XC30 Intel® Xeon® Compute Node



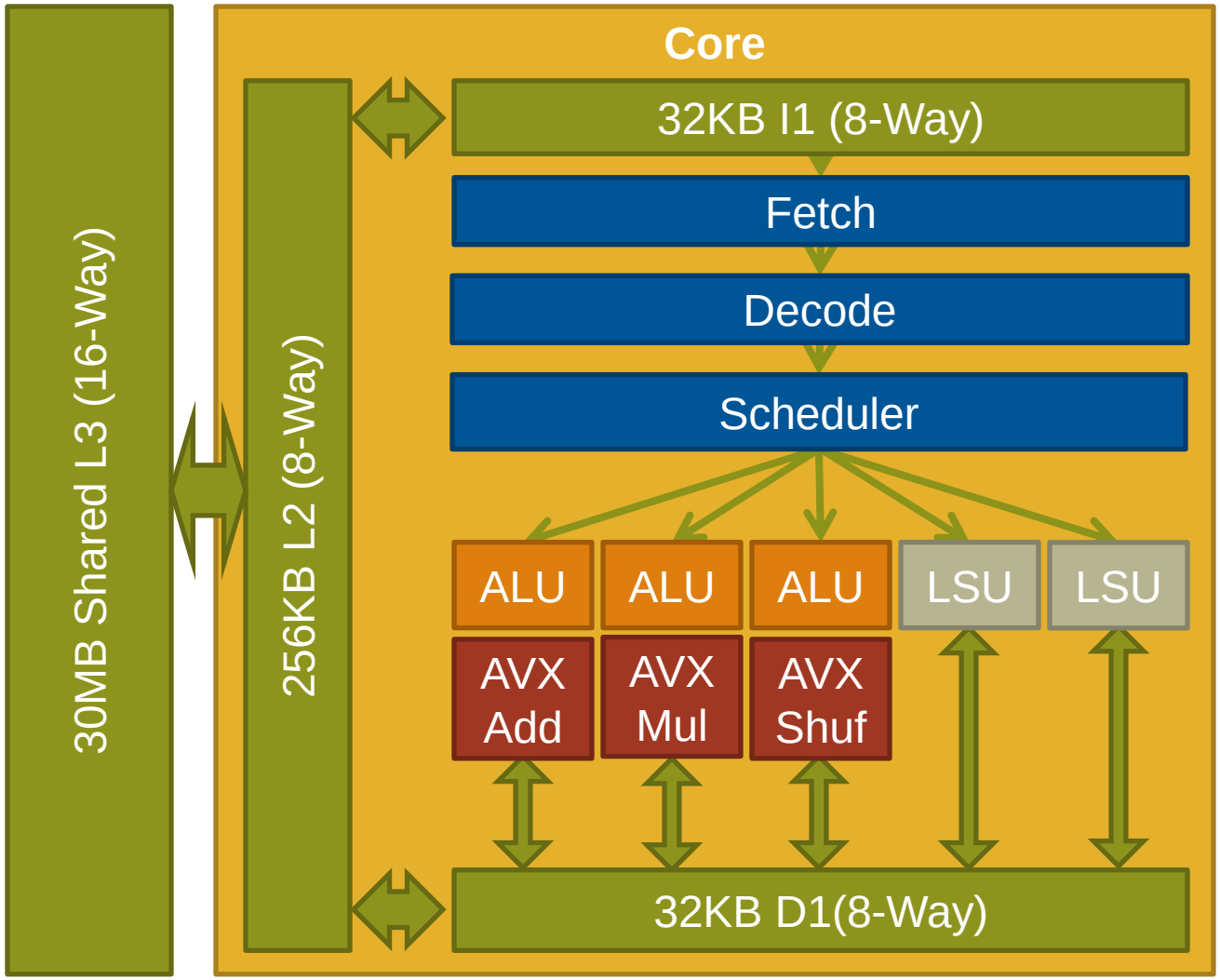
The XC30 Compute node features:

- **2 x Intel® Xeon® Sockets/die**
  - 12 core Ivybridge
  - QPI interconnect
  - Forms 2 NUMA nodes
- **8 x 1833MHz DDR3**
  - 8 GB per Channel
  - 64 GB total
- **1 x Aries NIC**
  - Connects to shared Aries router and wider network
  - PCI-e 3.0

Intel® Xeon® Ivybridge 12-core socket/die



Intel Xeon Ivybridge Core Structure



- Manufactured on a 22nm Process
- 256 bit AVX Instructions (4 double precision floating point)
  - 1 x Add
  - 1 x Multiply
  - 1 x Other
- 2 Hardware threads (Hyperthreads)
- Peak DP FP per node 8FLOPS/clock

Placement and Scheduling

Scheduling and Placement

There are always two stages to launching a job on a Cray XC30 with a batch scheduler.

