Introduction to Compiling

Daniel Lucio
User Support

Oct 18th 2011, ORNL, TN

HowTo compile

- The Cray system supports a variety of compilers from a variety of vendors for the major languages C (UPC), C++ and Fortran.

PGI , CCE , INTEL , PATHSCALE , GNU

- Because of the multiplicity of possible compilers, Cray supplies compiler drivers, wrapper scripts, and disambiguation man pages. No matter which vendor's compiler module is loaded, always use one of the following commands to invoke the compiler.

Language	Wrapper name
`C (UPC)`	`cc`
`C++`	`CC`
`Fortran (77-95)`	`ftn`

HowTo compile

The compilers also come as modules called Programming Environments. Each compiler has a different module which can be treated as any other module.

Compiler	module name	compiler module name
PGI	PrgEnv-pgi	pgi
CCE	PrgEnv-cray	cce
Intel	PrgEnv-intel	intel
GNU	PrgEnv-gnu	gcc
Pathscale	PrgEnv-pathscale	pathscale

Why use the wrappers

- Use the Cray compiler wrappers `cc`, `CC` and `ftn`, to compile programs for the compute nodes.
- The compiler wrappers know where most of the correct Cray provided libraries and include files are, if the corresponding module is loaded.
- You do not need to know where the MPI libraries are.
- The wrappers automatically add the correct tuning parameters for the Istanbul Processor.
- Automatically add support for numerical libraries.
- Forces the final executable to be self-contained

How to compile

MPI Hello World example

```
/* C Example */
#include <stdio.h>
#include <mpi.h>


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( "Hello2 world2 from process %d of %d\n", rank, size );

  MPI_Finalize();

  return 0;
}
```

```
> cc -o hello hello.c
```

How to compile

MPI Hello World example with another compiler

```
> module swap PrgEnv-pgi PrgEnv-gnu
> cc -o hello hello.c
```

MPI Hello World example with an older compiler version

```
> module swap pgi/9.0.3 pgi/7.2.5
> module swap xtpe-istanbul xtpe-barcelona
> cc -o hello hello.c
```

What compiler flags to use?

```
-fast –Mipa=fast      -Minfo=all –Mneginfo=all
```

How to compile

Using 3rd party hdf5/1.6.7 library example

```
> module load hdf5/1.6.7
> module help hdf5/1.6.7

----------- Module Specific Help for 'hdf5/1.6.7' -----------------

Sets up environment to use serial HDF5 1.6.7 with any compiler.
Usage:   ftn test.f90 ${HDF5_FLIB}  OR  h5fc test.f90
    or   cc test.c ${HDF5_CLIB}     OR  h5cc test.c
The hdf5 module must be reloaded if you change the PrgEnv
   or you must issue a 'module update hdf5' command.
This version is deprecated and will soon be no longer available.

> cc -o myhdf5test  h5_copy18.c ${HDF5_CLIB}
```

The WRONG way to do a Makefile

```
F90=pgf90

##################################################
# 1. scalapack/lapack/blas libraries
##################################################
LDFLAGS+=-L/opt/xt-libsci/10.4.7/gnu/lib
LB_LIBS=-lsci_istanbul_mp -lgfortran

##################################################
# 2. HDF5 libraries
##################################################
HDF5_INCLUDES = -I/sw/xt/hdf5/1.6.10/cnl2.2_gnu4.4.3/include
HDF5_LIBS = ${HDF5_CLIB} -lhdf5_hl -lhdf5 -lsz -lz

##################################################
# 3. gsl
##################################################
GSL_HOME=/sw/xt/gsl/1.13/cnl2.2_gnu4.3.3
CPPFLAGS+=-I$(GSL_HOME)/include
LDFLAGS+=-L$(GSL_HOME)/lib
GSL_LIBS=-lgsl

##################################################
```

The RIGHT way to write a Makefile

```
F90=ftn

##################################################
# 1. scalapack/lapack/blas libraries
##################################################
LDFLAGS+=
LB_LIBS=

##################################################
# 2. HDF5 libraries
##################################################
HDF5_INCLUDES = ${HDF5_CLIB}
HDF5_LIBS = -lhdf5_hl ${HDF5_CLIB}

##################################################
# 3. gsl
##################################################
GSL_HOME=
CPPFLAGS+=-I${GSL_DIR}/include
LDFLAGS+=-L${GSL_DIR}/lib
GSL_LIBS=-lgsl –lgslcblas

##################################################
```

Choosing a Compiler

- Try every compiler available to you, there’s no “best”
  - PGI, Cray, Pathscale, Intel, GCC are all available, but not necessarily on all machines
- Each compiler favors certain optimizations, which may benefit applications differently
- Test your answers carefully
  - Order of operation may not be the same between compilers or even compiler versions
  - You may have to decide whether speed or precision is more important to you

Choosing Compiler Flags

- PGI
  - -fast –Mipa=fast
  - man pgf90; man pgcc; man pgCC
- Cray
  - <none, turned on by default>
  - man crayftn; man craycc ; man crayCC
- Pathscale
  - -Ofast
  - man eko (“Every Known Optimization”)
- GNU
  - -O2 / -O3
  - man gfortran; man gcc; man g++
- Intel
  - -fast
  - man ifort; man icc; man iCC

