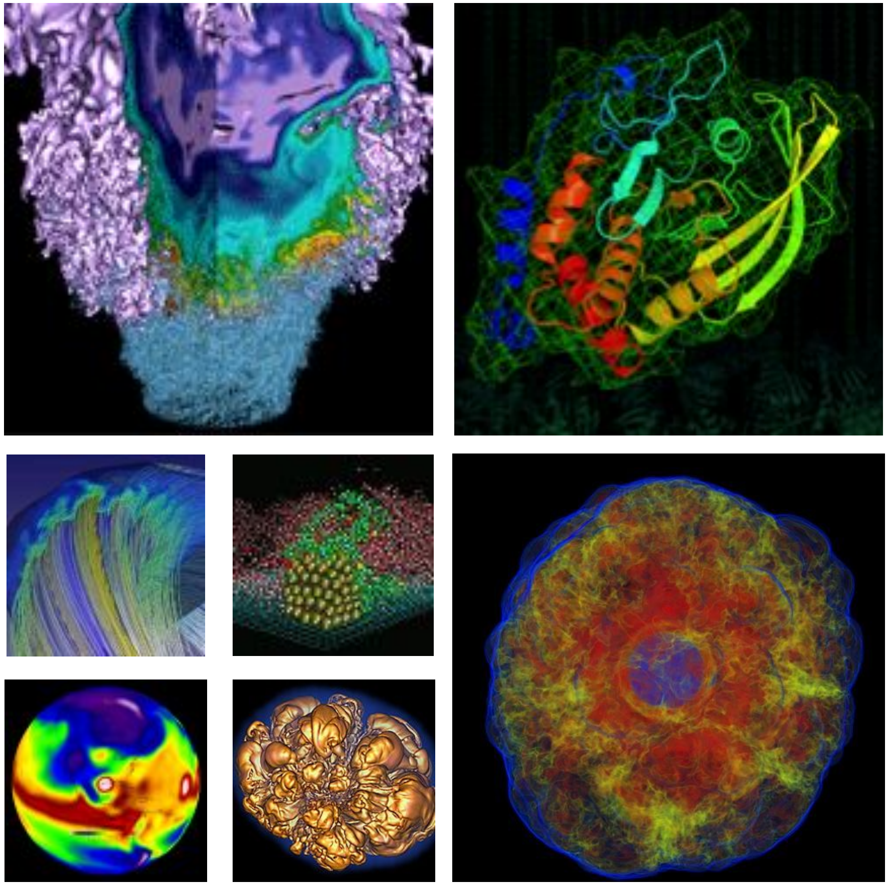
Programming environment and Compilation

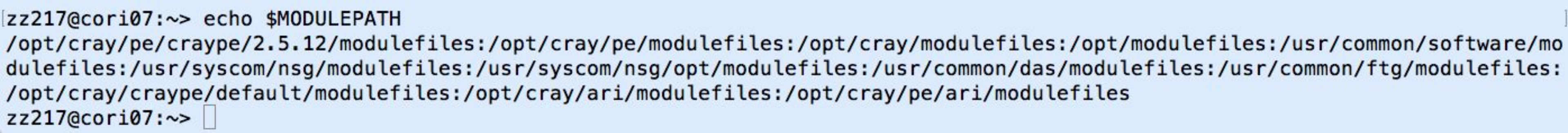


Zhengji Zhao
NERSC User Engagement Group

NERSC New User Training,
March 21, 2018, Berkeley CA

Many pre-compiled applications and libraries are available on Cori and Edison

- **NERSC uses the module utility to manage software packages at NERSC**
  - Dynamically modify the user's shell environment via loading and unload modules, e.g., module load <modulefile>
  - A modulefile contains information to configure the shell environment, such as PATH, LD_LIBRARY_PATH, MANPATH, etc., for an application or a library
- **Use the module avail command to see all available software**
  - module avail lammps
  - module avail –S fftw; module avail -S hdf5
  - MODULEPATH
- **Software modules are provided by Cray and NERSC**



```
zz217@cori07:~> echo $MODULEPATH
/opt/cray/pe/craype/2.5.12/modulefiles:/opt/cray/pe/modulefiles:/opt/cray/modulefiles:/opt/modulefiles:/usr/common/software/mo
dulefiles:/usr/syscom/nsg/modulefiles:/usr/syscom/nsg/opt/modulefiles:/usr/common/das/modulefiles:/usr/common/ftg/modulefiles:
/opt/cray/craype/default/modulefiles:/opt/cray/ari/modulefiles:/opt/cray/pe/ari/modulefiles
zz217@cori07:~>
```

Access to the pre-compiled applications and libraries

- **Most of the pre-compiled applications and libraries are open to all users**
  - module load cray-petsc
  - module load espresso
- **Some of the software have access control**
  - VASP is open to users who have existing license by themselves. Users need to confirm their licenses as described at http://www.nersc.gov/users/software/applications/materials-science/vasp#toc-anchor-2
  - SIESTA is open to researchers at academic or public (non-defense) labs. https://www.nersc.gov/users/software/applications/materials-science/siesta/#toc-anchor-2

