Overview of Compilers and Libraries on XC30

Cray Programming Environment Distribution Focus on Performance and Productivity

Programming Languages
- Fortran
- C
- C++
- Python

Programming models
- Distributed Memory (Cray MPT)
  - MPI
  - SHMEM
- Shared Memory
  - OpenMP 3.1
  - OpenACC
- PGAS & Global View
  - UPC (CCE)
  - CAF (CCE)
  - Chapel

Compilers
- Cray Compiling Environment (CCE)
- GNU
- 3rd Party Compilers

Tools
Environment setup
- Modules

Debuggers
- TOTALVIEW TECHNOLOGIES
- Allinea (DDT)
- lgdb

Debugging Support Tools
- Abnormal Termination Processing
- STAT

Performance Analysis
- CrayPat
- Cray Apprentice2

Scoping Analysis
- Reveal

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

I/O Libraries
- NetCDF
- HDF5

Cray developed
Licensed ISV SW
3rd party packaging
Cray added value to 3rd party

THE SUPERCOMPUTER COMPANY

Using the Compiler Driver Commands

- You use compiler driver commands to launch all Cray XC compilers
- The syntax for the compiler driver is:
  - `cc` | `CC` | `ftn` [Cray_options | Intel_options | GNU_options] files
- For example, to use any Fortran compiler (CCE, Intel, GNU) to compile prog1.f90
  - Use this command:
    - `% ftn prog1.f90`
- The compiler drivers are setup by the PrgEnv-**???** Module
  - PrgEnv-**cray**, PrgEnv-**gnu**, PrgEnv-**intel**
- Check that the craype-ivybridge module is loaded
- The drivers automatically support an MPI build
  - No need to use specific wrappers such as mpiifort, mpicc

PLEASE NOTE : Cross Compiling Environment

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

- **Do not use crayftn, craycc, ifort, icc, gcc, g++… unless you want a Linux executable for the service node**
  - ALWAYS Use `ftn`, `cc`, or `CC` instead
  - Use the direct compiler commands if the executable is supposed to run on the service nodes (utilities, setup, …)

About the `–I`, `–L` and `–l` flags

- **The compiler driver commands add specific flags depending on the loaded modules**
  - E.g. flags for debugging information when perftools module is loaded
- **The modules add library and include paths for you, don't do this yourself (for example in a Makefile)**
  - No additional MPI flags are needed (included by wrappers)
  - You do not need to add any `–I`, `–l` or `–L` flags for the Cray provided libraries
  - If your Makefile needs an input for `–L` to work correctly, try using '.'
  - If you really, really need a specific path, try checking 'module show X' for some environment variables
- **Use the `–v`/`–V` options or set the environment variable `CRAYPE_INFO_MESSAGE_ON` to get details**

Dynamic vs Static linking

- **Currently static linking is default when using the compiler driver commands**
  - Independent of the PrgEnv loaded
  - Will change in the future. Already changed when linking for GPUs (XK7 nodes)
- **To specify how to link**
  - you can either set `CRAY_LINK_TYPE` to `static` or `dynamic`
  - Or link with the **-static** or **-dynamic** options of the compiler driver commands
- **Features of dynamic linking:**
  - smaller executable, automatic use of new libs
  - Might need longer startup time to load and find the libs
  - Environment (modules) should be the same between your compiler setup and your batch script (eg. when switching to PrgEnv-intel)
- **Features of static linking:**
  - Faster startup
  - Larger executable (but who cares)
- **If you want to hardcode the rpath into the executable use**
  - `CRAY_ADD_RPATH=yes`
  - This will always load the same version of the lib when running, independent on version being installed

The Cray Compilation Environment (CCE)

CCE Overview

- **Cray technology focused on scientific applications**
  - Takes advantage of automatic vectorization
  - Takes advantage of automatic shared memory parallelization
- **Standard conforming languages and programming models**
  - ANSI/ISO Fortran 2003 and Fortran 2008 standards compliant
  - ANSI/ISO C99 and C++2003 compliant
  - OpenMP 3.1 compliant, working on OpenMP 4.0
  - OpenACC 1.0
- **OpenMP and automatic multithreading fully integrated**
  - Share the same runtime and resource pool
  - Aggressive loop restructuring and scalar optimization done in the presence of OpenMP
  - Consistent interface for managing OpenMP and automatic multithreading

