Programming Environment

A Programming Environment (PrgEnv) is a set of related software components, such as compilers, scientific software libraries, implementations of parallel programming paradigms, batch job schedulers, and other third-party tools, all of which cooperate with each other.

Three programming environments are available on Blue Waters, namely the Cray Programming Environment, the PGI programming environment, and the Gnu programming environment. The programming environment is managed by the module command.

The Cray Programming Environment, including the Cray compiler suite, is loaded by default upon login, but it is a simple matter to change from one Programming Environment to another by using the module command.

Type module list to generate a display of all currently loaded software modules; the Programming...
All of this information is Available in the Blue Waters Portal:

- [https://bluewaters.ncsa.illinois.edu/programming](https://bluewaters.ncsa.illinois.edu/programming)
- Portal is at bluewaters.ncsa.illinois.edu
  - Documentation
  - User Guide
  - Programming
Languages and Programming Environments on Blue Waters

- Compilers/Programming Models
  - C/C++
  - Fortran 77, 90, 95, 2003, 2008
  - MPI
  - OpenMP
- Charm++
- PGAS Languages
  - Co-Array Fortran
  - UPC
- Accelerator Programming Environments (for XK nodes)
  - OpenACC
  - NVIDIA CUDA
  - (not officially supported) OpenCL (via PGI compilers)
Portal Will Soon Contain Full Table

**BW compiler capability table (draft for portal)**

*Added by Craig Stelfen, last edited by Craig Stelfen on Nov 12, 2013 (view change)*

<table>
<thead>
<tr>
<th>capability</th>
<th>Cray</th>
<th>PGI</th>
<th>Gnu</th>
<th>Intel</th>
<th>compiler name</th>
<th>source file ext</th>
<th>to enable:</th>
<th>required modules</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>cc</td>
<td>.c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>CC</td>
<td>.CC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran 77</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>fn</td>
<td>.F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran 90</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>fn</td>
<td>.F90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran 95</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>fn</td>
<td>.F95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran 2003</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>SOME FEATURES</td>
<td>fn</td>
<td>.F03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran 2008</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>SOME FEATURES</td>
<td>fn</td>
<td>.F08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI compilers</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td>automatic (compiler wrappers already have mpi-c equivalent functionality)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenMP</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td>-h omp (on by default)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td>-mmp=nonum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td>-fopenmp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td>-openmp</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Partitioned Global Address Space (PGAS) Languages**
### Partitioned Global Address Space (PGAS) Languages

<table>
<thead>
<tr>
<th></th>
<th>Cray</th>
<th>PGI</th>
<th>Gnu</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co-array Fortran</td>
<td><strong>YES</strong></td>
<td></td>
<td><strong>YES</strong></td>
<td>ftn automatic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(to disable use -tnocall)</td>
</tr>
<tr>
<td>Unified Parallel C</td>
<td><strong>YES</strong></td>
<td></td>
<td>SOME FEATURES</td>
<td>cc -h upc</td>
</tr>
</tbody>
</table>

### Accelerator-targeting Languages

<table>
<thead>
<tr>
<th></th>
<th>Cray</th>
<th>PGI</th>
<th>Gnu</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenACC</td>
<td><strong>YES</strong></td>
<td></td>
<td></td>
<td>PrgEnv-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>craype-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(including flags for large static memory)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ftn flags: -h acc, noomp -fpic -dynamic -G2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cc flags: -h pragma=acc -h nopragma=omp -fpic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-dynamic -Gp</td>
</tr>
<tr>
<td>NVIDIA CUDA</td>
<td><strong>YES</strong></td>
<td><strong>YES</strong></td>
<td></td>
<td>PrgEnv-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cudatool</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(including flags for large static memory)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ftn flags: -acc -ta=nvidia -cldart -mcode=medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cc flags: -acc -ta=nvidia -cldart -mcode=medium</td>
</tr>
<tr>
<td>OpenCL</td>
<td><strong>YES</strong></td>
<td></td>
<td></td>
<td>cudatool</td>
</tr>
<tr>
<td>(not officially supported on Blue Waters)</td>
<td>NOT OFFICIALLY</td>
<td></td>
<td></td>
<td>cudatool</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Cray</th>
<th>PGI</th>
<th>GNU</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>nvecc .cu</td>
<td>-gencode=arch=compute_35,code=compute_35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>nvecc .cu</td>
<td>-gencode=arch=compute_35,code=compute_35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-IOpenCL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cudatool</td>
</tr>
</tbody>
</table>

Presentation Title
Reminder: Cray Compiler Environment Uses COMPILER WRAPPERS

- cc (C)
- CC (C++)
- ftn (Fortran)
- NEVER use mpicc
- switch compiler types using modules:
  - PrgEnv-gnu
  - PrgEnv-pgi
  - PrgEnv-cray
Everything on Blue Waters is controlled by *modules*