1. Requesting the right amount of resource from the batch scheduler

2. Launching the parallel application on the allocated compute nodes

Both of these stages have been affected by upgrades to software and hardware when compared to Cray XE6

Glossary of terms

PE/Processing Element

- A discrete software process with an individual address space. One PE is equivalent to:

1 MPI Rank, 1 Coarray Image, 1 UPC Thread, or 1 SHMEM PE

Threads

- A logically separate stream of execution inside a parent PE that shares the same address space

CPU

- The minimum piece of hardware capable of running a PE. It may share some or all of its hardware resources with other CPUs
  Equivalent to a single “Intel Hyperthread”

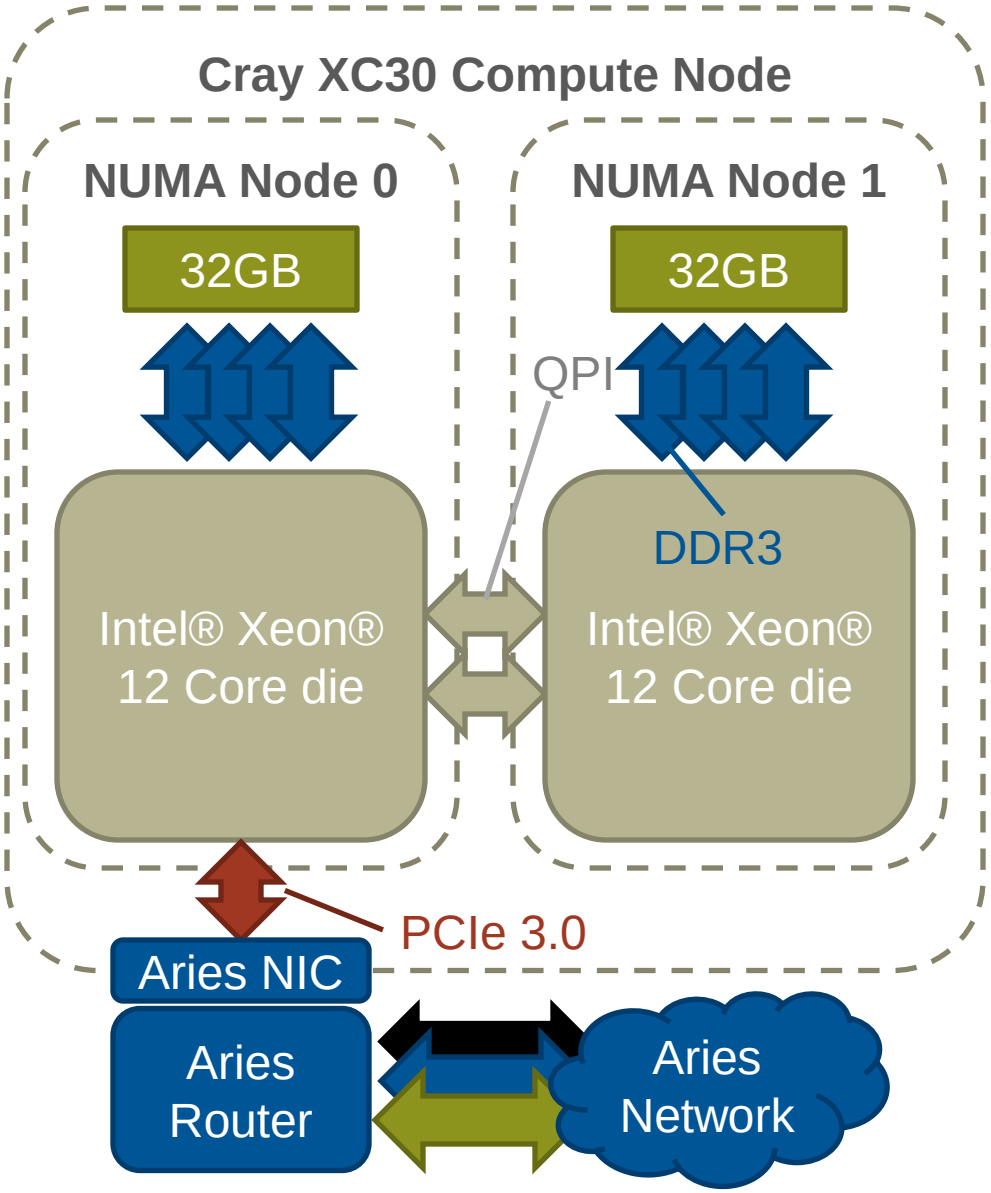
Compute Unit

- The individual unit of hardware for processing, may be seen described as a “core”. Poite may provide multiple CPUs.

Hyperthreads

- **Hyper-threads are a feature of modern Intel processors**
  - Essentially they are a form of hardware multithreading that avoids costly context switches.
  - Designed to mask long pauses in execution, e.g. network communication, memory accesses.
- **Allows OS to schedule two processes (PEs) simultaneously on the same hardware core (Compute unit)**
  - Each thread context is held in hardware and scheduled automatically (virtuall no overhead of a context switch)
- **The Cray XC30 software and hardware stacks provide full support for using Hyperthreads.**
  - 48 CPUs visible to the OS
    - CPUs 0-11 & 24-35 on Socket 0
    - CPUs 12-23 & 36-47 on Socket 1
    - Shared compute unit pairs of CPUs are 0&24, 7&31, 8&32 and 15&39 etc.

Ivybridge – Single Stream Mode



An Ivybridge XC30 blade has two sockets, each with 12 Intel Cores. Each Intel core is termed a “Compute Unit”.

