

# BLUE WATERS

SUSTAINED PETASCALE COMPUTING

February 11, 2015

## Programming Environment and Running Jobs



GREAT LAKES CONSORTIUM  
FOR PETASCALE COMPUTATION

CRAY®

# Modules

- The user environment is controlled using the modules environment management system.
- The module utility helps you quickly identify software that is available on the system and makes it easier to modify your environment.
- List all available modules and versions:
  - module avail
- List all modules currently loaded
  - module list

## Modules (cont.)

- Modules may be loaded, unloaded, or swapped either on a command line or in \$HOME/.bashrc (.cshrc for csh ) shell startup file. E.g.
  - `module load craype-hugepages2M`
  - `Module load ddt`
  - `swap (different version or different compiler)`
    - `change module version:`
      - `module swap XYZ/1.0 XYZ/1.1`
    - `change compiler:`
      - **`module swap PrgEnv-cray PrgEnv-gnu`**  
(PrgEnv-\* modules are special; explained later)

## Using Modules In (bash) Job Scripts

- You must run the initialization command first:
  - **. /opt/modules/default/init/bash**
  - before any “module X Y” commands

## Default Module List Is Expected

- good commands to use:
  - module load XXX
  - module unload YYY (if it conflicts with something you need)
  - module swap
- Do NOT
  - **module purge**

(this makes a mess of your environment and makes it very difficult for us to help you)



## Module Versions

- Try to avoid version dependency
- System typically has:
  - default version of package
  - newer stable version of package
  - a couple of older versions
- older package versions are culled periodically
- If you DO have a version dependency, please let us know (submit a ticket: old default 1.2.3 works, new default 1.2.5 does not)

# Blue Waters Programming Environments

Four compiler sets available, managed by PrgEnv-\* module:

- Cray Programming Environment (default)
- PGI programming environment
- Gnu programming environment
- Intel programming environment

# Blue Waters Programming Environments

- Programming Environments managed through the module utility.
- Modules help ensure that your environment is always configured properly. Paths, libraries, etc, will be properly set by the chosen programming environment using module.
- Compiler wrappers:
  - ftn* (for fortran) *cc* (for C) and *CC* (for C++)
  - enable the use of desired compilers, and their corresponding include files, library paths etc.
- With PrgEnv-cray, for instance:
  - *cc* points to the Cray C compiler
  - *CC* points to the Cray C++ compiler
  - *ftn* points to the Cray Fortran compiler



# Use Wrapped Compilers (for MPI production multinode codes)

- Cross-compile environment (build on login, run on compute node)
- Use cc, CC, ftn
- do not use: gcc, gfortran (for instance)
- You'll do a lot of:
  - CC=cc
  - CXX=CC
  - MPICC=cc
- The big exception is nvcc for CUDA-C programming
  - (linking is done with CC)

## Using Wrapper Compilers

- Name is always cc,CC,ftn
- Options reflect underlying compiler (which PrgEnv-\* module that's loaded
  - For instance, -Wall (gcc option) works for cc when PrgEnv-gnu is loaded, not when PrgEnv-cray is
- “man cc” for general options
- “man gcc” for specific options

## Compiler Wrappers and Modules Work Together: (Especially for **experts**)

- Modules configure internal (invisible to the user) calls to `-l` and `-L` for *system* libraries within compiler wrappers
- you should not use `-L` and `-l` for system libraries (that would create path and thus version dependencies)
- you will typically use `-I` and `-L` and `-l` for your own, local libraries

# Compilers are NOT versioned by PrgEnv-\*

- With PrgEnv-cray loaded:  
Cray compiler version set by version of the cce module  
("Cray Compiler Environment")
- Prgenv-gnu: compiler versioned by "gcc" module
- PrgEnv-pgi: compiler versioned by "pgi" module
- PrgEnv-intel: compile versioned by "intel" module

# Programming Models

- MPI
- OpenMP
- Hybrid Programming: MPI + OpenMP



## MPI

- Compiling and linking is performed using wrapper scripts ftn, cc, and CC for source code written in Fortran, C, and C++, respectively.
  - Wrappers invoke the appropriate compiler based on the current Programming Environment
  - There is NO “mpicc” on Blue Waters. The compiler wrappers already have that functionality built in.
  - Wrappers automatically link in a wide variety of libraries as necessary, including MPI ( for instance, -lmpi is not required and will cause the link step to fail).