- module available
- module load
- module swap

modules control loaded libraries, include files, other modules, and control behavior of compiler wrappers

“man module” for details
Details available in man page

• Man pages very detailed
• very specific to software package and modules loaded
• Maintained by Cray as current documentation
• Man pages specific to specific commands
  • “man ftn” has ~120 lines
  • “man crayftn” has ~2300 lines
Reminder: CROSS-Compiler Environment

• Standard Programming Paradigm:
  • Compile on login nodes
  • Run on compute nodes
• compiled executables sometimes (often) do weird things if you run them on login nodes
• To test codes, run interactive job and use aprun from command line
C compatibility
(compiler wrapper: “cc” (lowercase) )

• ANSI C
  • Except… (accepts // comments, for instance)
Fortran (compiler wrapper: “ftn”)

- source language controlled by filename extension
  - .f .F (Fortran 77)
  - .f90 .F90 (Fortran 90)
  - .f95 .F95 (Fortran 95)
  - .f03 .F03 (Fortran 2003)
  - .f08 .F08 (Fortran 2008)
- .F* files invoke pre-processor before compiling, .f* files do not preprocess
Compiler Versioning

• Modules that switch between compiler flavors:
  • PrgEnv-cray
  • PrgEnv-pgi
  • PrgEnv-gnu
• However, those do not switch between compiler versions; those must be switched with their own modules
Cray Compiler Environment modules (switch versions of Cray cc, CC, ftn)

- cce/8.1.3
- cce/8.1.4
- cce/8.1.5
- cce/8.1.6
- cce/8.1.7
- cce/8.1.8
- cce/8.1.9 (default)
- cce/8.2.0
- cce/8.2.1
Message Passing Interface

- Paralllellization by launching multiple MPI “rank”s that are all aware of each other
- Multiple rank launch by job system (aprun on Cray)
- Data parallelism at code level
- Communication at code level
Message Passing Interface (library)
ALL Compilers Have MPI built-in

- Book-keeping
  - MPI_Init
  - MPI_Comm_rank
- Single-point Message Sending
  - MPI_Send
  - MPI_Recv
- Collective Operations
  - MPI_Bcast
  - MPI_Allreduce
What MPI Code Looks like

- `#include <stdio.h>
  #include <mpi.h>`

  ```c
  int main (argc, argv)
  int argc;
  char *argv[];
  {
  int rank, size;

  MPI_Init (&argc, &argv); /* starts MPI */
  MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* get current process id */
  MPI_Comm_size (MPI_COMM_WORLD, &size); /* get number of processes */
  printf( "Hello world from process %d of %d\n", rank, size );
  MPI_Finalize();
  return 0;
  }
  ```
Open Message Passing Overview

- Parallelization of individual loops/code blocks
- Controlled in code by compiler directives
- Parallelization enabled at compile time, extent controlled at runtime
- Data parallelism and communication among OMP threads controller by compiler
- MPI + OMP is called “hybrid”
What OMP Code Looks like

• `#pragma omp parallel num_threads(2)`
• {
  • if (omp_get_thread_num() == 0)
  • {
  • /* Write to the data buffer that will be read by thread */
  • data = 42;
  • /* Flush data to thread 1 and strictly order the write to data relative to the write to the flag */
  • `#pragma omp flush(flag, data)`
  • /* Set flag to release thread 1 */
  • flag = 1;
  • /* Flush flag to ensure that thread 1 sees the change */
  • `#pragma omp flush(flag)`
  • }
}
Invoking OMP Directives in Compilers

- Cray (on by default)
  - -h omp
- PGI
  - -mp=nonuma
- Gnu
  - -fopenmp
Charm++ Overview

- http://charm.cs.illinois.edu
- Execution parallelism by programming independent software “chare” objects that communicate with each other; the main chare starts application like main() in C
- Execution Parallelism controlled at runtime by the Charm++ runtime system
- Data parallelism and communication in code
Invoking Charm++ on Blue Waters

- Download Charm++
- Build it against uGNI communication library OR against Cray MPI
  - `./build charm++ gemini_gni_crayxe` OR
  - `./build charm++ mpi-crayxe`
- Compile your Charm++ code with the resulting `charmcc`
Charm++ example

- extern /* readonly */ CProxy_Main mainProxy;
- extern /* readonly */ int numElements;