This talk will focus on

- Cori (Haswell and KNL nodes) and Edison
- Compile/link lines

  Compiler +

  Compiler Flags +

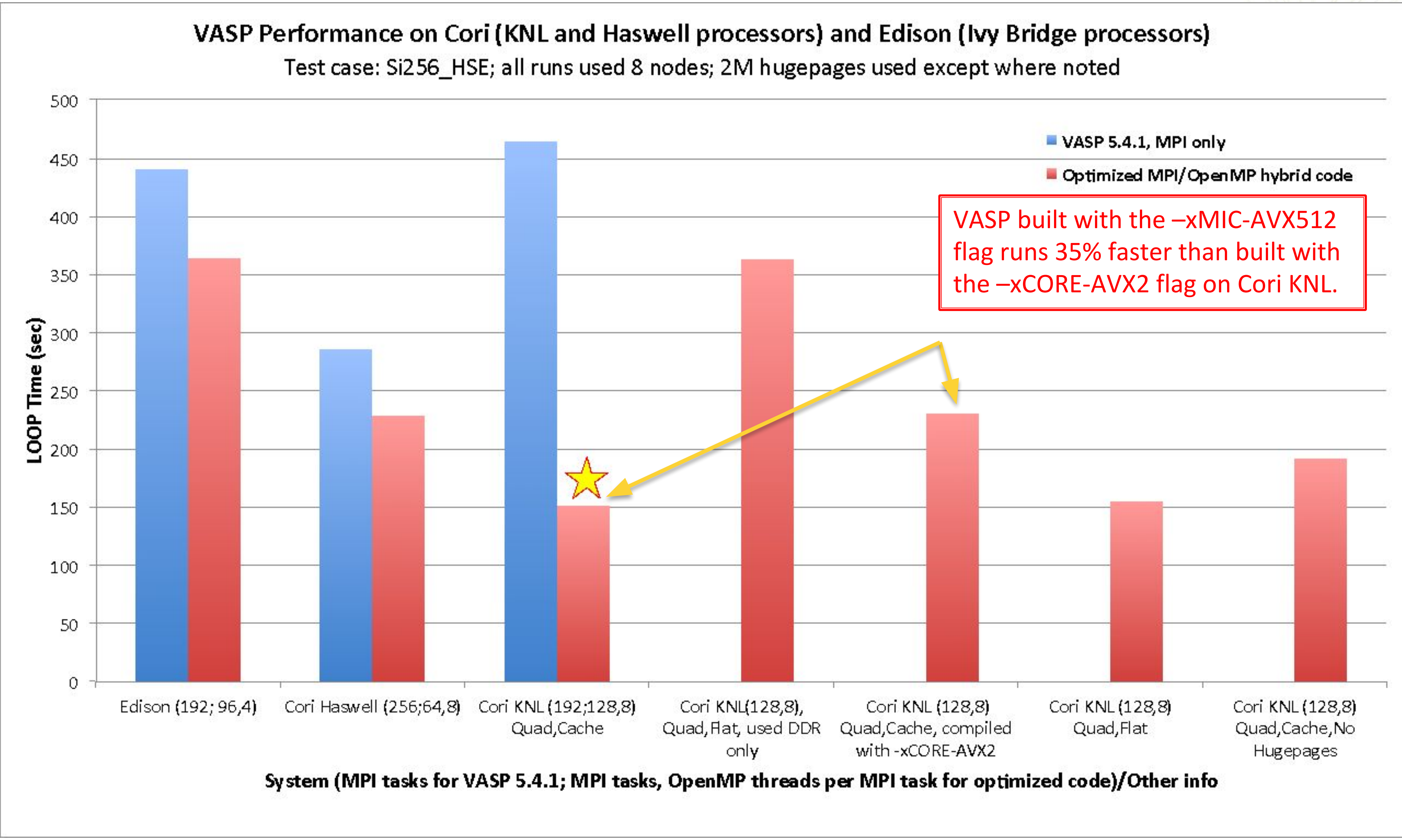
  –I/path/to/headers +

  –L/path/to/library –l<library>

- Available compilers, libraries, how to compile - a couple of tips, compile for Cori KNL, summary

Separate builds for Cori Haswell, KNL and Edison are recommended

- **Cori KNL and Haswell**
  - Cori has 9688 single-socket **Intel® Xeon Phi™ Processor 7250 ("Knights Landing")** nodes @1.4 GHz with 68 cores (272 threads) per node, two 512 bit vector units per core, and 16 GB high bandwidth on-package memory (MCDRAM) with 5X the bandwidth of DDR4 DRAM memory (>400 GB/sec) and 96 GB DDR4 2400 MHz memory per node.
  - In addition, Cori has 2388 dual-socket 16-core **Intel® Xeon™ Processor E5-2698 v3 ("Haswell")** nodes @2.3GHz with 32 cores (64 threads) per node, two 256 bit vector units per core, 128 GB 2133 MHz DDR4 memory.
  - Cori nodes are interconnected with Cray's Aries network with Dragonfly topology.
- **Edison**
  - Edison has 5586 dual-socket 12-core **Intel(R) Xeon(R) CPU E5-2695 v2 ("Ivy Bridge")** nodes @2.40GHz with 24 cores (48 threads) per node, two 256 bit vector units per core, 64 GB DDR3 1866 MHz memory. Edison nodes are interconnected with Cray's Aries network with Dragonfly topology.
- **Compilations on Edison, Cori (Haswell and KNL) are very similar, and binaries can be built to be compatible. However, we recommend a separate build for each platform for optimal performance.**



VASP Performance on Cori (KNL and Haswell processors) and Edison (Ivy Bridge processors)

Test case: Si256_HSE; all runs used 8 nodes; 2M hugepages used except where noted

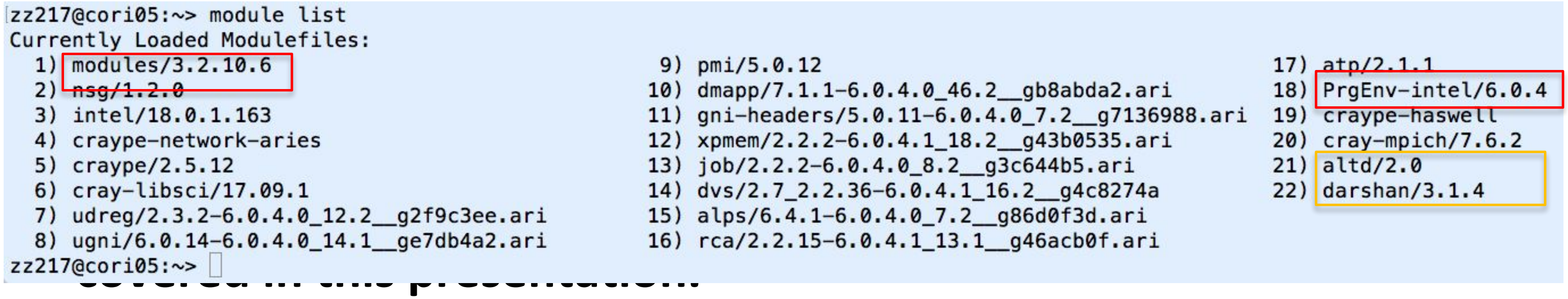
VASP built with the –xMIC-AVX512 flag runs 35% faster than built with the –xCORE-AVX2 flag on Cori KNL.

Building your application separately for each platform could be important to get optimal performance.

Intel, GNU and Cray compilers are available on Cori and Edison

- **Three programming environments are supported**
  - PrgEnv-intel, PrgEnv-gnu, and PrgEnv-cray loads the corresponding programming environment which includes the compilers and matching libraries
  - Intel programming environment is the default on both Cori and Edison.
  - module swap PrgEnv-Intel PrgEnv-cray to swap compilers and programing environment.

Modules loaded by default on Cori – default programming environment:



```
zz217@cori05:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.6                              9) pmi/5.0.12                                17) atp/2.1.1
  2) nsg/1.2.0                                    10) dmapp/7.1.1-6.0.4.0_46.2__gb8abda2.ari     18) PrgEnv-intel/6.0.4
  3) intel/18.0.1.163                             11) gni-headers/5.0.11-6.0.4.0_7.2__g7136988.ari 19) craype-haswell
  4) craype-network-aries                         12) xpmem/2.2.2-6.0.4.1_18.2__g43b0535.ari     20) cray-mpich/7.6.2
  5) craype/2.5.12                                13) job/2.2.2-6.0.4.0_8.2__g3c644b5.ari        21) altd/2.0
  6) cray-libsci/17.09.1                          14) dvs/2.7_2.2.36-6.0.4.1_16.2__g4c8274a      22) darshan/3.1.4
  7) udreg/2.3.2-6.0.4.0_12.2__g2f9c3ee.ari       15) alps/6.4.1-6.0.4.0_7.2__g86d0f3d.ari
  8) ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari       16) rca/2.2.15-6.0.4.1_13.1__g46acb0f.ari
zz217@cori05:~>
```

covered in this presentation.