In “Single stream” mode we elect to use only one Hyperthread (CPU) from each of these “Compute Units”

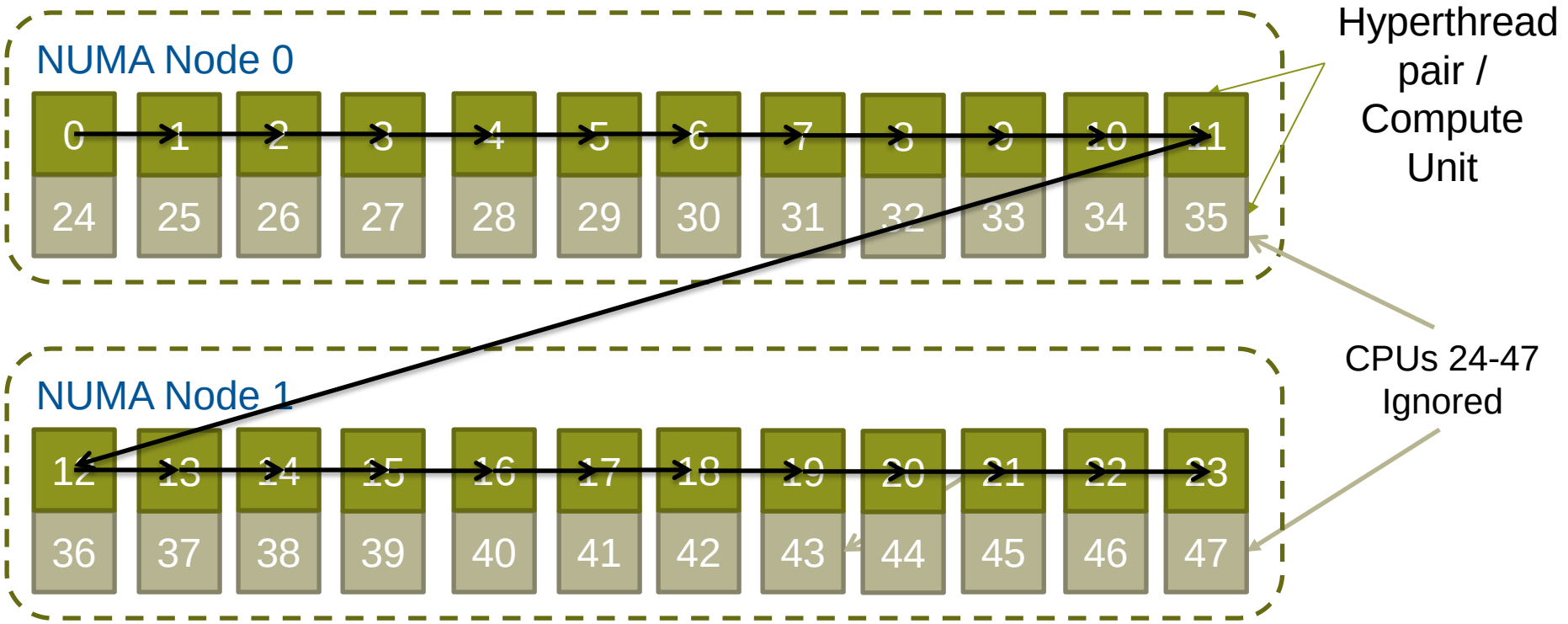
This means there is a total of 24 “CPUs” per node.

Therefore , without oversubscribing, there can only be a maximum of 24 PEs and threads assigned to one node.

Ignore Hyperthreads "-j1" Single Stream Mode

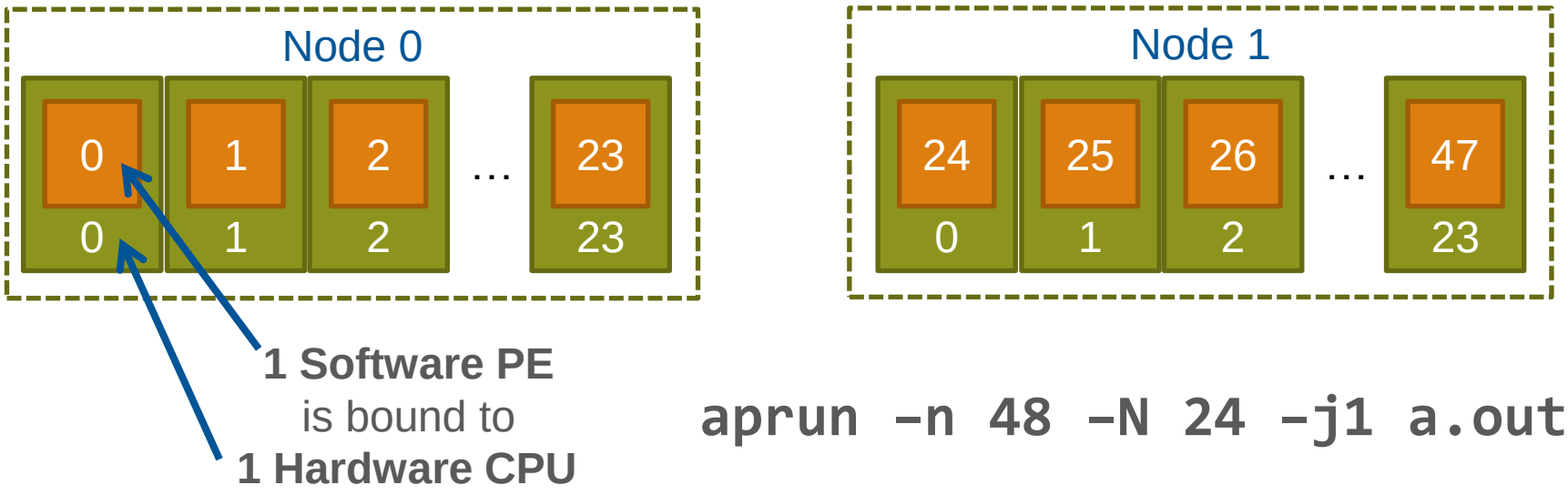
Users can choose whether to use single or dual stream mode at runtime using aprun's "-j1" or "-j2" options.