- Compiler Feedback
  - Most compilers can tell you a lot about what they do to your code
    - Did your important loop vectorize?
    - Was a routine inlined?
    - Was this loop unrolled?
  - It's just as important to know what the compiler didn't do
    - Some C/C++ loops won't vectorize without some massaging
    - Some loop counts aren't known at runtime, so optimization is limited
  - Flags to know:
    - PGI: -Minfo=all –Mneginfo=all
    - Cray: -rm (Fortran) (or also –O[neg]msgs), -hlist=m (C) (or also –h [neg]msgs)
    - Pathscale: -LNO:simd_verbose=ON
    - Intel: -vec-report1

```
! Matrix Multiply
do k = 1, N
  do j = 1, N
    do i = 1, N
      c(i,j) = c(i,j) + &
           a(i,k) * b(k,j)
    end do
  end do
end do
```

```
24, Loop interchange
  produces reordered loop
  nest: 25,24,26
26, Generated an alternate
  loop for the loop
         Generated vector
  sse code for the loop
         Generated 2
  prefetch instructions
  for the loop
```

Common impediments to vector optimization

There are several common coding issues that may prevent vectorization. The programmer may have enough knowledge to provide additional information to the compiler to work around these issues.

In C and C++ the compiler may not have enough information about the pointers passed into a subroutine to be able to determine that those pointers don't overlap. (-Msafeptr option or pragma or restrict keyword)

Function calls can be inlined to allow vectorization (-Minline)

Constants may be of the wrong type (-Mfcon)

Loops may be too long or too short. In both cases, additional options to the vectorizer may be successful in generating vector code.

Original

```
(   42) C       THE ORIGINAL
(   43)
(   44)        DO 48070 I = 1, N
(   45)         A(I) =  (B(I)**2 + C(I)**2)
(   46)         CT   = PI * A(I) + (A(I))**2
(   47)         CALL SSUB (A(I), CT, D(I), E(I))
(   48)         F(I) = (ABS (E(I)))
(   49)  48070 CONTINUE
(   50)
```

PGI

44, Loop not vectorized: contains call

Pathscale

Nothing

Restructured

```
(    69) C       THE RESTRUCTURED
(    70)
(    71)       DO 48071 I = 1, N
(    72)        A(I) = (B(I)**2 + C(I)**2)
(    73)        CT   = PI * A(I) + (A(I))**2
(    74)        E(I) = A(I)**2 + (ABS (A(I) + CT)) * (CT * ABS (A(I) - CT))
(    75)        D(I) = A(I) + CT
(    76)        F(I) = (ABS (E(I)))
(    77) 48071 CONTINUE
(    78)
```

PGI

71, Generated an alternate loop for the inner loop

Unrolled inner loop 4 times

Used combined stores for 2 stores

Generated 2 prefetch instructions for this loop

Unrolled inner loop 4 times

Used combined stores for 2 stores

Generated 2 prefetch instructions for this loop

Pathscale

(lp48070.f:71) LOOP WAS VECTORIZED.

Additional compiler optimizations

The –fast flag is the 90/90 solution for code optimization. That is, it achieves about 90% of the possible performance for about 90% of the codes.

That means there are some additional areas that can be explored.

Interprocedural analysis can be helpful for C codes and Fortran codes without interface blocks. (Interface blocks are to the language specification what IPA is to the compiler)

```
> ftn -fast -Minfo -Mipa=fast foo.f -o foo
```

***If compiling and linking are done in separate steps, you must be sure to pass the IPA flag to the linker too.

IPA involves an additional pass of the compiler.

Compiler Limitations

Known issues with the PGI compiler con Cray XTs

- At this time, PGI compilers do not produce code optimized for AMD Interlagos processors. Code compiled with the PGI compilers can be expected to suffer performance degradation if executed on compute nodes with Interlagos CPUs. This limitation is expected to be removed in a future release.
- The PGI compilers are not able to handle template-based libraries such as Tpetra.
- When linking in ACML routines, you must compile and link all program units with `-Mcache_align` or an aggregate option that incorporates `-Mcache_align` such as `fastsse`.
- The `-Mconcur` (auto-concurrentization of loops) option is not supported on Cray systems.
- The `-mprof=mpi, -Mmpi, and -Mscalapack` options are not supported. The PGI debugger, PGDBG, is not supported on Cray systems.
- The PGI profiling tools, `pgprof` and `pgcollect`, are not supported on Cray systems.
- The PGI Compiler Suite does not support the UPC or Coarray Fortran parallel programming models.

Compiler Limitations

Known issues with the Pathscale compiler con Cray XTs

- The PathScale Compiler Suite does not support the UPC or Coarray Fortran parallel programming models.
- The PETSc 3.1.0.8 library of parallel linear and nonlinear solvers does not support the PathScale Compiler Suite. Use PETSc 3.0.x instead.
- Beginning with PathScale release 4.0.9, Cray no longer provides PathScale-specific versions of the Cray Scientific Libraries (LibSci), or of PETSc, Trilinos, or other third-party scientific libraries (TPSL). Users are advised to obtain third-party scientific libraries directly from the respective library developers.

Known issues with the GNU compiler con Cray XTs

- Code compiled with gfortran using the options `--whole-archive,-lpthread` gets the following warning message issued by libpthread.a (sem_open.o):

```
warning: the use of 'mktemp' is dangerous, better use 'mkstemp'
```

- The `--whole-archive` option is necessary to avoid a runtime segmentation fault when using OpenMP libraries. This warning can be safely ignored.

More information

More information about compiling can also be found at

NICS
http://www.nics.tennessee.edu/computing-resources/kraken/compiling

OLCF
http://www.olcf.ornl.gov/kb_articles/compilers-using-compilers-on-jaguar/

Cray
http://docs.cray.com