Use of the compiler wrappers, ftn, cc and CC, is recommended when compiling codes

- **Use ftn, cc, and CC to compile Fortran, C and C++ codes, respectively, instead of the underlying native compilers, such as ifort, icc, icpc, gfortran, gcc, g++, etc.**
  - The compiler wrappers wraps the underlying compilers with the additional compiler and linker flags depending on the modules loaded in the environment
  - The same compiler wrapper command (e.g. ftn) is used to invoke any compilers supported on the system (Intel, GNU, Cray)
- **Compilers wrappers link statically by default**
  - Preferred for performance at scale
- **Use –dynamic or set an environment variable CRAYPE_LINK_TYPE=dynamic to link dynamically**
  - A dynamically linked executable may take a some time to load shared libraries when running with a large number of processes
- **Compiler wrappers do cross compilation**
  - Compiling on login nodes to run on compute nodes
  - Use the –host=x86_64 configure option when compiling for KNL from a login nodes
  - To compile on a KNL node, do salloc –N 1 –q interactive –C knl –t 4:00:00 to get on a compute node

Why compiler wrappers?

- **They include the architecture specific compiler flags into the compilation/link line automatically.**

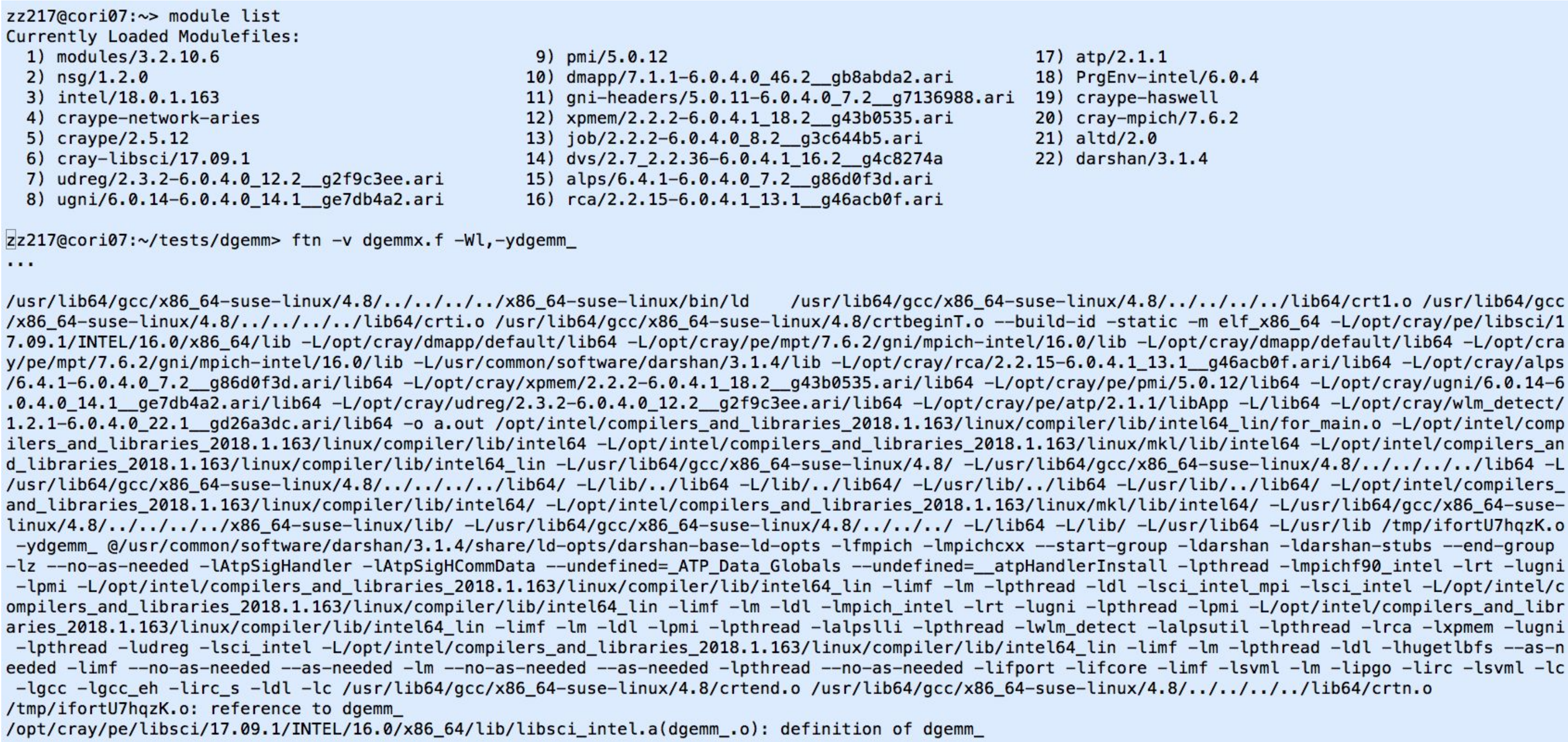
	Intel*)	GNU	Cray	Module
Cori KNL	-xMIC-AVX512	-march=knl	-h cpu=mic-knl	craype-mic-knl
Cori Haswell	-xCORE-AVX2	-march=core-avx2	-h cpu=haswell	craype-haswell
Edison Ivy Bridge	-xCORE-AVX-I	-march=corei7-avx	-h cpu=ivybridge	craype-ivybridge

*) for the latest Intel compilers, -march=knl,haswell,ivybridge can be used instead of –xcode.

- **They automatically add the header and library paths and libraries on the compilation/link lines, so you do not have to explicitly provide them.**
  - Compiler wrappers use the pkg-config tools to dynamically detect paths and libs from the environment (loaded cray modules and some NERSC modules)
  - The architecture specific builds of libraries will be linked into
- **Yet allow user provided options to take the precedence**
  - You may need to remove –xHost option in your compilation/link line

What do compiler wrappers link by default?