"-j1" is single stream mode where, aprun binds PEs and ranks to the 24 Compute Units (e.g. only use CPUs 0-23)



Default Binding - CPU

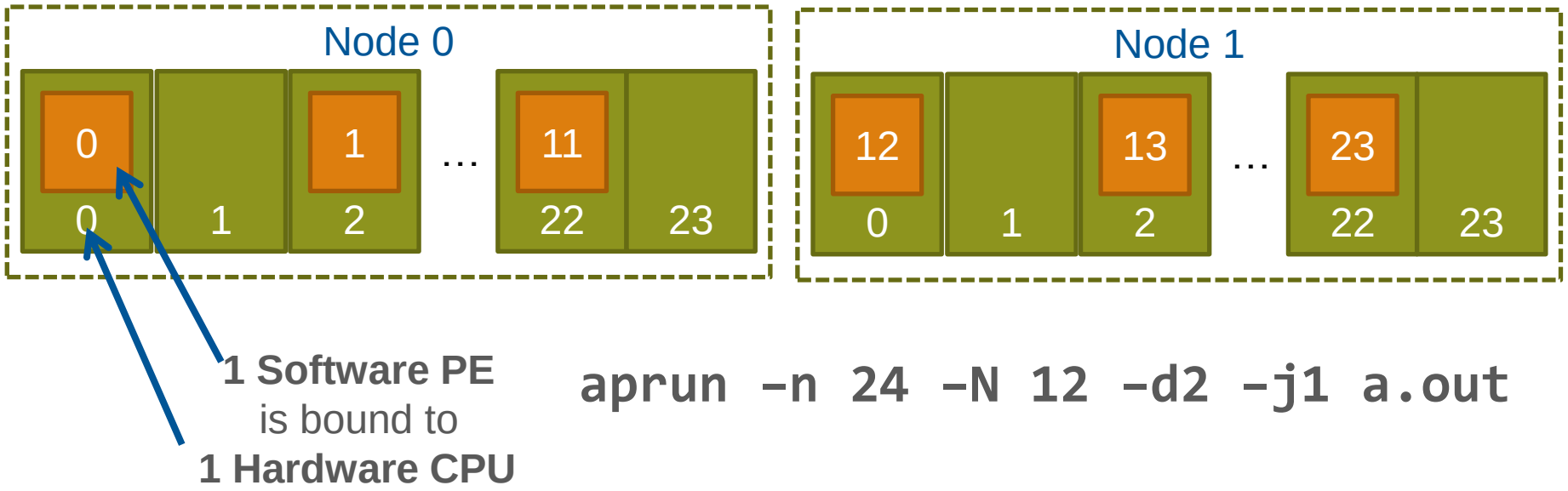
- By default aprun will bind each PE to a single CPU for the duration of the run.
- This prevents PEs moving between CPUs.
- All child processes of the PE are bound to the same CPU
- PEs are assigned to CPUs on the node in increasing order from 0. e.g.




```
aprun -n 48 -N 24 -j1 a.out
```

Default Thread Binding (pt 1)

- You can inform aprun how many threads will be created by each PE by passing arguments to the -d (depth) flag.
- aprun does not create threads, just the master PE.
- PEs are bound to the a single CPU and reserve space according to the depth argument, e.g

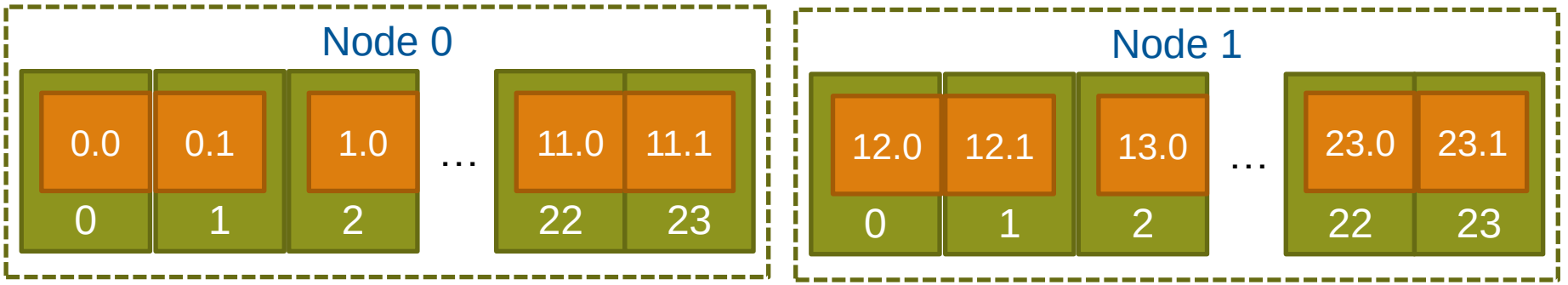



```
aprun -n 24 -N 12 -d2 -j1 a.out
```

