TSCC Bootcamp: Introduction to Accessing and Running Jobs on the TSCC System

Compiling & Linking

By: Mary Thomas

TSCC Compilers

Package	Topic Area	License Type	Package Home Page	User Install Location	Installed on: (L)ogin, (C)ompute, (B)oth
CMake	Cross Platform Makefile Generator	Open	CMake Home Page	/opt/cmake	B
GNU Compilers	C and Fortran Compilers	CentOS Core License	GNU Compiler Collection Home Page	/usr/bin/gcc –and– /usr/bin/gfortran	B
Intel Compilers	C and Fortran Compilers	Licensed (flexlm)	Intel Compilers Home Page	/opt/intel	B
Java	Compiler	Open	Java Home Page	/usr/bin/javac	B
PGI Compilers	C and Fortran Compilers	Licensed (flexlm)	PGI Compilers Home Page	/opt/pgi	B
UPC (Unified Parallel C) Compiler	Parallel Computing	BSD	UPC Home Page	/opt/upc	B

Porting Existing Codes

- Copy your application source files to $HOME directory or to a Data Oasis area for your account
  - /oasis/tscc/scratch/<*username*> where <*username*> is your TSCC login name.
- Data Oasis storage is not backed up
  - files stored on this system may be lost or destroyed
  - $HOME directory or project storage are permanent.
- Never run parallel or large serial jobs from the command line – they should always be run using the queue.

Compilers

- **TSCC compute nodes support several parallel programming models:**
  - **MPI**: Default Intel Compiler and openmpi_ibIntel MPI
  - **OpenMP**:
    - All compilers (GNU, Intel, PGI) have OpenMP flags.
  - **GPU** nodes: support CUDA, OpenACC.
  - **Hybrid** modes are possible (see examples below).
- **See:**
  - https://www.sdsc.edu/support/user_guides/tscc.html#compiling
  - https://www.sdsc.edu/support/user_guides/tscc-quick-start.html#gpu-queue

Compiling MPI Codes

- MPI source code should be compiled for TSCC using these commands:
  - mpicc [options] file.c (C and C++) myrinet/mx switch and the Portland Compiler
  - mpif77 [options] file.f (Fortran 77 source code) myrinet/mx switch and the Portland Compiler
  - mpif90 [options] file.f (free format code/dynamic memory allocation/object oriented Fortran source code) openmpi and the Intel compiler

Compiling Serial Code

- Serial source code should be recompiled
- Portland Group Compilers
  - pgcc [options] file.c (C and C++)
  - pgf77 [options] file.f (fixed form Fortran source code)
  - pgf90 [options] file.f90 (free format Fortran source code)
- Intel Compilers
  - icc [options] file.c (C and C++)
  - ifort [options] file.f (fixed form Fortran source code)
  - ifort [options] file.f90 (free format Fortran source code)
- Gnu Compilers
  - gcc [options] file.c (C and C++)
  - gfortran [options] file.f90 (free format Fortran source code)

To Access Infiniband

- PGI compilers + openmpi_ib
- PGI compilers + mvapich2_ib
- Intel compilers + openmpi_ib
- Intel compilers + mvapich2_ib
- GNU compilers + openmpi_ib
- GNU compilers + mvapich2_ib

Compatibility Options for Fortran

-Mcpp	run the Fortran preprocessor on source files prior to compilation
-Dname[=value]	specify name as a definition to use with conditional compilation directives or the Fortran preprocessor (-Mcpp)
-byteswapio	swap bytes from big-endian to little-endian or vice-versa for unformatted files
-i8	set size of INTEGER and LOGICAL variables to 8 bytes
-i4	set size of INTEGER and LOGICAL variables to 4 bytes
-i2	set size of INTEGER variables to 2 bytes
-r8	treat REAL and CMPLX types as REAL*8 and DCMPLX
-Msave	save all local variables between calls (static allocation)

Detecting Programming Errors

- -g produce symbolic debug information in object file (implies -O0); required for debugging with DDT
- -C/-Mbounds array bounds checking
- -kTrap=(option, option...) specifies behavior on floating point exceptions (used only for main program):
  - dnorm trap on denormalized (very, very small) operands
  - divz trap on divide by zero
  - fp trap on floating point exceptions
  - inexact trap on inexact result
  - inv trap on invalid operands
  - none (default) disables all traps
  - ovf trap on floating point overflow
  - unf trap on floating point underflow
- -traceback add debug information

Optimization levels of Portland Group compilers

- -O0 No optimization
- -O1 (default) Task scheduling within extended basic blocks.
  - Some register allocation; no global optimizations.
- -O2 all level 1 optimizations and global scalar optimizations
  - optimization over all blocks
- -O3/O4 all level 1 and 2 optimizations + more aggressive optimizations.
- -Mflushz flush very, very small values to zero.

PGI Optimization: use -fast flag

- Portland Compiler User Guide recommends the use of the ***-fast*** flag;
- Option is host dependent and has the following default config:
  - set optimization level at -O2
  - unroll loops (-Munroll=c:1)
  - do not generate code to set up a stack frame pointer for every function (-Mnoframe)
  - enable loop redundancy elimination

Using Numerical Libraries: PGI

- PGI includes Optimized ACML library (LAPACK/BLAS/FFT).