# OpenMP

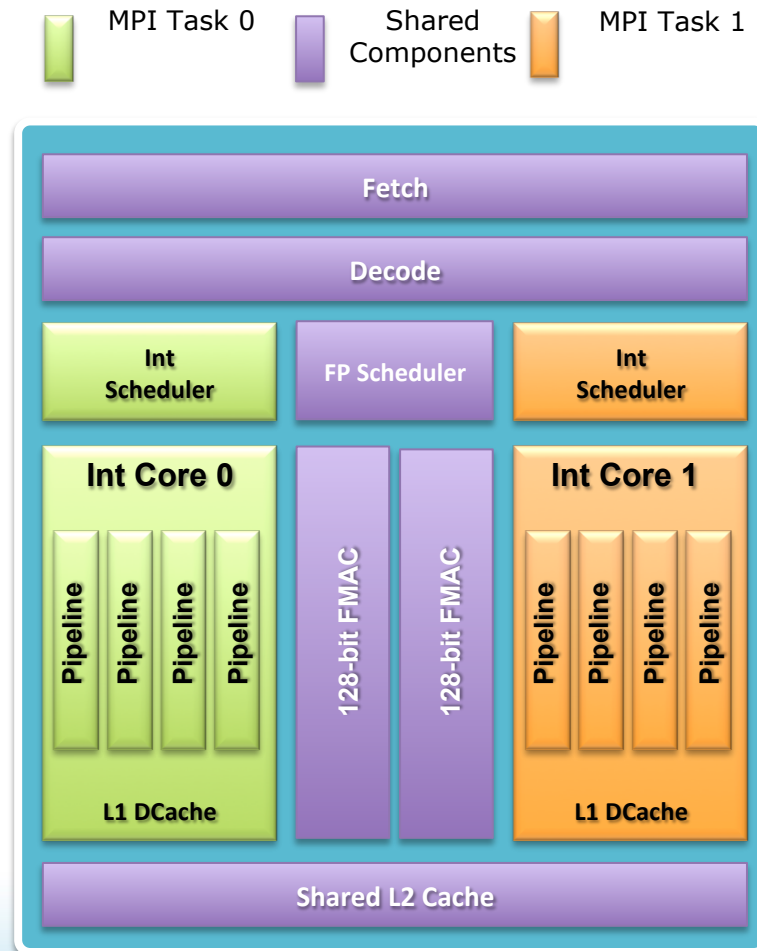
- a shared memory programming paradigm on the node
  - Cray compilers:
    - Default enabled: `-h thread2`
  - GNU compilers:
    - `-fopenmp`
  - PGI compilers:
    - `-mp`

## MPI+OpenMP

- MPI+OpenMP is an efficient way to exploit multicore processors on Blue Waters.
  - Each OpenMP thread typically runs on one compute core (i.e. maximum 32 on BW).
- Thread safety
  - Required to specify the desired level of thread support
  - set environment variable `MPICH_MAX_THREAD_SAFETY` to different values to increase the thread safety.
    - `MPI_THREAD_SINGLE` (default)
    - `MPI_THREAD_FUNNELED`
    - `MPI_THREAD_SERIALIZED`
    - `MPI_THREAD_MULTIPLE`

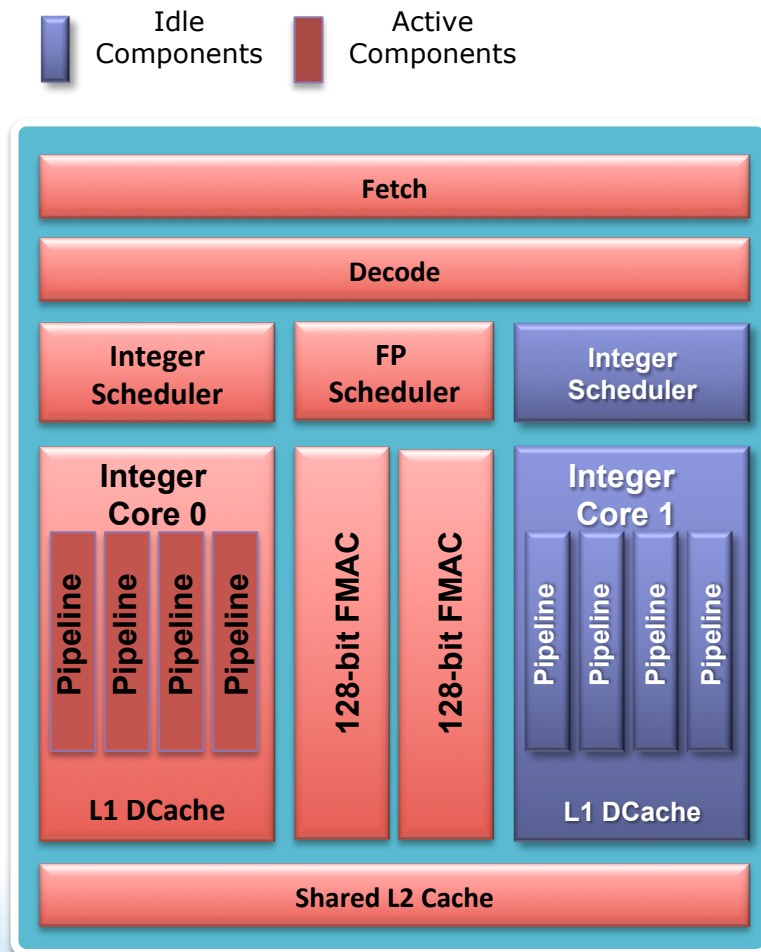
# Two MPI Tasks on a Compute Unit ("Dual-Stream Mode")