CCE Overview (cont)

- **Intel Sandy/Ivy Bridge support**
- **x86/NVIDIA compiler and library development**
- **Support for MPI 2.2**
- **PGAS languages (UPC & Fortran Coarrays) fully optimized and integrated into the compiler**
  - UPC 1.2 and Fortran 2008 coarray support
  - No preprocessor involved
  - Target the network appropriately
- **Support for hybrid programming using MPI across node and OpenMP within the node**
- **Support for IEEE floating-point arithmetic and IEEE file formats**
- **Cray performance tools and debugger support**
- **Program Library**
- **CCE 8.1 was released on September, 2012**
  - The full release overview can be found at: http://docs.cray.com/books/S-5212-81/

CCE Compiler Testing

- **Roughly 35,000 nightly regression tests run for Fortran (14,000), C (7,000), and C++ (14,000)**
  - Default optimization, but for multiple targets (x86, x86+AVX+FMA, x86+NVIDIA), plus “debug” and “production” compiler versions
  - Additionally, cycle through “options testing” with the same test base
    - Fortran: -G0, -G1, -G2, -O0, -Oipa0, -Oipa5 -hpic, -O3,fp3 -e0
    - C and C++: -Gn, -O0, -hipa0, -hipa5, -hpic, -O3 -hfp3 -hzero
    - Additional tests and suites have been added for GPU testing
    - And some “stress test” option sets to create worse-case scenarios for the compiler
    - Other combinations as necessary and by request
- **Performance regression testing done weekly using important applications and benchmarks**
- **Functional and performance regressions typically use an automated system that isolates the change to a specific compiler or library mod**
- **Issues that are found as a result of testing but not immediately addressed have bugs opened to track them**

General Cray Compiler Flags

- **Optimisation Options**
  - **-O2** — optimal flags [enabled by default]
  - **-O3** — aggressive optimization
  - **-O ipaN (ftn)** or **-hipaN (cc/CC)** — inlining, N=0-5 [default N=3]
- **Create listing files with optimization info**
  - **-ra (ftn)** or **-hlist=a (cc/CC)** — creates a listing file with all optimization info
  - **-rm (ftn)** or **-hlist=m (cc/CC)** — produces a source listing with loopmark information
- **Parallelization Options**
  - **-O omp (ftn)** or **-h omp (cc/CC)** — Recognize OpenMP directives [default]
  - **-O threadN (ftn)** or **-h threadN (cc/CC)** — control the compilation and optimization of OpenMP directives, N=0-3 [default N=2]

➔ **More info: man crayftn, man craycc, man crayCC**

Inlining with CCE

- **Inlining is enabled by default**
  - Command line option **-OipaN (ftn)** or **-hipaN (cc/CC)** where N=0…5, provides a set of choices for inlining behavior
    - 0 - All inlining and cloning are disabled. All inlining and cloning compiler directives are ignored.
    - 1 - Directive inlining. Inlining is attempted for call sites and routines that are under the control of an inlining compiler directive. Cloning disabled and cloning directives are ignored.
    - 2 - Inlining. Inline a call site to an arbitrary depth as long as the expansion does not exceed some compiler-determined threshold. Cloning disabled and cloning directives are ignored.
    - 3 (default) - Constant actual argument inlining and tiny routine inlining. This includes levels 1 and 2, plus any call site that contains a constant actual argument. Cloning disabled and cloning directives are ignored.
    - 4 - Aggressive inlining. This includes levels 1, 2, and 3, plus a call site does not have to reside in a loop body to inline. Cloning disabled and cloning directives are ignored.
    - 5 - Cloning. This includes levels 1, 2, 3, and 4, plus routine cloning is attempted if inlining fails at a given call site. Cloning directives are enabled.