- **Depending on the modules loaded, MPI, LAPACK/BLAS/ScaLAPACK libraries, and more**
- **Library names could be different from what you used before**



```
zz217@cori07:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.6                     9) pmi/5.0.12                                  17) atp/2.1.1
  2) nsg/1.2.0                           10) dmapp/7.1.1-6.0.4.0_46.2__gb8abda2.ari      18) PrgEnv-intel/6.0.4
  3) intel/18.0.1.163                    11) gni-headers/5.0.11-6.0.4.0_7.2__g7136988.ari 19) craype-haswell
  4) craype-network-aries                12) xpmem/2.2.2-6.0.4.1_18.2__g43b0535.ari      20) cray-mpich/7.6.2
  5) craype/2.5.12                       13) job/2.2.2-6.0.4.0_8.2__g3c644b5.ari         21) altd/2.0
  6) cray-libsci/17.09.1                 14) dvs/2.7_2.2.36-6.0.4.1_16.2__g4c8274a       22) darshan/3.1.4
  7) udreg/2.3.2-6.0.4.0_12.2__g2f9c3ee.ari 15) alps/6.4.1-6.0.4.0_7.2__g86d0f3d.ari
  8) ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari 16) rca/2.2.15-6.0.4.1_13.1__g46acb0f.ari

zz217@cori07:~/tests/dgemm> ftn -v dgemmx.f -Wl,-ydgemm_
...

/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../x86_64-suse-linux/bin/ld    /usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/crt1.o /usr/lib64/gcc
/x86_64-suse-linux/4.8/../../../../lib64/crti.o /usr/lib64/gcc/x86_64-suse-linux/4.8/crtbeginT.o --build-id -static -m elf_x86_64 -L/opt/cray/pe/libsci/1
7.09.1/INTEL/16.0/x86_64/lib -L/opt/cray/dmapp/default/lib64 -L/opt/cray/pe/mpt/7.6.2/gni/mpich-intel/16.0/lib -L/opt/cray/dmapp/default/lib64 -L/opt/cra
y/pe/mpt/7.6.2/gni/mpich-intel/16.0/lib -L/usr/common/software/darshan/3.1.4/lib -L/opt/cray/rca/2.2.15-6.0.4.1_13.1__g46acb0f.ari/lib64 -L/opt/cray/alps
/6.4.1-6.0.4.0_7.2__g86d0f3d.ari/lib64 -L/opt/cray/xpmem/2.2.2-6.0.4.1_18.2__g43b0535.ari/lib64 -L/opt/cray/pe/pmi/5.0.12/lib64 -L/opt/cray/ugni/6.0.14-6
.0.4.0_14.1__ge7db4a2.ari/lib64 -L/opt/cray/udreg/2.3.2-6.0.4.0_12.2__g2f9c3ee.ari/lib64 -L/opt/cray/pe/atp/2.1.1/libApp -L/lib64 -L/opt/cray/wlm_detect/
1.2.1-6.0.4.0_22.1__gd26a3dc.ari/lib64 -o a.out /opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin/for_main.o -L/opt/intel/comp
ilers_and_libraries_2018.1.163/linux/compiler/lib/intel64 -L/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel64 -L/opt/intel/compilers_an
d_libraries_2018.1.163/linux/compiler/lib/intel64_lin -L/usr/lib64/gcc/x86_64-suse-linux/4.8/ -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64 -L
/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/ -L/lib/../lib64 -L/lib/../lib64/ -L/usr/lib/../lib64 -L/usr/lib/../lib64/ -L/opt/intel/compilers_
and_libraries_2018.1.163/linux/compiler/lib/intel64/ -L/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel64/ -L/usr/lib64/gcc/x86_64-suse-
linux/4.8/../../../../x86_64-suse-linux/lib/ -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../ -L/lib64 -L/lib/ -L/usr/lib64 -L/usr/lib /tmp/ifortU7hqzK.o
 -ydgemm_ @/usr/common/software/darshan/3.1.4/share/ld-opts/darshan-base-ld-opts -lfmpich -lmpichcxx --start-group -ldarshan -ldarshan-stubs --end-group
-lz --no-as-needed -lAtpSigHandler -lAtpSigHCommData --undefined=_ATP_Data_Globals --undefined=__atpHandlerInstall -lpthread -lmpichf90_intel -lrt -lugni
 -lpmi -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin -limf -lm -lpthread -ldl -lsci_intel_mpi -lsci_intel -L/opt/intel/c
ompilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin -limf -lm -ldl -lmpich_intel -lrt -lugni -lpthread -lpmi -L/opt/intel/compilers_and_libr
aries_2018.1.163/linux/compiler/lib/intel64_lin -limf -lm -ldl -lpmi -lpthread -lalpslli -lpthread -lwlm_detect -lalpsutil -lpthread -lrca -lxpmem -lugni
 -lpthread -ludreg -lsci_intel -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin -limf -lm -lpthread -ldl -lhugetlbfs --as-n
eeded -limf --no-as-needed --as-needed -lm --no-as-needed --as-needed -lpthread --no-as-needed -lifport -lifcore -limf -lsvml -lm -lipgo -lirc -lsvml -lc
 -lgcc -lgcc_eh -lirc_s -ldl -lc /usr/lib64/gcc/x86_64-suse-linux/4.8/crtend.o /usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/crtn.o
/tmp/ifortU7hqzK.o: reference to dgemm_
/opt/cray/pe/libsci/17.09.1/INTEL/16.0/x86_64/lib/libsci_intel.a(dgemm_.o): definition of dgemm_
```

Compiler recommendations

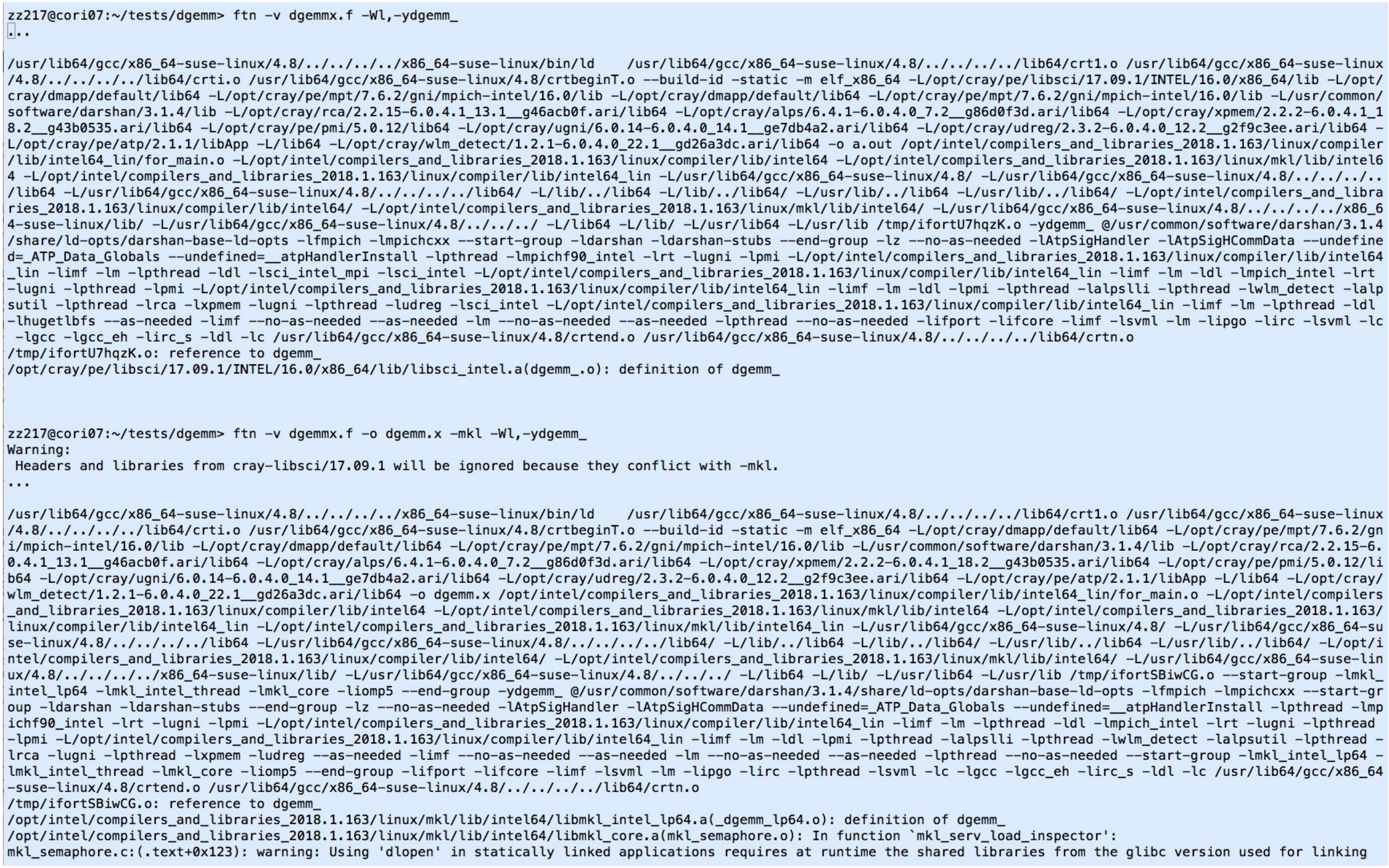
- **Will not recommend any specific compiler**
  - Intel - better chance of getting processor specific optimizations, especially for KNL
  - Cray compiler – many new features and optimizations, especially with Fortran
  - GNU - widely used by open software
- **Start with the compilers that vendor/code developers used so to minimize the chance to hit the compiler and code bugs, then explore different compilers if you care the performance.**