- An MPI task is pinned to each integer unit
  - Each integer unit has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit, instruction fetch, and the L2 Cache are shared between the two integer units
    - 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 a 2GB per task memory footprint
  - Code is not well vectorized



# One MPI Task on a Compute Unit ("Single Stream Mode")

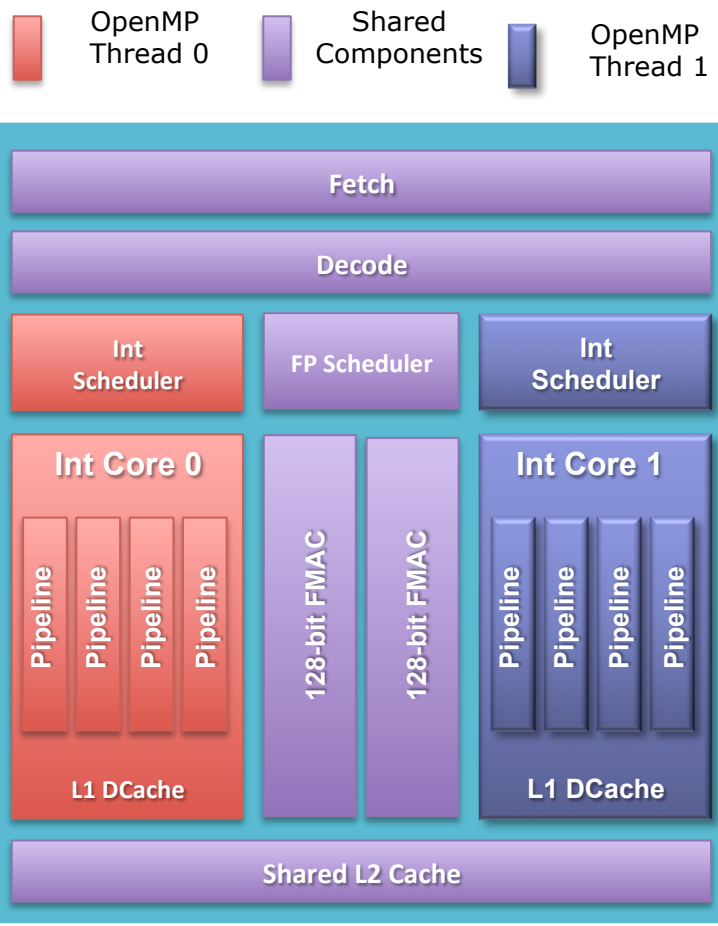
- Only one integer unit is used per compute unit
  - This unit has exclusive access to the 256-bit FP unit and is capable of 8 FP results per clock cycle
  - The unit has twice the memory capacity and memory bandwidth in this mode
  - The L2 cache is effectively twice as large
  - The peak of the chip is not reduced
- When to use
  - Code is highly vectorized and makes use of AVX instructions
  - Code benefits from higher per task memory size and bandwidth





# One MPI Task per compute unit with Two OpenMP Threads ("Dual-Stream Mode")

- An MPI task is pinned to a compute unit
- OpenMP is used to run a thread on each integer unit
  - 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 at each MPI rank



# Running in Dual or Single-Stream modes

- Dual-Stream mode is the current default mode. General use does not require any options. CPU affinity is set automatically by ALPS.
- Single-Stream mode is specified through the -j aprun option. Specifying -j 1 tells aprun to place 1 process or thread on each compute unit.
- When OpenMP threads are used, the -d option must be used to specify how many threads will be spawned per MPI process. See the aprun(1) man page for more details. The aprun -N option may be used to specify the number of MPI processes to assign per compute node or -S to specify the number of processes per Interlagos die. Also, the environment variable `$OMP_NUM_THREADS` needs to be set to the correct number of threads per process.
- For example, the following spawns 4 MPI processes, each with 8 threads, using 1 thread per compute unit.
  - `OMP_NUM_THREADS=8`
  - `aprun -n 4 -d 8 -j 1 ./a.out`