Inlining with CCE (cont)

- **By default, all inlining candidates come from the current source file**
- **The -Oipafrom= (ftn) or -hipafrom= (cc/CC) option instructs the compiler to look for inlining candidates from other source files, or a directory of source files**
  - “ftn -Oipafrom=b.f a.f” tells the compiler to look for inlining candidates within b.f when compiling a.f
  - “cc -hipafrom=./dir src.c” tells the compiler to look for inlining candidates in all the valid source files that exist in the directory ./dir when compiling src.c
- **Cross language inlining is not supported**

Whole-Program Compilation

- **The Program Library (PL) feature allows the user to specify a repository of compiler information for an application build**
  - This repository provides the framework for future productivity features such as
    - Whole program static error detection
    - Incremental recompilation
    - Provide support for the future Cray interactive whole program performance analysis and tuning assistant Reveal
- **Two command line options control the Program Library functionality**
  - **-h pl = <PL_path>** specifies the repository
    - -hpl=./PL.1 tells the compiler to either update the Program Library "./PL.1" if it exists, or create it if it does not exist.
    - <PL_path> should specify a single location to be used for entire application build. If a makefile changes directories during a build, an absolute path might be necessary.
  - **-h wp** enables whole-program mode
- **Needed by Reveal (to be covered in the tool talk on Thursday)**

Whole-Program Compilation (cont)

- **Whole-program mode (-hwp) requires a program library (-hpl=) and both options must be specified on all compilation command lines as well as on the link line.**
  - The compiler frontend is invoked for the compilation (-c) command lines
  - The compiler backend (inliner, optimizer, code generator) is invoked for all source files when the link line is specified.
  - While **-hwp** might have a negative affect on overall compile time due to increased inlining, it is most usually a compile time shift, where -c compilations become quite fast and the time spent on the link step increases.
  - Setting the environment variable “NPROC” to a number greater than 1 instructs the compiler to invoke NPROC backend processes concurrently. The backend invocations are independent of each other and setting NPROC to a level that is appropriate for the host build machine can improve compile time.
- **Whole-program mode (-hwp) allows the inliner to see all inline candidates in the application.**
  - This option makes cross file inlining automatic
    - Removes the need for **-O/-h ipafrom=**
    - Inlining heuristics are still controlled by **-O/-h ipaN**

Unrolling

- **By default, the compiler attempts to unroll all loops, unless the `NOUNROLL` directive is specified for a loop**
  - Generally, unrolling loops increases single processor performance at the cost of increased compile time and code size

- **-OunrollN (ftn) or -hunrollN (cc/CC) where N=0,1,2, globally control loop unrolling and changes the assertiveness of the UNROLL directive**
  - **0**: No unrolling (ignore all `UNROLL` directives and do not attempt to unroll other loops)
  - **1**: Attempt to unroll loops if there is proof that the loop will benefit
  - **2**: (Default) Attempt to unroll all loops (includes array syntax implied loops), except those marked with the `NOUNROLL` directive.

Vectorization

- **The -OvectorN (ftn) or -hvectorN (cc/CC) where N=0…3, specify the level of automatic vectorizing to be performed. Vectorization results in significant performance improvements with a small increase in object code size. Vectorization directives are unaffected by this option**
  - **0**: No automatic vectorization
  - **1**: Specifies conservative vectorization. Loop nests are restructured. No vectorizations that might create false exceptions are performed. Results may differ slightly from results obtained when N=0 is specified because of vector reductions
  - **2**: (Default) Specifies moderate vectorization. Characteristics include moderate compile time and size. Loop nests are restructured
  - **3**: Specifies aggressive vectorization. Loop nests are restructured. Vectorizations that might create false exceptions in rare cases may be performed.

Floating-Point Optimizations

- **The -hfpN option, where N=0…4, controls the level of floating-point optimizations: N=0 gives the compiler minimum freedom to optimize floating-point operations, while N=4 gives it maximum freedom. The higher the level, the less the floating-point operations conform to the IEEE standard.**
  - **N=0 and N=1**: Use this option only when your code pushes the limits of IEEE accuracy or requires strong IEEE standard conformance. Executable code is slower than higher floating-point optimization levels
  - **N=2**: default value. It performs various generally safe, non-conforming IEEE optimizations
  - **N=3**: This option should be used when performance is more critical than the level of IEEE standard conformance provided by N=2. This is the suggested level of optimization for many applications.
  - **N=4**:You should only use this option if your application uses algorithms which are tolerant of reduced precision.