Compiler flags

Intel	GNU	Cray	Description/ Comment
-O2	-O0	-O2	default
default , or –O3	-O2 or -O3,-Ofast	default	recommended
-qopenmp	-fopenmp	default, or –h omp	OpenMP
-g	-g	-g*	debug
-v	-v	-v	verbose

- **Validity check after compilation**
  - Run tests and check with the references if provided
  - When higher optimizations used, compare with the debug version to check the validity
- **Compilers' default behavior could vary between compilers**
  - Default number of OpenMP threads used is all CPU slots available for Intel and GNU compilers; 1 for Cray compiler.
  - * -g: Cray compiler disables all optimizations, try –G2;

Verbose output from compiler wrappers contains many useful information



```
zz217@cori07:~/tests/dgemm> ftn -v dgemmx.f -Wl,-ydgemm_
...

/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../x86_64-suse-linux/bin/ld    /usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/crt1.o /usr/lib64/gcc/x86_64-suse-linux
/4.8/../../../../lib64/crti.o /usr/lib64/gcc/x86_64-suse-linux/4.8/crtbeginT.o --build-id -static -m elf_x86_64 -L/opt/cray/pe/libsci/17.09.1/INTEL/16.0/x86_64/lib -L/opt/
cray/dmapp/default/lib64 -L/opt/cray/pe/mpt/7.6.2/gni/mpich-intel/16.0/lib -L/opt/cray/dmapp/default/lib64 -L/opt/cray/pe/mpt/7.6.2/gni/mpich-intel/16.0/lib -L/usr/common/
software/darshan/3.1.4/lib -L/opt/cray/rca/2.2.15-6.0.4.1_13.1__g46acb0f.ari/lib64 -L/opt/cray/alps/6.4.1-6.0.4.0_7.2__g86d0f3d.ari/lib64 -L/opt/cray/xpmem/2.2.2-6.0.4.1_1
8.2__g43b0535.ari/lib64 -L/opt/cray/pe/pmi/5.0.12/lib64 -L/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64 -L/opt/cray/udreg/2.3.2-6.0.4.0_12.2__g2f9c3ee.ari/lib64 -
L/opt/cray/pe/atp/2.1.1/libApp -L/lib64 -L/opt/cray/wlm_detect/1.2.1-6.0.4.0_22.1__gd26a3dc.ari/lib64 -o a.out /opt/intel/compilers_and_libraries_2018.1.163/linux/compiler
/lib/intel64_lin/for_main.o -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64 -L/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel6
4 -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin -L/usr/lib64/gcc/x86_64-suse-linux/4.8/ -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../..
/lib64 -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/ -L/lib/../lib64 -L/lib/../lib64/ -L/usr/lib/../lib64 -L/usr/lib/../lib64/ -L/opt/intel/compilers_and_libra
ries_2018.1.163/linux/compiler/lib/intel64/ -L/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel64/ -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../x86_6
4-suse-linux/lib/ -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../ -L/lib64 -L/lib/ -L/usr/lib64 -L/usr/lib /tmp/ifortU7hqzK.o -ydgemm_ @/usr/common/software/darshan/3.1.4
/share/ld-opts/darshan-base-ld-opts -lfmpich -lmpichcxx --start-group -ldarshan -ldarshan-stubs --end-group -lz --no-as-needed -lAtpSigHandler -lAtpSigHCommData --undefine
d=_ATP_Data_Globals --undefined=__atpHandlerInstall -lpthread -lmpichf90_intel -lrt -lugni -lpmi -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64
_lin -limf -lm -lpthread -ldl -lsci_intel_mpi -lsci_intel -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin -limf -lm -ldl -lmpich_intel -lrt
-lugni -lpthread -lpmi -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin -limf -lm -ldl -lpmi -lpthread -lalpslli -lpthread -lwlm_detect -lalp
sutil -lpthread -lrca -lxpmem -lugni -lpthread -ludreg -lsci_intel -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin -limf -lm -lpthread -ldl
-lhugetlbfs --as-needed -limf --no-as-needed --as-needed -lm --no-as-needed --as-needed -lpthread --no-as-needed -lifport -lifcore -limf -lsvml -lm -lipgo -lirc -lsvml -lc
 -lgcc -lgcc_eh -lirc_s -ldl -lc /usr/lib64/gcc/x86_64-suse-linux/4.8/crtend.o /usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/crtn.o
/tmp/ifortU7hqzK.o: reference to dgemm_
/opt/cray/pe/libsci/17.09.1/INTEL/16.0/x86_64/lib/libsci_intel.a(dgemm_.o): definition of dgemm_



zz217@cori07:~/tests/dgemm> ftn -v dgemmx.f -o dgemm.x -mkl -Wl,-ydgemm_
Warning:
 Headers and libraries from cray-libsci/17.09.1 will be ignored because they conflict with -mkl.
...

/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../x86_64-suse-linux/bin/ld    /usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/crt1.o /usr/lib64/gcc/x86_64-suse-linux
/4.8/../../../../lib64/crti.o /usr/lib64/gcc/x86_64-suse-linux/4.8/crtbeginT.o --build-id -static -m elf_x86_64 -L/opt/cray/dmapp/default/lib64 -L/opt/cray/pe/mpt/7.6.2/gn
i/mpich-intel/16.0/lib -L/opt/cray/dmapp/default/lib64 -L/opt/cray/pe/mpt/7.6.2/gni/mpich-intel/16.0/lib -L/usr/common/software/darshan/3.1.4/lib -L/opt/cray/rca/2.2.15-6.
0.4.1_13.1__g46acb0f.ari/lib64 -L/opt/cray/alps/6.4.1-6.0.4.0_7.2__g86d0f3d.ari/lib64 -L/opt/cray/xpmem/2.2.2-6.0.4.1_18.2__g43b0535.ari/lib64 -L/opt/cray/pe/pmi/5.0.12/li
b64 -L/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64 -L/opt/cray/udreg/2.3.2-6.0.4.0_12.2__g2f9c3ee.ari/lib64 -L/opt/cray/pe/atp/2.1.1/libApp -L/lib64 -L/opt/cray/
wlm_detect/1.2.1-6.0.4.0_22.1__gd26a3dc.ari/lib64 -o dgemm.x /opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin/for_main.o -L/opt/intel/compilers
_and_libraries_2018.1.163/linux/compiler/lib/intel64 -L/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel64 -L/opt/intel/compilers_and_libraries_2018.1.163/
linux/compiler/lib/intel64_lin -L/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel64_lin -L/usr/lib64/gcc/x86_64-suse-linux/4.8/ -L/usr/lib64/gcc/x86_64-su
se-linux/4.8/../../../../lib64 -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/ -L/lib/../lib64 -L/lib/../lib64/ -L/usr/lib/../lib64 -L/usr/lib/../lib64/ -L/opt/i
ntel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64/ -L/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel64/ -L/usr/lib64/gcc/x86_64-suse-lin
ux/4.8/../../../../x86_64-suse-linux/lib/ -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../ -L/lib64 -L/lib/ -L/usr/lib64 -L/usr/lib /tmp/ifortSBiwCG.o --start-group -lmkl_
intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 --end-group -ydgemm_ @/usr/common/software/darshan/3.1.4/share/ld-opts/darshan-base-ld-opts -lfmpich -lmpichcxx --start-gr
oup -ldarshan -ldarshan-stubs --end-group -lz --no-as-needed -lAtpSigHandler -lAtpSigHCommData --undefined=_ATP_Data_Globals --undefined=__atpHandlerInstall -lpthread -lmp
ichf90_intel -lrt -lugni -lpmi -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin -limf -lm -lpthread -ldl -lmpich_intel -lrt -lugni -lpthread
-lpmi -L/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64_lin -limf -lm -ldl -lpmi -lpthread -lalpslli -lpthread -lwlm_detect -lalpsutil -lpthread -
lrca -lugni -lpthread -lxpmem -ludreg --as-needed -limf --no-as-needed --as-needed -lm --no-as-needed --as-needed -lpthread --no-as-needed --start-group -lmkl_intel_lp64 -
lmkl_intel_thread -lmkl_core -liomp5 --end-group -lifport -lifcore -limf -lsvml -lm -lipgo -lirc -lpthread -lsvml -lc -lgcc -lgcc_eh -lirc_s -ldl -lc /usr/lib64/gcc/x86_64
-suse-linux/4.8/crtend.o /usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/crtn.o
/tmp/ifortSBiwCG.o: reference to dgemm_
/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel64/libmkl_intel_lp64.a(_dgemm_lp64.o): definition of dgemm_
/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel64/libmkl_core.a(mkl_semaphore.o): In function `mkl_serv_load_inspector':
mkl_semaphore.c:(.text+0x123): warning: Using 'dlopen' in statically linked applications requires at runtime the shared libraries from the glibc version used for linking
```