Default Thread Binding (pt 2)

- Each subsequently created child processes/thread is bound by the OS to the next CPU *(modulo by the depth argument).* e.g.

```
OMP_NUM_THREADS=2
aprun -n 24 -N 12 -d2 -j1 a.out
```

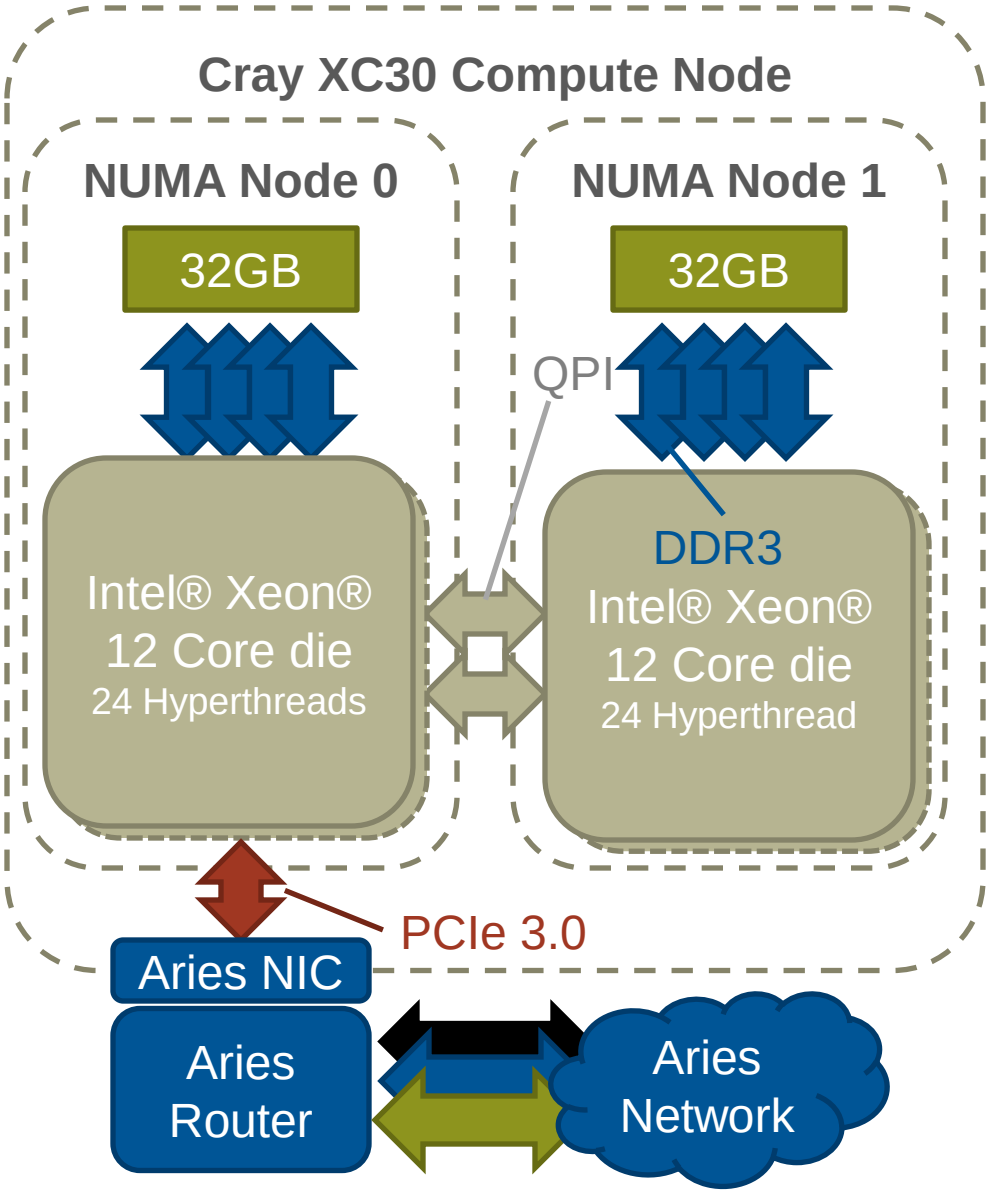


- Each PE becomes the master thread and spawns a new child thread. The OS binds this child thread to the next CPU.

BEWARE – Intel Helper Threads

- **The Intel OpenMP runtime is different to GNU and CCE.**
  - It creates an extra thread as a shepherd … (n+1 threads spawned)
  - It also has it's own method of binding to CPUs (KMP_AFFINITY)
- **Unfortunately both of these options can make things more complicated on Cray XC30 with aprun binding features.**
- **Cray's default advice…**
  - Don't use KMP_AFFINITY to bind threads:
    - `export KMP_AFFINITY=disabled`
    - `aprun -cc [numa_node|none] <exe>`
  - Study man aprun in detail
  - Contact the helpdesk/Centre of Excellence

Ivybridge – Dual Stream Placement



The alterative mode is “Dual stream” mode each of these “Compute Units” can host two Hyperthreads, or “CPUs”.