Floating-Point Optimization Flags Comparison

- **The table lists some of the optimizations performed; the compiler may perform other optimizations not listed.**

Optimization	fp0	fp1	fp2 (default)	fp3	fp4
Safety	Maximum	High	High	Moderate	Low
Complex divisions	Accurate and slower	Accurate and slower	Fast	Fast	Fast
Exponentiation rewrite	None	None	When benefit is very high	Always	Always
Strength reduction	None	None	Fast	Fast	Fast
Rewrite division as reciprocal equivalent	None	None	Yes	Aggressive	Aggressive
Floating point reductions	Slow	Fast	Fast	Fast	Fast
Expression factoring	None	Yes	Yes	Yes	Yes
Inline 32-bit operations	No	No	No	Yes	Yes

Why are CCE's results sometimes different?

- **We do expect applications to be conformant to language requirements**
  - This include not over-indexing arrays, no overlap between Fortran subroutine arguments, and so on
  - Applications that violate these rules may lead to incorrect results or segmentation faults
  - Note that languages do not require left-to-right evaluation of arithmetic operations, unless fully parenthesized
    - This can often lead to numeric differences between different compilers
    - Use **-hadd_paren** to add automatically parenthesis to select associative operations (+,–,*). Default is **-hnoadd_paren**
- **We are also fairly aggressive at floating point optimizations that violate IEEE requirements**
  - Use **-hfp[0-4]** flag to control that

Why are CCE's results sometimes different? (cont)

- **Results can vary with the number of ranks or threads**
  - Use **-hflex_mp=option** to control the aggressiveness of optimizations which may affect floating point and complex repeatability when application requirements require identical results when varying the number of ranks or threads.
  - **option** in order from least aggressive to most is:
    - intolerant: has the highest probability of repeatable results, but also has the highest performance penalty
    - conservative: uses more aggressive optimization and yields higher performance than intolerant, but results may not be sufficiently repeatable for some applications
    - default: uses more aggressive optimization and yields higher performance than conservative, but results may not be sufficiently repeatable for some applications
    - tolerant: uses most aggressive optimization and yields highest performance, but results may not be sufficiently repeatable for some applications

Fortran Source Preprocessing

For a source file to be preprocessed automatically, it must have an uppercase extension, either .F (for a file in fixed source form), or .F90, .F95, .F03, .F08, or .FTN (for a file in free source form). To specify preprocessing of source files with other extensions, including lowercase ones, use the -eP or -eZ options

- **-eP**: Performs source preprocessing on Fortran source files, **but does not compile**. Generates file.1, which contains the source code after the preprocessing has been performed and the effects have been applied to the source program.
- **-eZ**: similar to **-eP**, but it also performs compilation on Fortran source files

Other flags in brief

- **-h restrict=[a|f]**
  - C/C++ option which tells the compiler to treat certain classes of pointers as restricted pointers. You can use this option to enhance optimizations (this includes vectorization).
- **-h cacheN**
  - Specifies the levels of automatic cache management to perform. Values for N are between 0 (cache blocking turned off) and 3 (aggressive automatic cache management). Symbols are placed in the cache when the possibility of cache reuse exists. Default value is N=2.

Other flags in brief (cont)

- **-h Pic**
  - Generate position independent code (PIC), which allows a virtual address change from one process to another, as is necessary in the case of shared, dynamically linked objects. The virtual addresses of the instructions and data in PIC code are not known until dynamic link time.

- **-h[system|default]_alloc**
  - The **-hsystem_alloc** option causes the compiler to use the native malloc implementation provided by the OS. By default, the compiler uses a modified malloc implementation which offers better support for Cray memory needs. This is a link-time option.

Diagnostic Flags

- **-Rb (ftn) or -h bounds (cc/CC)**
  - Fortran: Enables checking of array bounds at runtime.
  - C/C++: Enables checking of pointer and array references at runtime. **-h nobounds** disables these checks.
- **-eo (ftn) or -hdisplay_opt (cc/CC)**
  - Display the compiler optimization settings currently in force.
- **-T (ftn)**
  - Disables the compiler but displays all options currently in effect.