Available libraries

- **Cray supports many software packages – Cray Developer Took kits (CDT)**
  - Access via modules, type “module avail” or “module avail –S” to see the available modules
  - There are different builds for different compilers
  - Programming environment modules allow the libraries built with the matching compilers to be linked to
- **NERSC also supports many libraries**
  - Many of them interact with the Cray compiler wrappers while some of them do not.
- **Where are the libraries ?**
  - Use “module show <module name> “ to see the installation paths

More examples of module commands:

```
module avail netcdf; module avail –S netcdf
module list
Module load cray-petsc ; module unload cray-petsc; module load cdt/18.03
Module swap PrgEnv-intel PrgEnv-gnu;
Module help cray-tpsl
```

Examples of linking to the Cray provided libraries using compiler wrappers

- **Linking to Cray MPI and Cray Scientific libraries are automatic if compiler wrappers are used**

  ```
  CC parallel_hello.cpp
  ftn dgemmx1.f90
  ```

- **Linking to HDF5 and NETCDF libraries are automatic, user just need to load the cray-hdf5 or cray-netcdf modules**

  ```
  module load cray-hdf5
  cc h5write.c
  ```

  - Note The library name could be different. Using the –v option to see the library names and other details link line information.
- **Linking to PETSc libraries are automatic, but users need to choose a proper module (real/complex,32/64 bit integer)**
  - E.g., module load cray-petsc-complex-64
  - Use cc –v test1.c to see the linking detail
- **Linking to fftw libraries – fftw 3 are now the default**
  - module load cray-fftw
  - Loading the cray-fftw module always links to the pthread version of the library, -lfftw3f_mpi -lfftw3f_threads -lfftw3f -lfftw3_mpi -lfftw3_threads -lfftw3, to link with OpenMP implementation, need to manually provide the libraries.

Examples of linking to the NERSC provided library modules

- **Some of the NERSC provided modulefiles are written to be recognized by the compiler wrappers, e.g., elpa module on Cori**

  ```
  module load elpa
  ftn –qopenmp –v test2.f90   # this  will automatically link to elpa and MKL ScaLAPACK libraries
  ```

  – Type module show <module name> to check if the envs <libname>_PKGCONFIG_LIBS, PE_PKGCONFIG_PRODUCTS, and PKG_CONFIG_PATH are defined in the modulefile, which compiler wrappers look for.

