BLUE WATERS

SUSTAINED PETASCALE COMPUTING

February 11, 2015

Programming Environment

New User Webinar

Craig Steffen

NCSA

NSF

GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

CRAY

Programming Environment

Languages
- Fortran
- C
- C++
- Python
- UPC

Performance Analysis
- Cray Performance Monitoring and Analysis Tool
- PAPI
- PerfSuite
- Tau

Compilers
- Cray Compiling Environment (CCE)
- PGI
- GNU
- Intel

Debuggers
- Allinea DDT
- lgdb

Prog. Env.
- Eclipse
- Traditional

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

IO Libraries
- NetCDF
- HDF5
- ADIOS

Resource Manager
- Adaptive Computing

Visualization
- VisIt
- Paraview
- YT

Tools

Environment setup
- Modules

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

Data Transfer
- globus online
- HPSS

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

Cray Linux Environment (CLE)/SUSE Linux

Cray developed	3rd party packaging
Under development	NCSA supported
Licensed ISV SW	Cray added value to 3rd party

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-gnu PrgEnv-cray

    (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 programing 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
- Partitioned Global Address Space (PGAS) paradigm
  - CAF
  - UPC
- Charm++

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

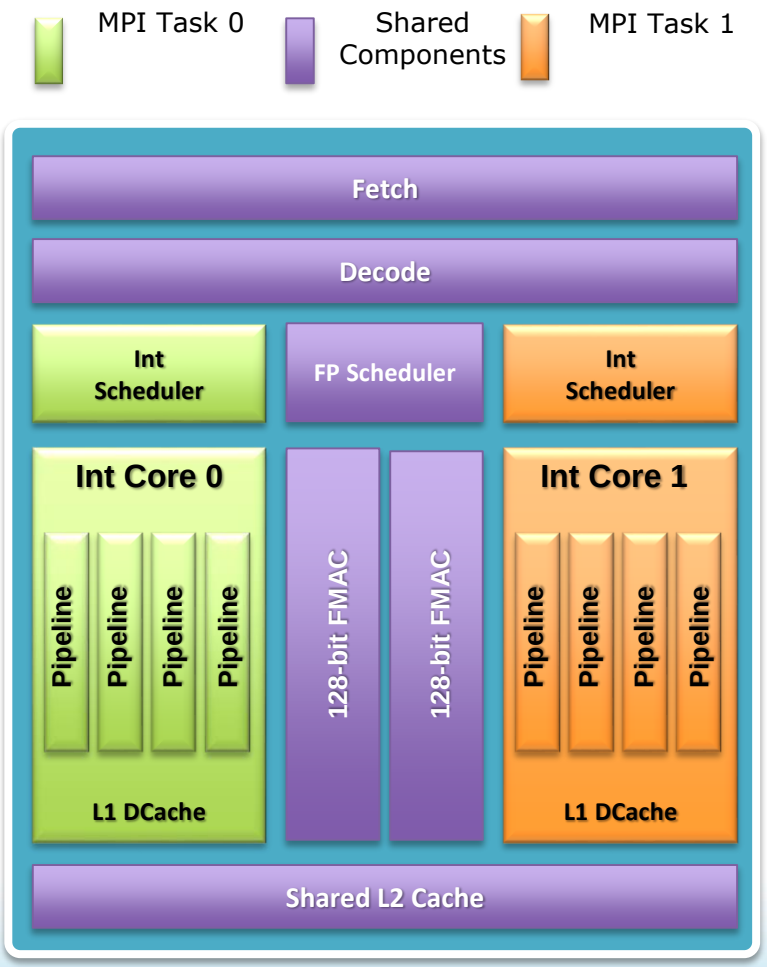
- a shared memory programming paradigm on the node
  - Cray compilers:
    - Default enabled: -h thread2
  - GNU compilers:
    - -fopenmp
  - PGI compilers:
    - -mp

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

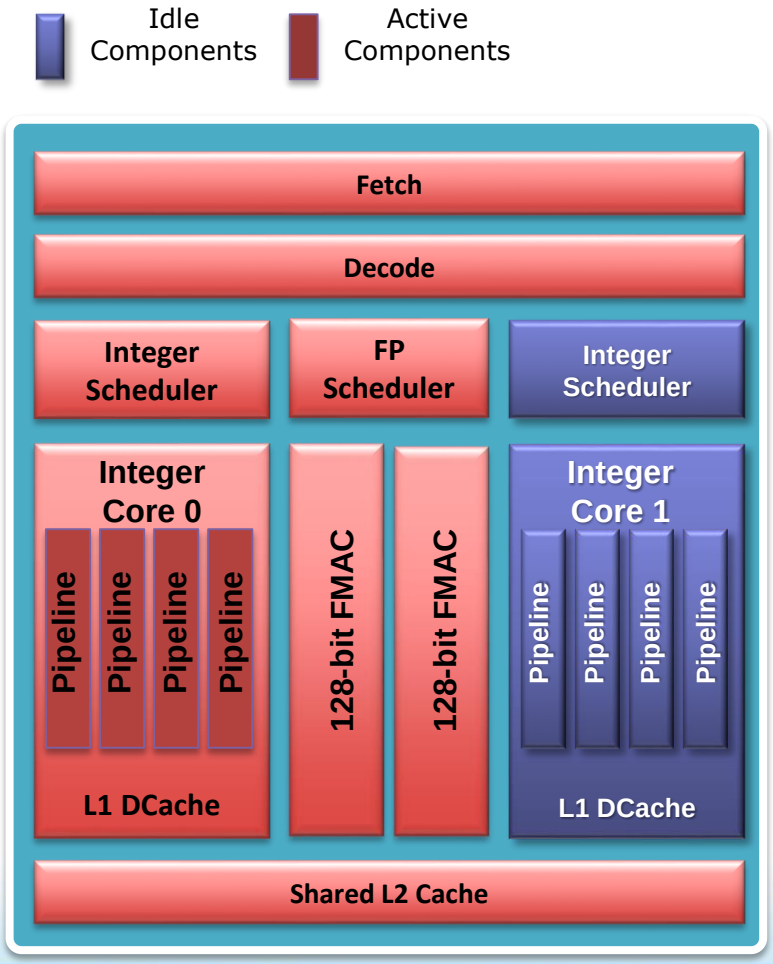
Two MPI Tasks on a Compute Unit ("Dual-Stream Mode")