Recommended CCE Compilation Options

- **Use default optimization levels**
  - It's the equivalent of most other compilers -O3 or -fast
  - It is also our most thoroughly tested configuration
- **Use -O3,fp3 (or -O3 -hfp3, or some variation) if the application runs cleanly with these options**
  - **-O3** only gives you slightly more than the default **-O2**
  - We also test this thoroughly
  - **-hfp3** gives you a lot more floating point optimization (default is **-hfp2**)
- **If an application is intolerant of floating point reordering, try a lower -hfp number**
  - Try **-hfp1** first, only **-hfp0** if absolutely necessary (**-hfp4** is the maximum)
  - Might be needed for tests that require strict IEEE conformance
  - Or applications that have 'validated' results from a different compiler
- **Do not use too aggressive optimizations , e.g. -hfp4**
  - Higher numbers are not always correlated with better performance

OpenMP

- **OpenMP is ON by default**
  - This is the opposite default behavior that you get from GNU and Intel compilers
  - Optimizations controlled by **-OthreadN (ftn)** or **-hthreadN (cc/CC)**, N=0-3 [default N=2]
  - To shut off use **-O/-h thread0** or **-xomp (ftn)** or **-hnoomp**

- **Autothreading is NOT on by default**
  - **-hautothread** to turn on
  - Interacts with OpenMP directives

- **If you do not want to use OpenMP and have OMP directives in the code, make sure to shut off OpenMP at compile time**

Cray programming environment: assign

- Associates options with Fortran I/O unit numbers and file names for use during the library open processing, i.e. you can tell the Fortran runtime how to treat a file, without changing your code
  - **assign [assign options] assign_object**

- Interesting assign options
  - **-R** removes all assign options for assign_object
  - **-N <numcon>** specifies foreign numeric conversion

- assign_object used to specify the object of assign options
  - **f:<filename>** applies to filename
  - **u:<unit>** applies to Fortran unit number
  - **g:su** applies to all Fortran sequential unformatted files

How to handle byte-swapped files with CCE

- **Explicit usage of assign**
  - Can control which files are byte-swapped

```
export FILENV=.assign
assign -R
assign -N swap_endian f:aof
aprun a.out
```

- **Link the application with –hbyteswapio**
  - Forces byte-swapping of all input and output files for direct and sequential unformatted I/O
  - This is equivalent to set

```
assign -N swap_endian g:su ←all sequential unformatted
assign -N swap_endian g:du ←all direct unformatted
```

➔ **More info: man assign (when PrgEnv-cray loaded)**

CRAY
THE SUPERCOMPUTER COMPANY

Default Output Formats

- **List-directed output depends on the value being written**
  - `assign` command can be used to change that
- **Let's take this code for example**

```
integer :: ia(4)
real    :: ra(4)
ia = 102
ra = 200.10
print *, ' ia=',ia
print *, ' ra=',ra
```

Output

```
ia= 4*102
ra= 4*200.100006
```

By setting

```
export FILENV=FILETMP
assign -U on g:sf
```

and rerunning the code (without recompiling it), the output becomes

```
ia=          102          102          102          102
ra=     200.1000     200.1000     200.1000     200.1000
```

➔ **More info: man assign (when PrgEnv-cray loaded)**

CRAY THE SUPERCOMPUTER COMPANY

CCE Directives

- **The Cray compiler supports a full and growing set of directives and pragmas**
  - Fortran:
    - `!dir$ concurrent`
    - `!dir$ ivdep`
    - `!dir$ interchange`
    - `!dir$ unroll`
    - `!dir$ loop_info [max_trips] [cache_na]` ... Many more
    - `!dir$ blockable`
  - For C/C++ replace `!dir$` with `#pragma [_CRI]`
    - The `_CRI` specification is optional; it ensures that the compiler will issue a message concerning any directives that it does not recognize. Diagnostics are not generated for directives that do not contain the `_CRI` specification.

➔ More info: man directives
man loop_info

Macros

- **Cray compilers define the following macros:**
  - Fortran: `_CRAYFTN`
  - C/C++: `_CRAYC`

- **For example, the macros can be used to ensures that other compilers will not interpret the directives by encapsulating them inside `#if` … `#endif`**

```
#if _CRAYC
     #pragma _CRI directive
#endif
```

  - Some compilers diagnose any directives that they do not recognize. The Cray C/C++ compilers diagnose directives that are not recognized only if the _CRI specification is used.