- To link to the libraries:

```
pg90/pgf77 myprog.f -llapack –lblas
```

Using Numerical Libraries: Intel

- Intel developed the Math Kernel Library (MKL)
- Linear algebra, FFT and other numerical routines.
  - Basic linear algebra subprograms (BLAS) + sparse routines
  - Fast Fourier Transforms (FFT) in 1 and 2 dimensions, complex and real
  - The linear algebra package, LAPACK
  - A C interface to BLAS
  - Vector Math Library (VML)
  - Vector Statistical Library (VSL)
  - Multi-dimensional Discrete Fourier Transforms (DFTs)
- Documentation available in HTML and PDF format:
  - ${MKL_ROOT}/../Documentation.

Example: Loading the Intel Compilers

- The Intel compilers and the MVAPICH2 MPI implementation will be loaded by default.
- If you have modified your environment, you can reload by executing the module purge & load commands at the Linux prompt, or placing the load commands in your startup file (`~/.cshrc or ~/.bashrc`)

```
$ module purge
$ module list
No Modulefiles Currently Loaded.
$ module load gnutools
$ module list
Currently Loaded Modulefiles:
  1) gnutools/xx
$ module load intel mvapich2_ib
$ module list
Currently Loaded Modulefiles:
  1) gnutools/xx          2) intel/xx      3) mvapich2_ib/xx
```

Using the PGI Compiler

- PGI (formerly The Portland Group, Inc.), was a company that produced a set of commercially available Fortran, C and C++ compilers for high-performance computing systems.
- It is now owned by NVIDIA.
- PGI compilers can be loaded by executing the following commands at the Linux prompt or placing in your startup file (~/.cshrc or ~/.bashrc).
- For AVX support, compile with -fast

```
[$] module purge
[$] module load gnutools
[$] module load pgi
[$] module list
Currently Loaded Modulefiles:
  1) gnutools/xx    2) pgi/xx
```

- For more information on the PGI compilers run: **man [pgf90 | pgcc | pgCC]**

Parallel Code: MPI Hello World

Change to the MPI examples directory:

```
!  Fortran example
   program hello
   include 'mpif.h'
   integer rank, size, ierror, tag, status(MPI_STATUS_SIZE)

   call MPI_INIT(ierror)
   call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
   call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
   print*, 'node', rank, ': Hello and Welcome to Webinar
Participants!'
   call MPI_FINALIZE(ierror)
   end
```

Compile the code using :

```
mpif90 -o hello_mpi hello_mpi.f90
```

OpenMP Hello World

```
[PROGRAM OMPHELLO
      INTEGER TNUMBER
      INTEGER OMP_GET_THREAD_NUM

!$OMP PARALLEL DEFAULT(PRIVATE)
      TNUMBER = OMP_GET_THREAD_NUM()
      PRINT *, 'Hello from Thread Number[',TNUMBER,'] and Welcome Bootcampers!'
!$OMP END PARALLEL

      STOP
      END
```

Check the environment

```
[$] module list
Currently Loaded Modulefiles:
  1) gnutools/2.69          2) pgi.x
```

Compile the code

```
[$] mpif90 –mp   -o hello_openmp hello_openmp.f90
```

Hybrid MPI + OpenMP Hello World

```
#include <stdio.h>
#include "mpi.h"
#include <omp.h>

int main(int argc, char *argv[]) {
  int numprocs, rank, namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int iam = 0, np = 1;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name(processor_name, &namelen);

  #pragma omp parallel default(shared) private(iam, np)
  {
    np = omp_get_num_threads();
    iam = omp_get_thread_num();
    printf("Hello Webinar particpants from thread %d out of %d from process %d out of %d on %s\n",
           iam, np, rank, numprocs, processor_name);
  }

  MPI_Finalize();
}
```

Compile the code using :

```
Mpif90 –mp -o hello_mpi hello_mpi.f90
```

GPU COMPILATION

GPU/CUDA: Compile

- Set the environment in the same way you set it for PGI or Intel
- Then compile the code
- Example below is for matrix multiplication code on another HPC machine

```
[$] module purge
[$] which nvcc
/usr/bin/which: no nvcc in (/usr/lib64/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/opt/sdsc/bin:/o
pt/sdsc/sbin:/opt/ibutils/bin:/usr/java/latest/bin:/opt/pdsh/bin:/opt/rocks/bin:/opt/
rocks/sbin:/home/user/bin)
[$] module load cuda
[$] which nvcc
/usr/local/cuda-7.0/bin/nvcc
[$] nvcc -o matmul -I.  matrixMul.cu
```

GPU/CUDA: check node for GPU card

Note: you will be able to compile GPU code on the login nodes, but they will not run. To see if your node has GPU hardware, run *lspci.* Typically, login nodes do not have GPU hardware.

```
[$] lspci | grep VGA
09:00.0 VGA compatible controller: ASPEED Technology, Inc. ASPEED Graphics Family
(rev 30)
```