## NUMA Considerations

- Each Interlagos *socket* has 2 NUMA memory domains, each with 4 Bulldozer Modules. Access to memory located in a different NUMA domain (even within the same node) is slower than access to your own NUMA domain.
- XE nodes (2 sockets) have 4 NUMA domains, XK (1 socket) have 2
- OpenMP performance is usually better when all threads in a process execute in the same NUMA domain. For the Dual-Stream case, 8 CPUs share a NUMA domain, while in Single-Stream mode 4 CPUs share a NUMA domain. Using a larger number of OpenMP threads per MPI process than these values may result in lower performance due to cross-domain memory access.
- When running 1 process with threads over the NUMA domains, it's critical to initialize (not just allocate) memory from the thread that will use it in order to avoid NUMA side effects.

## Note: Cray aprun terminology

- “man aprun” for full manual and options for running parallel codes
- Cray calls an MPI rank a “PE” (“processing element”)
- The next slide(s) reflect Cray’s terminology

## Setting Process Affinity – aprun options

Common aprun options are:

- n: Number of processing elements PEs for the application
- N: Number of PEs to place per node
- S: Number of PEs to place per NUMA node.
- d: Number of CPU cores required for each PE and its threads
- cc: Binds PEs to CPU cores.
- r: Number of CPU cores to be used for core specialization
- j: Dual or single stream/integer cores to use for a PE
- ss: Enables strict memory containment per NUMA node

} must  
match #  
of nodes

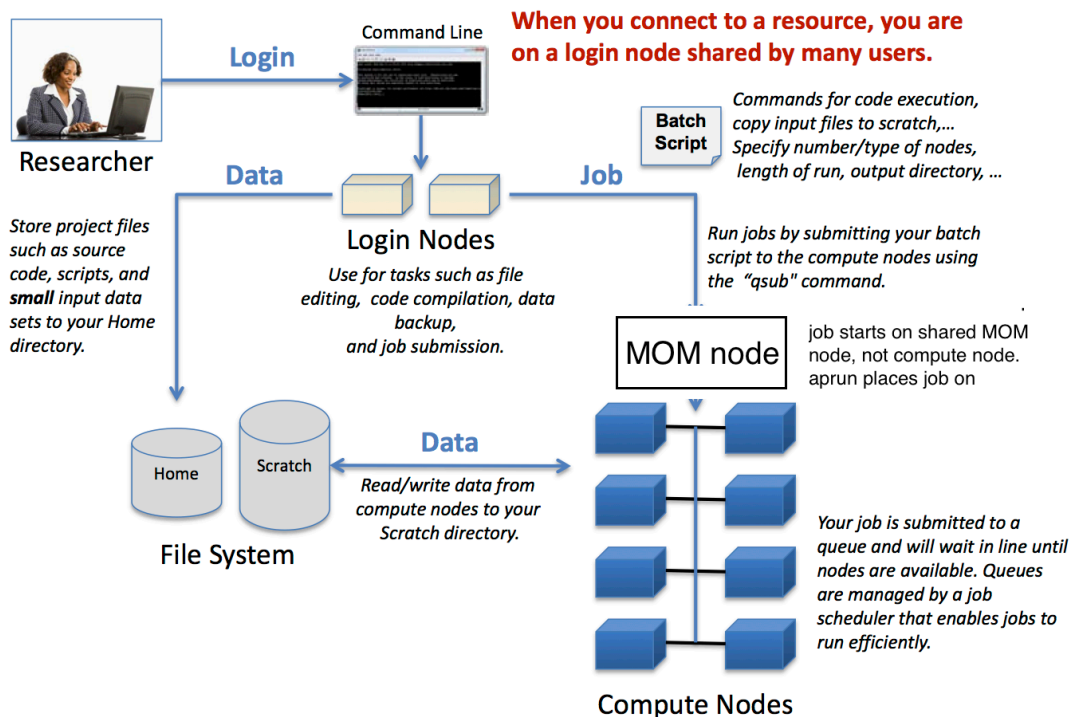
(see “man aprun” for full details)



# Running Jobs

- Job resource manager and scheduler provide resources.
- Like a traditional cluster the login nodes are shared.
- Unlike a traditional cluster MOM nodes are shared.
- aprun places application on compute nodes.
- Try to limit batch script execution on MOM node.

## Running Jobs Overview



## Running Jobs

- Example scripts

<https://bluewaters.ncsa.illinois.edu/batch-jobs>

- Interactive batch jobs

<https://bluewaters.ncsa.illinois.edu/interactive-jobs>

## User Info

- For additional information see the Blue Waters portal
- [New User Webinar material](#)
- [User Guide](#)