Loopmark: Compiler Feedback

- **ftn –rm …   or   cc/CC –hlist=m …**
- **Compiler generates an <source file name>.lst file**
  - Contains annotated listing of your source code with letter indicating important optimizations

```
%%%    L o o p m a r k   L e g e n d    %%%
Primary Loop Type       Modifiers
------- ---- ----       ---------
                        a - vector atomic memory operation
A  - Pattern matched    b - blocked
C  - Collapsed          f - fused
D  - Deleted            i - interchanged
E  - Cloned             m - streamed but not partitioned
I  - Inlined            p - conditional, partial and/or computed
M  - Multithreaded      r - unrolled
P  - Parallel/Tasked    s - shortloop
V  - Vectorized         t - array syntax temp used
W  - Unwound            w - unwound
```

Example: Cray loopmark messages

```
b-------<  do i3=2,n3-1
b b-----<     do i2=2,n2-1
b b Vr--<        do i1=1,n1
b b Vr            u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
b b Vr    *          + u(i1,i2,i3-1) + u(i1,i2,i3+1)
b b Vr            u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
b b Vr    *          + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
b b Vr-->        enddo
b b Vr--<        do i1=2,n1-1
b b Vr            r(i1,i2,i3) = v(i1,i2,i3)
b b Vr    *         - a(0) * u(i1,i2,i3)
b b Vr    *         - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
b b Vr    *         - a(3) * ( u2(i1-1) + u2(i1+1) )
b b Vr-->        enddo
b b----->     enddo
b------->  enddo
```

Example: Cray loopmark messages (cont)

ftn-6289 ftn: VECTOR File = resid.f, Line = 29
A loop starting at ***line 29 was not vectorized*** *because a recurrence was found on "U1" between lines 32 and 38.*
ftn-6049 ftn: SCALAR File = resid.f, Line = 29
A loop starting ***at line 29 was blocked with block size 4.***
ftn-6289 ftn: VECTOR File = resid.f, Line = 30
A loop starting at ***line 30 was not vectorized*** *because a recurrence was found on "U1" between lines 32 and 38.*
ftn-6049 ftn: SCALAR File = resid.f, Line = 30
A loop starting at ***line 30 was blocked with block size 4.***
ftn-6005 ftn: SCALAR File = resid.f, Line = 31
A loop starting at ***line 31 was unrolled 4 times.***
ftn-6204 ftn: VECTOR File = resid.f, Line = 31
A loop starting at ***line 31 was vectorized.***
ftn-6005 ftn: SCALAR File = resid.f, Line = 37
A loop starting at ***line 37 was unrolled 4 times.***
ftn-6204 ftn: VECTOR File = resid.f, Line = 37
A loop starting at ***line 37 was vectorized.***

Compiler Message System

- The `explain` command displays an explanation of any message issued by the compiler. The command takes as an argument, the message number, including the number's prefix (`ftn-` for ftn or `CC-` for cc/CC)

  Example:

```
% cc bug.c
CC-24 cc: ERROR File = bug.c, Line = 1
An invalid octal constant is used.
int i = 018;
         ^

1 error detected in the compilation of "bug.c".
% explain CC-24
An invalid octal constant is used.
Each digit of an octal constant must be between 0 and
7, inclusive. One or more digits in the indicated
octal constant are outside of this range. Change each
digit in the octal constant to be within the valid
range.
```

➔ **More info: man explain (when PrgEnv-cray loaded)**

Compiler Message System (cont)

- **-h [no]msgs**
  - Enables or disables the writing of optimization messages to `stderr`. Default is **-h nomsgs**
- **-h [no]negmsgs**
  - Enables or disables the writing of messages to `stderr` that indicate why optimizations such as vectorization, inlining, or cloning did not occur in a given instance. Default is -h nonegmsgs
- **-m *n* (ftn) or -h msglevel_*n* (cc/CC)**
  - Specifies the lowest level of severity of messages to be issued. Messages at the specified level and above are issued. Values of ***n*** are:
    - 0: Comment
    - 1: Note
    - 2: Caution
    - 3: Warning (default)
    - 4: Error
- **-M msg*n*[,…] (ftn) or -h nomessage=*n*[:...] (cc/CC)**
  - Suppresses specific messages at the warning, caution, note, and comment levels, where ***n*** is the number of a message to be disabled (multiple numbers are possible)

