Compilation Guide

- Overview
- GNU Compilers
- Cray Compilers
- Recommended Optimization Options

Overview

PLEASE NOTE: Cross Compiling Environment

You are compiling on a Linux login node but generating an executable for a CLE compute node.

All applications that will run in parallel on the Cray XE6 should be compiled with the standard language wrappers. The compiler drivers for each language are:

- cc wrapper around the C compiler
- CC wrapper around the C++ compiler
- ftn wrapper around the Fortran compiler

These scripts will choose the required compiler version, target architecture options, scientific libraries and their include files automatically from the module environment. The wrapper scripts, will create a highly optimized executable tuned for the compute nodes.

cc, CC, ftn are wrappers. They guarantee that codes compiled using the wrappers are prepared for running on the compute nodes.

The scripts choose which compiler (crayftn, craycc, crayCC, gfortran, gcc, g++, etc.) to use from the PrgEnv module loaded. The Cray compilers are the default compilers.

Unless you have a reason to, do not use gcc, gfortran, pgcc directly. Those commands will produce an executable for the login nodes! Login nodes do not support running distributed memory applications.

Cray compiler wrappers (ftn, cc and CC) must be used for building an executable, instead of native compiler commands (pgf90, pgcc, pgCC, gfortran, gcc, g++, etc.) since pat_build cannot build an instrumented executable from an executable built with a native compiler.

For example, cc- Invokes the C compiler, regardless of which compiler module is currently loaded. This command links in the fundamental header files and libraries required in order to produce code that can be executed on the Cray compute nodes. For more information, see the cc(1) man page.

GNU Compilers

To make this compiler the one used by the ftn, cc, and CC compiler wrappers, type in the following:

```
module swap PrgEnv-cray PrgEnv-gnu
```

<table>
<thead>
<tr>
<th>NON-MPI</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran 90 or 95</td>
<td>gfortran or gf90 ftn</td>
</tr>
<tr>
<td>Fortran 77</td>
<td>g77 ftn</td>
</tr>
<tr>
<td>C</td>
<td>gcc cc</td>
</tr>
<tr>
<td>C++</td>
<td>g++ CC</td>
</tr>
</tbody>
</table>

Recommended optimization options

The -Ofast option is appropriate with all PGI compilers:

**OpenMP**

Add this flag to any of the above:

```
-Oopenmp
```

If you want to use **cross compiler** do something like this:

```
ftn -Oopenmp code.f90 -o code
```

If you want to compile on compute nodes do something like this:

```
ftn -Oopenmp md.f90 -o md -static
```

or

```
gfortran -Oopenmp md.f90 -o md -static
```
Cray compilers

Cray compiler is default on Beagle2.

The Cray compilers are available on Beagle2 are they are the default compilers.

<table>
<thead>
<tr>
<th></th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran 90 or 95</td>
<td>ftn</td>
</tr>
<tr>
<td>Fortran 77</td>
<td>ftn</td>
</tr>
<tr>
<td>C</td>
<td>cc</td>
</tr>
<tr>
<td>C++</td>
<td>CC</td>
</tr>
</tbody>
</table>

Use `-hdisplay_opt` to see what are the default options of the compiler.

```
ams@login5:/lustre/beagle2/ams/open/XE6-beagle-april-2011/openmp> ftn -hdisplay_opt -h omp md.f90 -o md_new
Options: -O cache2,fp2,scalar2,thread2,vector2,mpi0,modinline,ipa3,noaggress
         -O autoprefetch,noautothread,fusion2,nomsgs,noneqmsgs,nooverindex
         -O pattern,shortcircuit2,unroll2,nozeroinc
         -h noadd_paren,align_arrays,caf,noconcurrent,nocontiguous,nofp_trap
         -h nofunc_trace,nomessage,noopmp_analyze,noopmp_trace,nopat_trace
         -h safe_addr
         -h omp,noacc
         -h flex_mp=default
         -h cpu=x86-64,abudhabi,gemini
         -K trap=none
         -s default32 -eh
         -d acdgjnopvBDEFIPQRSTZ0
         -e mqswAC
```

Recommended optimization options

please see Cray's Programming Environment User's Guide

short overview is available here: http://archer.ac.uk/training/courses/craytools/pdf/compilers.pdf

OpenMP

Add this flag to any of the above:

```
-h omp
```

OpenMP Case Study

Below is performance analysis of Molecular Dynamic program compiled with different compilers, compiler options:

below compilation is done using: PrgEnv-gnu/5.2.40
build on compute nodes with: `gfortran -fopenmp md.f90 -o md -static`

32 threads 10.5670 seconds
4 threads 77.5051 seconds
2 threads 151.272 seconds
1 thread 264.683 seconds

build on login nodes with: `ftn -fopenmp md.f90 -o md -static`

32 threads 7.93083 seconds
4 threads 58.9941 seconds
2 threads 115.892 seconds
1 thread 217.989 seconds

build on login nodes with: `ftn -fopenmp md.f90 -o md`

32 threads 7.92050 seconds
4 threads 58.9770 seconds
2 threads 115.884 seconds
1 thread 217.953 seconds

run on login and build with: `gfortran -fopenmp md.f90 -o md`

export OMP_NUM_THREADS=1
312.363 seconds

export OMP_NUM_THREADS=2
156.193 seconds

export OMP_NUM_THREADS=4
79.7515 seconds

below compilation is done using: `PrgEnv-cray/5.2.40`

build on login nodes with: `ftn -h omp md.f90 -o md`

32 threads 4.65179 seconds
4 threads 34.6667 seconds
2 threads 69.3108 seconds
1 thread 103.958 seconds

build on login nodes with: `ftn -fopenmp -O3 md.f90 -o md -static`

32 threads 2.15595 seconds
4 threads 15.0827 seconds
2 threads 30.0341 seconds
1 thread 52.1023 seconds