This means there is a total of 48 CPUs per node.

And so the maximum number of PEs and threads that can be assigned without oversubscription is 48 PEs.

Include Hyperthreads "-j2" Dual Stream Mode

Specifying "`-j2`" in aprun assigns PEs to all of the 48 CPUs available. However CPUs that share a common Compute Unit are assigned consecutively

This means threads will share Compute Units with default binding

Some Hyperthread advice

- **Pure MPI, Mix-mode/hybrid and MPMD all continue to function as they did on Cray XE6 by default.**
  - Just remember that number of NUMA nodes drops to 2.
- **By default, aprun will use only 1 CPU per compute unit.**
  - We expect this to the optimal way to run most HPC codes.
- **The default mode leaves the Hyperthreads open for the Operating System to use**
  - Can be used for CPU specialisation and MPI Progress engines.
- **We expect only codes with low computational intensity to benefit from actively using hyperthreads**
  - Others may transparently benefit from the additional OS and library features

PBS Pro 12 - Requesting resources

- **The version of PBS has changed between HECToR and Archer (now running PBS 12)**
- **This is a requirement, but it also allows for several improvements that were previously unavailable**
  - The numbers of queues will be significantly reduced (perhaps just 4)
  - One of these queues will potentially support quick job turn-around create a “debug” queue
- **However, the upgrade has changed the way users ask for resources from the PBS batch scheduler.**
  - Users will now request resources in quanta of nodes rather than MPI ranks
  - Older mpp* style notation will be rejected by the scheduler as it is no longer accepted.

PBS Select notation

PBS now asks users to select “chunks” of resources for their jobs.

This replaces the older notation so users must remove all `mppwidth`, `mppnppn` and `mppdepth` statements from batch scripts.

The simplest way to think of a chunk is as one entire node of the XC30. Users can submit jobs requesting numbers of nodes using:

```
#PBS -l select=<num_nodes>
```

(or via the command line qsub options)

Writing job scripts

ARCHER jobs will have access to new environment variables, e.g. `$NUM_NODES` which contains the number of nodes allocated to the job.

Scripts are then free to launch jobs via aprun using any layout as long as it does not exceed the number of nodes allocated. E.g. for generic aprun

```
aprun -n n -N N -d d -j j a.out
```

$$(d \times N) <= (24 * j)$$

$$\text{ceiling}(n / N) <= \$NUM\_NODES$$

NB. OMP_NUM_THREADS will be set by default to 1.
All job scripts should set or unset this variable as necessary.

PBS – Fine tuning

PBS allows to users to specify more details about how many PEs and OpenMP threads will run in a chunk.

Users can specify:

```
-l select=<#nodes>:mpiprocs=<#ppn>:ompthreads=<#threads>
```

On ARCHER jobs default to `mpiprocs=24` and `ompthreads=1`.

Each job will be launched with environment variables:

```
$NUM_NODES=<#nodes>
$NUM_PPN=<#ppn>          # copied from mpiprocs
$NUM_DEPTH=<#threads> # copied from ompthreads
```

Some simple examples

```
#!/bin/bash
#PBS -l select=64:mpiprocs=24
#PBS -l walltime=6:00:00

NUM_WIDTH=$[ ${NUM_NODES} * ${NUM_PPN} ]
cd ${PBS_O_WORKDIR}

export OMP_NUM_THREADS=1 # Added for safety
aprun -n ${NUM_WIDTH} -N ${NUM_PPN} mpiapp.exe
```

```
#!/bin/bash
#PBS -l select=32:mpiprocs=24:ompthreads=2
#PBS -l walltime=12:00:00

NUM_WIDTH=$[ ${NUM_NODES} * ${NUM_PPN} ]
cd ${PBS_O_WORKDIR}

export OMP_NUM_THREADS=${NUM_DEPTH}
aprun -n ${NUM_WIDTH} -N ${NUM_PPN} -d ${NUM_DEPTH} -j2 app.exe
```

The simplest example

```
#!/bin/bash
#PBS -l select=64
#PBS -l walltime=6:00:00

cd ${PBS_O_WORKDIR}

aprun -B mpiapp.exe
```

ARCHER queues default to `mpiprocs=24:ompthreads=1`

Therefore this aprun will use:

```
aprun -n 1536 -N 24 -d 1 -j1 mpiapp.exe
```

Using the Higher Memory Nodes

ARCHER has one group of nodes that has 128 GB of main memory per node.

To tell the scheduler that you require these nodes, add `bigmem=true` to the select request.

e.g.

```
-l select=64:mpiprocs=24:ompthreads=1:bigmem=true
```

Libraries

Intel Math Kernel Library (a potential cray-libsci alternative)

Intel provides the Intel Math Kernel Library (MKL) with the composer suite.

Like cray-libsci, MKL provides sets of routines for scientific and engineering applications (also includes financial applications).

Routines are highly optimised for Intel's own processors and can offer extremely good performance.