THE SUPERCOMPUTER COMPANY

Brief Overview on Intel and GNU Compilers

CCE – GNU – Intel compilers

- **More or less all optimizations and features provided by CCE are available in Intel and GNU compilers**
  - GNU compiler serves a wide range of users & needs
    - Default compiler with Linux, some people only test with GNU
    - Defaults are conservative (e.g. -O1)
      - -O3 includes vectorization and most inlining
    - Performance users set additional options
  - Intel compiler is typically more aggressive in the optimizations
    - Defaults are more aggressive (e.g -O2), to give better performance "out-of-the-box"
      - Includes vectorization; some loop transformations such as unrolling; inlining within source file
    - Options to scale back optimizations for better floating-point reproducibility, easier debugging, etc.
    - Additional options for optimizations less sure to benefit all applications
  - CCE is even more aggressive in the optimizations by default
    - Better inlining and vectorization
    - Aggressive floating-point optimizations
    - OpenMP enabled by default

Cray, Intel and GNU compiler flags

Feature	Cray	Intel	GNU
Listing	-ra (fnt) -hlist=a (cc/CC)	-opt-report3	-fdump-tree-all
Free format (ftn)	-f free	-free	-ffree-form
Vectorization	By default at -O1 and above	By default at -O2 and above	By default at -O3 or using -ftree-vectorize
Inter-Procedural Optimization	-hwp	-ipo	-flto (note: link-time optimization)
Floating-point optimizations	-hfpN, N=0...4	-fp-model [fast|fast=2|precise| except|strict]	-f[no-]fast-math or -funsafe-math-optimizations
Suggested Optimization	(default)	-O2 -xAVX	-O2 -mavx -ftree-vectorize -ffast-math -funroll-loops
Aggressive Optimization	-O3 -hfp3	-fast	-Ofast -mavx -funroll-loops
OpenMP recognition	(default)	-fopenmp	-fopenmp
Variables size (ftn)	-s real64 -s integer64	-real-size 64 -integer-size 64	-freal-4-real-8 -finteger-4-integer-8

Linking with MKL and PrgEnv-cray

- **PrgEnv-cray compatible with sequential, not threaded, MKL**
  - MKL can be used as an alternative to Cray's libsci for CCE
- **Examples assume you have loaded the Intel module (to define the env var INTEL_PATH)**
  - Typical case: You want to use MKL BLAS and/or LAPACK

```
-L ${INTEL_PATH}/mkl/lib/intel64/ \
-Wl,--start-group \
-lmkl_intel_lp64 -lmkl_sequential -lmkl_core \
-Wl,--end-group
```

  - Another typical case: You want to use MKL serial FFTs/DFTs

    Same as above (need more for FFTW interface)

  - A less typical case: You want to use MKL distributed FFTs

```
-L ${INTEL_PATH}/mkl/lib/intel64/ \
-Wl,--start-group \
-lmkl_cdft_core -lmkl_intel_lp64 -lmkl_sequential \
-lmkl_core -lmkl_blacs_intelmpi_lp64 \
-Wl,--end-group
```

- **The Intel MKL Link Line Advisor can tell you what to add to your link line**
  - http://software.intel.com/sites/products/mkl/

Compiler man pages and documentation

- **For more information on individual compilers**

PrgEnv	C	C++	Fortran
PrgEnv-cray	man craycc	man crayCC	man crayftn
PrgEnv-intel	man icc	man icpc	man ifort
PrgEnv-gnu	man gcc	man g++	man gfortran
Wrappers	man cc	man CC	man ftn

- **To verify that you are using the correct version of a compiler, use:**
  - **-V** option on a cc, CC, or ftn command with CCE and Intel
  - **--version** option on a cc, CC, or ftn command with GNU
- **Cray Reference Manuals:**
  - C and C++: http://docs.cray.com/books/S-2179-81/
  - Fortran: http://docs.cray.com/books/S-3901-81/

Questions?

THE SUPERCOMPUTER COMPANY