- **Most of the NERSC provided modulefiles do not interact with the compiler wrappers, user need to provide the include path and library path and libraries manually, e.g. GSL**

  ```
  Module load gsl
  ftn test3.f90 $GSL
  ```

  – GSL is set as -I/usr/common/software/gsl/2.1/intel/include -L/usr/common/software/gsl/2.1/intel/lib -lgsl -lgslcblas

Linking to Intel MKL library

- **Resource:**
  - Intel® Math Kernel Library Link Line Advisor, https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/
  - Learn from Intel compiler verbose output, -mkl={parallel,sequential,cluster}
- **For intel compiler, use –mkl flag**
  - ftn test1.f90 –mkl # default to parallel –multi-threaded lib
  - The loaded cray-libsci will be ignored if –mkl is used.
- **For GNU compiler (e.g., to link to 32-bit integer build):**
  - Save the MKLROOT from the Intel compiler module, and then
  - Threaded: -L$MKLROOT/lib/intel64 –Wl,--start-group -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -liomp5 -Wl,--end-group –lpthread –lm –ldl
  - ScaLAPACK: -L$MKLROOT/lib/intel64 -Wl,--start-group -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_scalapack_lp64 -lmkl_blacs_intelmpi_lp64 -lmkl_core -Wl,--end-group -lgomp –lpthread –lm -ldl
  - Note that mkl modules could be out-dated

Dynamic linking examples with compiler wrappers

- **Some of the libraries are available only in shared library format, e.g., TBB, use –dynamic in the link line or use CRAYPE_LINK_TYPE=dynamic**
- **Linking to Intel TBB library**

```
module load tbb
CC –dynamic test4.C –tbb
```

or

```
module load tbb
Export CRAYPE_LINK_TYPE=dynamic
CC test4.C –tbb
```

Linking to Intel MPI library – Use native compilers

- **Cray MPICH libraries are recommended for performance especially at scale.**
- **Compiler wrappers links to Cray MPICH libraries.**
- **However, if you need to link to Intel MPI library, do**

```
Module load impi
mpiifort test1.f90
```

  - Note that the binaries linked to the Intel MPI need to run with srun instead of mpirun to get a proper process/thread affinity, http://www.nersc.gov/users/computational-systems/cori/running-jobs/advanced-running-jobs-options/#toc-anchor-6
  - Native Intel compilers link dynamically

Resolving missing libraries and unresolved symbols

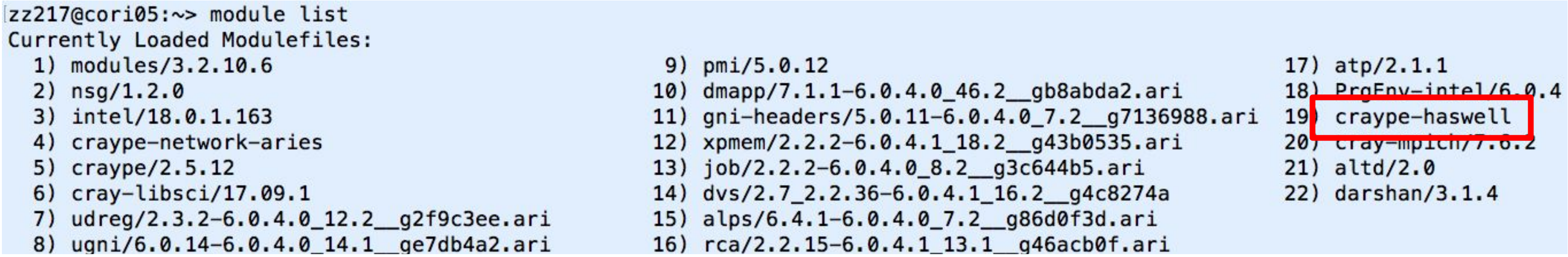
- **Only shared libraries available**
  - E.g., /usr/bin/ld: cannot find –ldl
  - Try -dynamic
- **Static library linking order, try -Wl,--start-group … -Wl,--end-group, e.g.,**

```
LLIBS     =     -Wl,--start-group $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a \
                $(MKLROOT)/lib/intel64/libmkl_scalapack_lp64.a
$(MKLROOT)/lib/intel64/libmkl_blacs_intelmpi_lp64.a \
                $(MKLROOT)/lib/intel64/libmkl_intel_thread.a $(MKLROOT)/lib/intel64/libmkl_core.a
-Wl,--end-group \
                /usr/common/software/wannier90/1.2/knl/intel/lib/libwannier.a \
                -Wl,-zmuldefs
```

- **Use grep –R and readelf –s |grep <symbol> to search for the unresolved symbols.**

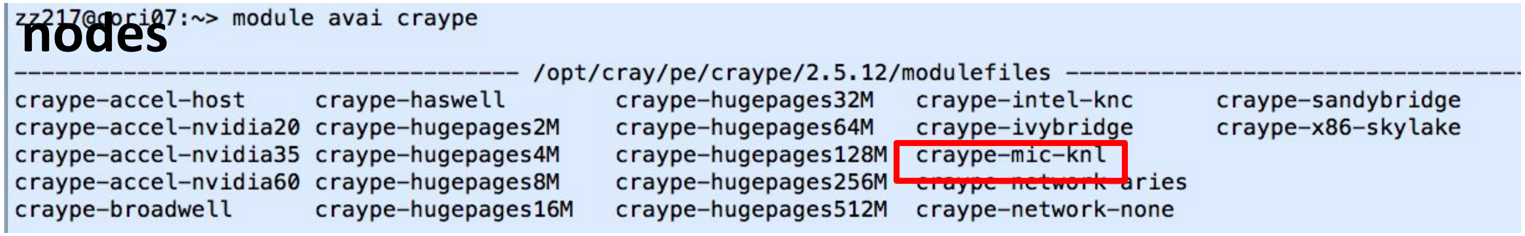
How to compile for Cori KNL

- **The craype-* module sets the target that the compiler wrappers (cc, CC, ftn) build for**
  - craype-haswell: -xCORE-AVX2 (Intel compiler);-mhaswell (GNU); -hhaswell (Cray)
  - craype-mic-knl: -xMIC-AVX512 (Intel compiler); `−mknl` (GNU); `−hmic−knl` (Cray)
- **craype-haswell is default on login nodes**



```
zz217@cori05:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.6                    9) pmi/5.0.12                                17) atp/2.1.1
  2) nsg/1.2.0                          10) dmapp/7.1.1-6.0.4.0_46.2__gb8abda2.ari    18) PrgEnv-intel/6.0.4
  3) intel/18.0.1.163                   11) gni-headers/5.0.11-6.0.4.0_7.2__g7136988.ari 19) craype-haswell
  4) craype-network-aries               12) xpmem/2.2.2-6.0.4.1_18.2__g43b0535.ari    20) cray-mpich/7.6.2
  5) craype/2.5.12                      13) job/2.2.2-6.0.4.0_8.2__g3c644b5.ari       21) altd/2.0
  6) cray-libsci/17.09.1                14) dvs/2.7_2.2.36-6.0.4.1_16.2__g4c8274a     22) darshan/3.1.4
  7) udreg/2.3.2-6.0.4.0_12.2__g2f9c3ee.ari 15) alps/6.4.1-6.0.4.0_7.2__g86d0f3d.ari
  8) ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari 16) rca/2.2.15-6.0.4.1_13.1__g46acb0f.ari
```

- **craype-mic-knl is for KNL nodes, JUST do module swap craype-haswell craype-mic-knl before compiling for Cori KNL nodes**



```
zz217@cori07:~> module avai craype

------------------------------------ /opt/cray/pe/craype/2.5.12/modulefiles ------------------------------------
craype-accel-host      craype-haswell         craype-hugepages32M    craype-intel-knc       craype-sandybridge
craype-accel-nvidia20  craype-hugepages2M     craype-hugepages64M    craype-ivybridge       craype-x86-skylake
craype-accel-nvidia35  craype-hugepages4M     craype-hugepages128M   craype-mic-knl
craype-accel-nvidia60  craype-hugepages8M     craype-hugepages256M   craype-network-aries
craype-broadwell       craype-hugepages16M    craype-hugepages512M   craype-network-none
```