If the node ***does*** have a GPU card, you will see output similar to the following:

```
[$] ssh gpu-hotel "/sbin/lspci | grep VGA"
01:00.0 VGA compatible controller: NVIDIA Corp.. NV44 [GeForce 6200 LE] (rev a1)
02:00.0 VGA compatible controller: NVIDIA Corp.. GF100 [GeForce GTX 480] (rev a3)
03:00.0 VGA compatible controller: NVIDIA Corp.. GF100 [GeForce GTX 480] (rev a3)
```

GPU/CUDA: Check Environment!

obtaining device information: enum_gpu.cu (1)

```
#include "../common/book.h"
int main( void ) {
    cudaDeviceProp  prop;
    int count;
    HANDLE_ERROR( cudaGetDeviceCount( &count ) );
    for (int i=0; i< count; i++) {
        HANDLE_ERROR( cudaGetDeviceProperties( &prop, i ) );
        printf( "   --- General Information for device %d ---\n", i );
        printf( "Name:  %s\n", prop.name );
        printf( "Compute capability:  %d.%d\n", prop.major, prop.minor );
        printf( "Clock rate:  %d\n", prop.clockRate );
        printf( "Device copy overlap:  " );
        if (prop.deviceOverlap)
            printf( "Enabled\n" );
        else
            printf( "Disabled\n");
        printf( "Kernel execution timeout :  " );
        if (prop.kernelExecTimeoutEnabled)
            printf( "Enabled\n" );
        else
            printf( "Disabled\n" );
```

GPU/CUDA: Check Environment!

obtaining device information: enum_gpu.cu (2)

```
        printf( "   --- Memory Information for device %d ---\n", i );
        printf( "Total global mem:  %ld\n", prop.totalGlobalMem );
        printf( "Total constant Mem:  %ld\n", prop.totalConstMem );
        printf( "Max mem pitch:  %ld\n", prop.memPitch );
        printf( "Texture Alignment:  %ld\n", prop.textureAlignment );
        printf( "   --- MP Information for device %d ---\n", i );
        printf( "Multiprocessor count:  %d\n",
                    prop.multiProcessorCount );
        printf( "Shared mem per mp:  %ld\n", prop.sharedMemPerBlock );
        printf( "Registers per mp:  %d\n", prop.regsPerBlock );
        printf( "Threads in warp:  %d\n", prop.warpSize );

        printf( "Max threads per block:  %d\n",
                    prop.maxThreadsPerBlock );
        printf( "Max thread dimensions:  (%d, %d, %d)\n",
                    prop.maxThreadsDim[0], prop.maxThreadsDim[1],
                    prop.maxThreadsDim[2] );
        printf( "Max grid dimensions:  (%d, %d, %d)\n",
                    prop.maxGridSize[0], prop.maxGridSize[1],
                    prop.maxGridSize[2] );
        printf( "\n" );
        }
}
```

GPU/CUDA: Check Environment*

```
----------------------------------------------
--- General Information for device 0 ---
Name:  Tesla C1060
Compute capability:  1.3
Clock rate:  1296000
Device copy overlap:  Enabled
Kernel execution timeout :  Disabled
--- Memory Information for device 0 ---
Total global mem:  4294770688
Total constant Mem:  65536
Max mem pitch:  2147483647
Texture Alignment:  256
--- MP Information for device 0 ---
Multiprocessor count:  30
Shared mem per mp:  16384
Registers per mp:  16384
Threads in warp:  32
Max threads per block:  512
Max thread dimensions:  (512, 512, 64)
Max grid dimensions:  (65535, 65535, 1)
----------------------------------------------
--- General Information for device 2 ---
Name:  GeForce GT 240
Compute capability:  1.2
Clock rate:  1340000
Device copy overlap:  Enabled
Kernel execution timeout :  Disabled
--- Memory Information for device 2 ---
Total global mem:  1073020928
Total constant Mem:  65536
Max mem pitch:  2147483647
Texture Alignment:  256
```

```
----------------------------------------------
--- General Information for device 1 ---
Name:  Tesla C1060
Compute capability:  1.3
Clock rate:  1296000
Device copy overlap:  Enabled
Kernel execution timeout :  Disabled
   --- Memory Information for device 1 ---
Total global mem:  4294770688
Total constant Mem:  65536
Max mem pitch:  2147483647
Texture Alignment:  256
   --- MP Information for device 1 ---
Multiprocessor count:  30
Shared mem per mp:  16384
Registers per mp:  16384
Threads in warp:  32
Max threads per block:  512
Max thread dimensions:  (512, 512, 64)
Max grid dimensions:  (65535, 65535, 1)
----------------------------------------------
   --- MP Information for device 2 ---
Multiprocessor count:  12
Shared mem per mp:  16384
Registers per mp:  16384
Threads in warp:  32
Max threads per block:  512
Max thread dimensions:  (512, 512, 64)
Max grid dimensions:  (65535, 65535, 1)
```

**Data above is typical of GPUs, but not necessarily those on TSCC resrources*