- 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



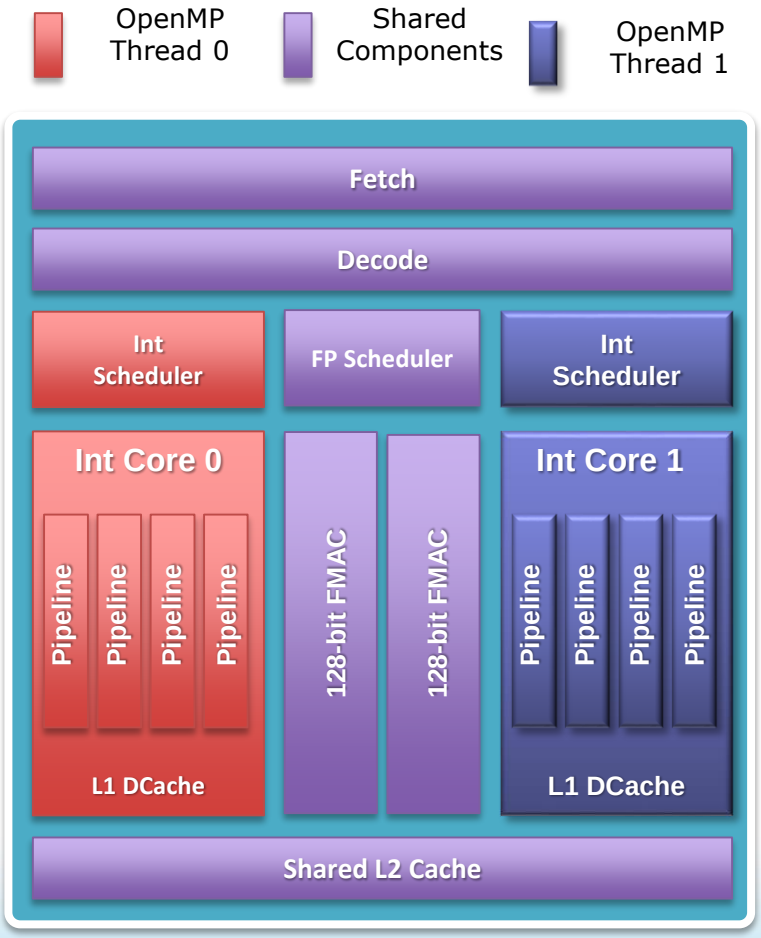
One MPI Task on a Compute Unit ("Single Stream Mode")

- 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.

PGAS

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

Coarray Fortran (CAF)

- Coarray Fortran is a small set of extensions to Fortran for Single Program Multiple Data (SPMD) parallel programming
  - included in the current standard (Fortran 2008).
- Cray Fortran: -h caf (on by default)
- Gfortran:
  - -fcoarray=<keyword>

UPC

- An extension of C that supports a single shared, partitioned global address space
- UPC is fully integrated into the Cray C compiler, to enable:
  - -h upc

Charm++

- Charm++ provides processor virtualization
  - Object oriented C++ programming
  - Migratable object-based dynamic load balancing
  - Fault tolerance and many other features
- To build Charm++ on BW
  - ./build charm++ gni-crayxe-hugepages-smp

Note: Cray aprun teminology

- “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

} must match # of nodes

- -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

(see “man aprun” for full details)

Blue Waters Debugging Tools

- **DDT** – A parallel debugger from Allinea Software, can be used for scalar, multi-threaded and large-scale parallel applications.
- **ATP** - Abnormal Termination Processing from Cray, a utility for debugging. If an application takes a system trap, ATP performs analysis on the dying application.
- **STAT** - The Stack Trace Analysis Tool gathers and merges stack traces from a parallel application's processes. The tool produces call graphs. STAT is also capable of gathering stack traces with more fine-grained information, such as the program counter or the source file and line number of each frame

Blue Waters Debugging Tool - DDT

- How to use:
  - Set up for x11 forwarding: ssh -Y bw.ncsa.illinois.edu
  - Complile with the –g option: e.g. ftn -g test.f90 -o test
  - Starting a DDT debugging section with one of the following:
    - submit a job through DDT
    - manually launch a program with DDT
    - attach DDT to a running program
    - start a debug session from inside an interactive job
- The first three begin by launching ddt using the commands:
  - Module load ddt
  - ddt
- More details in the ddt section on the portal

Blue Waters Debugging Tool - ATP

To use ATP for program abnormal terminations, do:

- Load atp module by ``module load atp''
- Recompile and link the code
- Modify job script as follows:

module load atp

export ATP_ENABLED=1 # or setenv ATP_ENABLED 1

aprun …

- Submit the job
- If the application terminates, use gdb or stat-view to examine the resulting diagnostic files
- (details on portal)

Debugging Tool: STAT

- Stack Trace Analysis
- (very sparse) page on portal
- If you need to use STAT, please contact help+bw@ncsa.Illinois.edu