How to compile for Cori KNL

Best: Compiler settings to target KNL

Alternate:

```
CC -axMIC-AVX512,CORE-AVX2 <more-options> mycode.c++
```

- **Only valid when using Intel compilers (cc, CC or ftn)**
- **-ax<arch> adds an "alternate execution paths" optimized for different architectures**
  - Makes 2 (or more) versions of code in same object file
- **NOT AS GOOD as the craype-mic-knl module**
  - (module causes versions of libraries built for that architecture to be used – e.g. MKL)

How to compile for Cori KNL

Recommendations:

- **For best performance, use the craype-mic-knl module**

```
 module swap craype-haswell craype-mic-knl
CC -O3 -c myfile.c++
```

- **If the same executable must run on KNL *and* Haswell nodes, use craype-haswell but add KNL-optimized execution path**

```
 CC -axMIC-AVX512,CORE-AVX2 -O3 -c myfile.c++
```

What to link

Utility libraries

- **Not performance-critical (by definition)**
  - KNL can run Xeon binaries .. can use Haswell-targeted versions
- **I/O libraries (HDF5, NetCDF, etc) should fit in this category too**
  - (for Cray-provided libraries, compiler wrapper will use craype-* to select best build anyway)

What to link

Performance-critical libraries

- **MKL: has KNL-targeted optimizations**
- **PETSc, SLEPc, Caffe, Metis, etc:**
  - (soon) has KNL-targeted builds
- **Modulefiles will use craype-{haswell,mic-knl} to find appropriate library**
  - Currently FFTW,LibSci,PETSc, TPSL have separate builds for KNL
- **Key points:**
  - Someone else has already prepared libraries for KNL
  - No need to do-it-yourself
  - Load the right craype- module

Saving and restoring the programming environment on Cori and Edison

- **Module snapshot - capture the currently loaded module environment for later restore.**
  - module snapshot [-f|--force] filename
  - module restore filename
  - The filename is saved at ${HOME}/.module_snapshots by default; use MODULE_SNAPSHOT_DIR for alternative location
- **cdt modules – allows a clean switch to previous CDT version**
- **Using shifter - preserve both system and user software**
  - Shifter is a software package that allows user-created images to run at NERSC. More info at http://www.nersc.gov/users/software/using-shifter-and-docker/using-shifter-at-nersc/
- **ALTD can provide your binary linking info**
  - https://my.nersc.gov/libraryusage-cs.php
  - link_info.sh /path/to/your/binary/<binary name>

Frequency of Programming Environment Software Default Changes

- **CDT Update Policy:**
  - New software will be installed every 3 months, usually in December, March, September, and June. The new versions will not be made the defaults when installed.
  - New software defaults will be set twice a year: Once in January at the Allocation Year Rollover (the previous year's September release) and once in June (the March release). The actual versions may vary subject to software verification and security or urgent fixes.
  - Only three CDT versions, the current default, the previous default, and the newest, will be made available on the system at any given time. If you need a CDT version that we have removed from the system, please contact consult at nersc.gov. Note: Cray CDT (Cray Developer Toolkit) contains Programming Environment Software release including Cray compiling, message passing interface, performance, debugging, third party libraries, etc. Note some of the software packages may have more than three versions unintentionally. We will delete the extra versions in next maintenance.
- **Timelines:**
  - Cori, https://www.nersc.gov/users/computational-systems/cori/cori-timeline/
  - Edison, https://www.nersc.gov/users/computational-systems/edison/updates-and-status/timeline-and-updates/

Sharing your software installations with others

- **Install your applications/libraries in the /global/common/software/<your repo> directory**
  - This is a read-only file system on compute nodes.
  - Shared library applications may get performance benefit.
- **Add a modulepath to your environment,**
  - module use /global/common/software/<your repo>/modulefiles, or export MODULEPATH= /global/common/software/<your repo>/modulefiles:$MODULEPATH
- **Write a modulefile for your application or library**
- **Use the craypkg-gen module to generate a modulefile for your library installation so that compiler wrappers can automatically pickup the paths and libraries**

  module load craypkg-gen

  craypkg-gen -p installation_dir   to generate the package config file

  craypkg-gen –m installation_dir –o /global/common/software/<your repo> to generate the modulefile that can interact with the compiler wrappers
- **Use env SITE_MODULE_NAMES for library modules to swap modules PrgEnv-* modules switch.**
  - Type “module show darshan” for an example

Using Spack to install your software

- **Spack is a flexible package manager that builds and installs multiple versions and configurations of software.**
- **NERSC is transition to using Spack to build our software**
  - Most of the NERSC staff supported software on Edison have been bbuilt with Spack
  - Cori will be next to use Spack
- **Try to install your software, especially the dependent software with Spack - ask NERSC consultant for help**
  - git clone https://github.com/spack/spack.git
  - Spack list
  - Spack install <package-name>
  - Tutorial: http://spack.readthedocs.io/en/latest/getting_started.html

Summary

- **Use compiler wrappers where possible, which allows to include architecture specific optimization flag and link to the libraries**
  - Check for the –xHost option
- **Explore available compilers, Intel, Cray, GNU**
  - Validity check after builds is important
- **Use provided libraries where available**
  - Use module show <module name> to see the paths if needed

Summary -continued

- **For Cori KNL, do**
  - `module swap craype-haswell craype-mic-knl`
- **Build on login nodes whenever possible**
- **Learn from the compiler verbose output**
- **Read compiler and linker man pages**
- **Use module snapshots, shifter to preserve and restore your build environment**
- **Consider creating a modulefile for your software install for yourself and to share with others.**
- **Try out Spack**

Recommended readings

- **NERSC website, especially,**
  - http://www.nersc.gov/users/computational-systems/cori/programming/compiling-codes-on-cori/
  - http://www.nersc.gov/users/computational-systems/edison/programming/
  - We are moving user documentation pages to http://docs.nersc.gov,
  - https://docs.nersc.gov/development/compilers/
  - For further compiler optimizations read intel slides: e.g., https://www.nersc.gov/users/training/events/intel-compilers-tools-and-libraries-training-march-6-2018/
- **Compiler and linker man pages:**
  - ifort, icc, icpc, crayftn,etc
  - man ld (-Wl,-zmuldefs, -Wl,-y<symbol>)
- **To save and restore build environment:**
  - man module, subcommand snapshot/restore; also http://www.nersc.gov/users/software/nersc-user-environment/modules/#toc-anchor-3
  - Shifter, http://www.nersc.gov/users/software/using-shifter-and-docker/using-shifter-at-nersc/

Acknowledgement

- Steve Leak for providing some of the slides used in this talk.

Thank you!