Distributed as mutually exclusive libraries, it is not a modle. Instead advice on linking with Intel and GNU compilers can be found here:

http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor

Linking with MKL and PrgEnv-cray

- **PrgEnv-cray compatible with sequential, not threaded, MKL**
- **Assume you have loaded the intel module as well as cce (this defines the `$INTEL_PATH`)**
  - Typical case: You want to use MKL BLAS and/or LAPACK

```
-L ${INTEL_PATH}/mkl/lib/intel64/ \
-Wl,--start-group \
-lmkl_intel_lp64 -lmkl_sequential -lmkl_core \
-Wl,--end-group
```

  - Another typical case: You want to use MKL serial FFTs/DFTs

Same as above (need more for FFTW interface)

  - A less typical case: You want to use MKL distributed FFTs

```
-L ${INTEL_PATH}/mkl/lib/intel64/ \
-Wl,--start-group \
-lmkl_cdft_core -lmkl_intel_lp64 -lmkl_sequential \
-lmkl_core -lmkl_blacs_intelmpi_lp64 \
-Wl,--end-group
```

- **The Intel MKL Link Line Advisor can tell you what to add to your link line**
  - http://software.intel.com/sites/products/mkl/

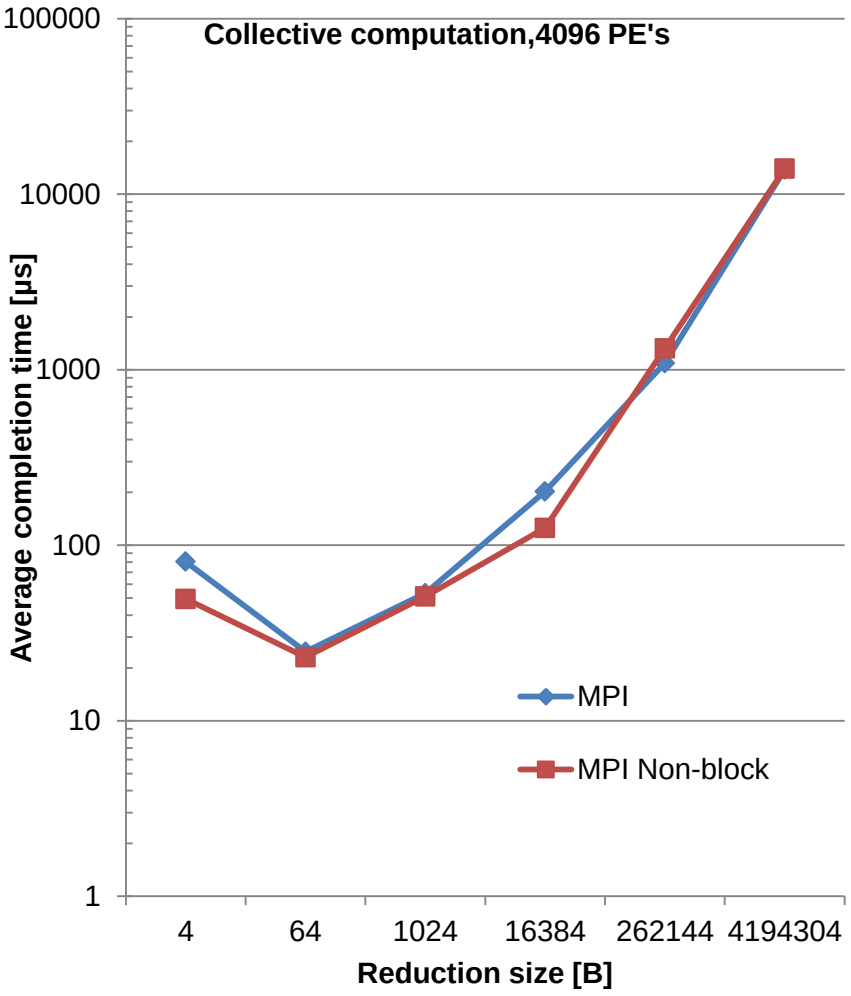
MPI Features / Functionality for XC

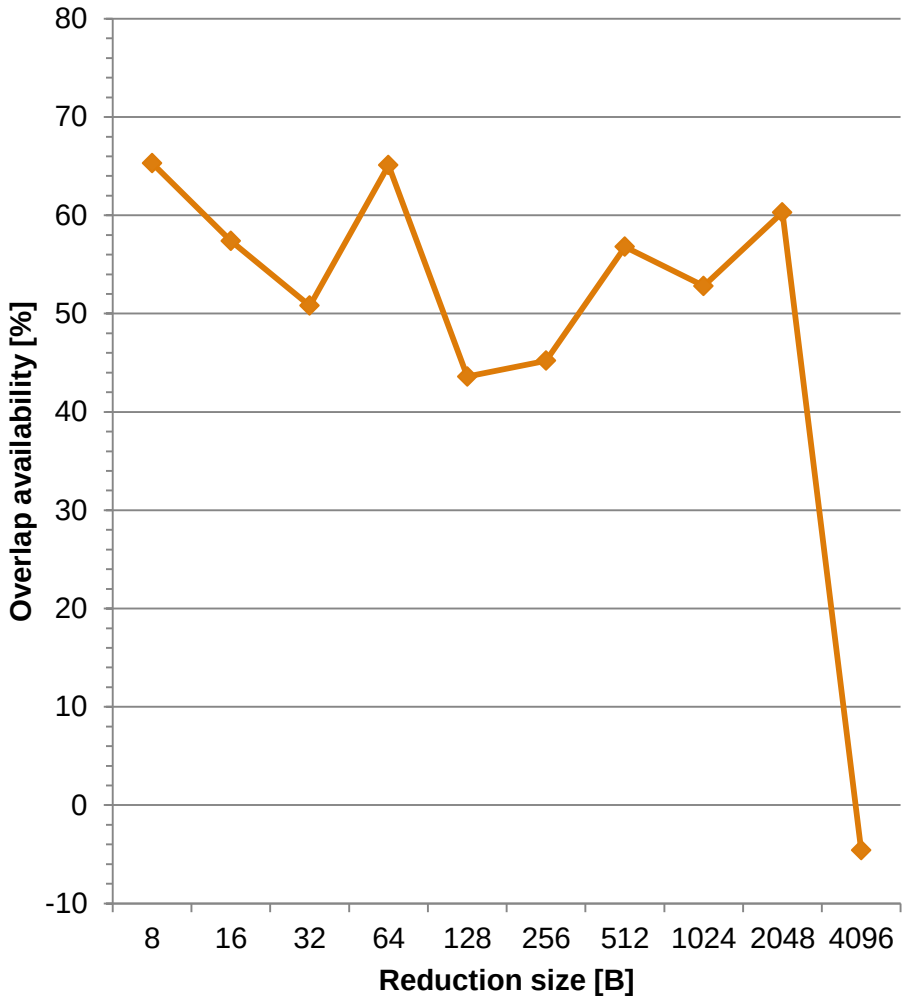
- MPI on XC behaves essentially the same as MPI on XE
  - uGNI interface is the same for XE and XC
  - MPICH3 code base is nearly the same
    - Messaging Paths (VSHORT, EAGER, RENDEZVOUS) are Identical
- Enhanced Features for XC
  - Modified MPI Asynchronous Progress Engine Threads
    - Threads can be placed on unused Intel hyper thread cores
  - XC Hardware Collective Engine (CE)
    - XC supports hardware-offload of Barrier & Allreduce collectives
    - Invoke these via MPICH_USE_DMAPP_COLL env variable
    - Must also link libdmapp into your application directly.

MPI - Async Progress Engine Support

- Used to improve communication/computation overlap
  - Each MPI rank starts a “helper thread” during MPI_Init
- Helper threads progress MPI engine while application computes
- Only inter-node messages that use Rendezvous Path are progressed (relies on BTE for data motion)
- To enable on XC when using 1 stream per core:
  - `export MPICH_NEMESIS_ASYNC_PROGRESS=1`
  - `export MPICH_MAX_THREAD_SAFETY=multiple`
  - `export MPICH_GNI_USE_UNASSIGNED_CPUS=enabled`
  - Run application: `aprun –n XX a.out`
- To enable on XC when using 2 streams per core recommend running with the corespec option:
  - `export MPICH_NEMESIS_ASYNC_PROGRESS=1`
  - `export MPICH_MAX_THREAD_SAFETY=multiple`
  - Run application with corespec: `aprun –n XX -r [1-2] a.out`
- 10% or more performance improvements with some apps

Non Blocking MPI (inc MPI-3)





Data courtesy of P. Manninen Cray Finland

CPU Specialisation

- Despite the low-noise nature of the XC30's CNL Linux OS it occasionally is necessary to run OS/kernel/daemon processes on CPUs.
- If all CPUs are in use then the OS must swap a user process out to execute the OS/kernel/deamon process.
- Normally this introduces only a small amount of noise to the application which evens out over the length of the run.
- However, there are certain pathological cases which amplify these delays if there are frequent synchronisations between nodes (e.g. collectives) preventing scaling.
- CPU specialisation reserves some CPUs for the OS/system/daemon tasks (like OS, MPI progress engines, daemons). This improves overall performance

CPU Specialisation (pt 2)

- On the XC30 the reserved CPU's are automatically chosen to be from any unused CPUs on Compute Units (e.g. spare Hyperthreads), even if "`-j1`" has been selected.

- Users specify how many CPUs to reserve by adding a "`-r <CPUs>`" flag to the aprun command. The sum total of "`-r`" and "`-N`" must not exceed 48 (the total number of CPUs on the node). E,g

```
aprun -n 1024 -N 24 -r 8 -j 1 a.out
```

- Required for use with MPI environment variables, `MPICH_GNI_USE_UNASSIGNED_CPUS` and `MPICH_NEMESIS_ASYNC_PROGRESS` flags.

Hyperthreading optimization chart

Single Stream Mode – No MPI Async:

Collect performance baseline here

Maximize per cpu performance

Little MPI communication overlap for medium size messages

Dual Stream Mode – Without MPI Async

Goals:
Optimizing per node performance
or
Maximizing performance by using many PEs

Is this "better"?

Single Stream Mode – With MPI Async

Goals:

Maximize per cpu performance

Improve communication performance

Does overall performance improve?

Dual Stream Mode – With MPI Async

Goals:
Optimizing per node perf. or
Maximizing perf. using many Pes
and
Improve communication performance…

But give up using 1 or 2 hyperthreads

Is this "better"?

How to get more help

- Support website: http://www.archer.ac.uk
- Cray XC30 Advanced Tools Workshop
  Edinburgh @ EPCC
  28th – 29th January 2014