Hello::Hello() {
  // Nothing to do when the Hello chare object is created.
}

Hello::Hello(CkMigrateMessage *msg) {}
PGAS Languages

- Partitioned Global Address Space Languages
- Co-Array Fortran (CAF)
- Unified Parallel C (UPC)
- Execution parallelism via aprun at runtime
- Data parallelism by allocation scheme within compiler
Co-Array Fortran Example

program hello
    implicit none

    integer        :: myrank, numprocs

    myrank   = this_image()
    numprocs = num_images()

    if (myrank == 1) print *, 'Hello world'
    print *, 'I am image number',myrank,' of ',numprocs
end program hello
UPC Example

- #include <stdio.h>
  #include <upc.h>

  int main(int argc, char** argv)
  {
    if (MYTHREAD == 0) printf("hello world\n");
    printf("I am thread number %d of %d threads\n", MYTHREAD, THREADS);
    return 0;
  }
Invoking PGAS Compilers

- Cray only (have to have loaded PrgEnv-cray)
- CAF: turned on automatically in Cray
  [to disable –h nocaf]
- UPC:
  cc -h upc
Open-Accelerator (OpenACC)

- Code written in C, Fortran
- Loops and code blocks to run on the accelerator are designated by compiler directives
- openacc.org
- https://bluewaters.ncsa.illinois.edu/openacc
- Available using Cray or PGI compiler stack
OpenACC on Blue Waters

• *NCSA recommends using only one programming environment (Cray/Fortran) for your code development and testing.* At this time, the OpenACC standard is not yet mature enough to ensure complete portability between the Cray and PGI compilers.

• Note that combining OpenACC directives within OpenMP regions will result in runtime errors with the Cray programming environment so it's best to disable OpenMP when getting started with the Cray compilers.
OpenACC Example

• `#pragma acc data copy(A), create(Anew)`
  • `while ( error > tol && iter < iter_max )`
  • `{`
    • `error = 0.0;`
  • }

• `#pragma acc kernels`
  • `for( int j = 1; j < n-1; j++)`
  • `{`
    • `for( int i = 1; i < m-1; i++)`
    • `{`
      • `Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]`
        • `+ A[j-1][i] + A[j+1][i]);`
      • `error = fmax( error, fabs(Anew[j][i] - A[j][i]));`
    • `}`
  • `}`
Example List of OpenACC Directives

- `#pragma acc parallel` [C]
- `!$acc parallel` [Fortran]

- `#pragma acc loop`
  - `collapse ( n )`
  - `seq`
  - `tile ( expression-list )`
Invoking OpenACC (in Cray)

- modules loaded
  - PrgEnv-cr
  - craype-accel-nvidia35
- ftn -h acc,noomp -fpic -dynamic -G2
- cc -h pragma=acc -h noppragma=omp -fpic -dynamic -Gpb
Invoking OpenACC (in PGI)

• Modules loaded:
  • PrgEnv-pgi
  • cudatoolkit
• `ftn -acc -ta=nvidia -lcudart -mcmodel=medium`
• `cc -acc -ta=nvidia -lcudart -mcmodel=medium`
NVIDIA CUDA Overview

• Required module: cudatoolkit
• Code explicitly divided between functions to run on CPU and functions to run on Accelerator
• Execution parallelism explicit, but may be configured at runtime
• Communication (function hand-offs) in code
C (on CPU) calling CUDA function (on GPU)

- // No MPI here, only CUDA
- void computeGPU(float *hostData, int blockSize, int gridSize)
- {
-     int dataSize = blockSize * gridSize;
-     // Allocate data on GPU memory
-     float *deviceInputData = NULL;
-     CUDA_CHECK(cudaMalloc((void **)&deviceInputData, dataSize * sizeof (float)));
-     float *deviceOutputData = NULL;
-     CUDA_CHECK(cudaMalloc((void **)&deviceOutputData, dataSize * sizeof (float)));
-     // Copy to GPU memory
-     CUDA_CHECK(cudaMemcpy(deviceInputData, hostData, dataSize * sizeof (float), cudaMemcpyHostToDevice));
-     // Run kernel
-     simpleMPIKernel <<<gridSize, blockSize>>> (deviceInputData, deviceOutputData);
What CUDA Code Looks Like

- `__global__` void simpleMPIKernel(float *input, float *output)
  - {
    - int tid = blockIdx.x * blockDim.x + threadIdx.x;
    - output[tid] = sqrt(input[tid]);
  - }

- Memory within the CUDA function is controlled explicitly
Using nvcc

• The NVIDIA nvcc is an exception to the compiler wrappers rule—use it separately

compile step:

dvcc -c
  -gencode=arch=compute_35,code=compute_35
  -o simpleMPIcuda.o simpleMPI.cu

link step:

CC -o simpleMPI simpleMPI.cpp
   simpleMPIcuda.o
Cray libsci_acc
(applies to OpenACC and CUDA)

- Cray has optimized OpenACC libraries for linking
- required PrgEnv-cray
- module load craype-accel-nvidia35
- man intro_libsci_acc