The Effects of Compiler Optimizations on Materials Science and Chemistry Applications at NERSC

Megan Bowling, Zhengji Zhao, and Jack Deslippe

U.S. DEPARTMENT OF ENERGY | Office of Science

NERSC National Energy Research Scientific Computing Center

BERKELEY LAB Lawrence Berkeley National Laboratory

Motivation

- NERSC provides a large number of application codes to improve scientific productivity of users
  - materials science and chemistry applications consumes 1/3 of computing cycles at NERSC each year
  - More than 1200 number of unique users (1/1/2011-5/1/2012) for more than 15 precompiled applications
- Applications can have large performance difference depending on compilers and libraries used
  - User reports, staff tests, …
- Our applications are compiled mainly with PGI compiler for Fortran codes, and GNU for C/C++ codes without confirming if they are the optimal compilers for each specific application.
- Optimization across compilers and libraries without modifying source codes is a low effort optimization

How much of a performance gain could this low-effort optimization bring to NERSC users?

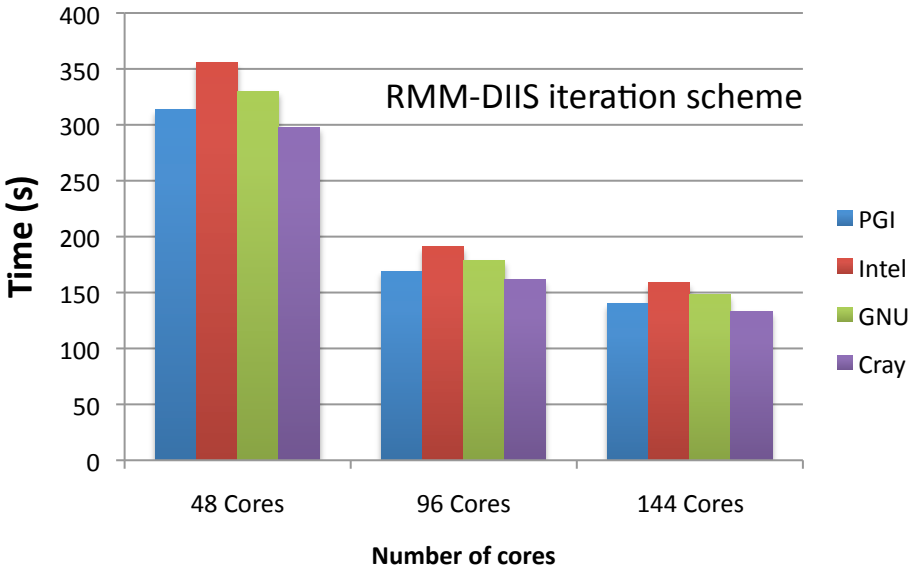
Different Compiler and Library Options Explored on Hopper

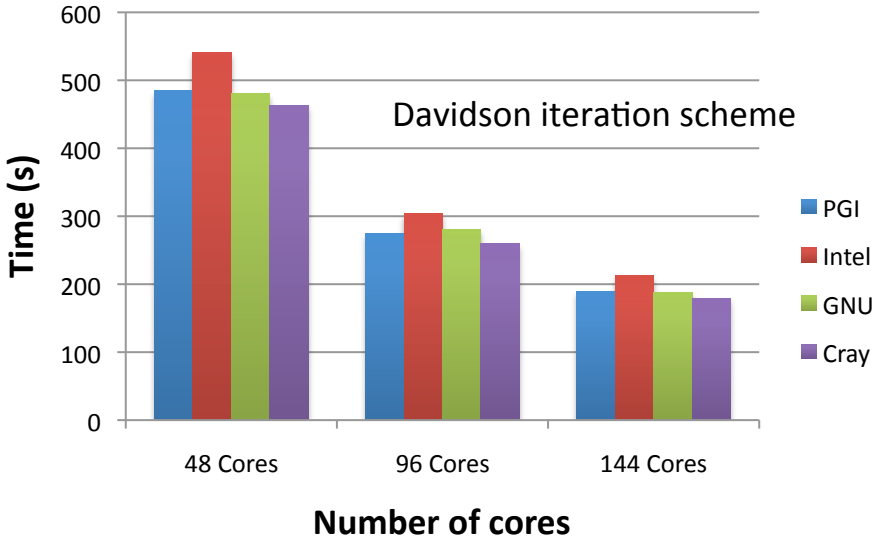
- Hopper – NERSC's peta-flop Cray XE6 System, 153,216 cores, and 6,384 nodes with 24 cores per node
- Compilers available on Hopper
  - PGI
  - GNU
  - Intel
  - Pathscale, did not test due to limited support from Cray on Hopper
  - Cray, failed to build and/or run some of the applications tested
- Libraries: Libsci, ACML, FFTW2, FFTW3
- Compiled codes on Hopper, and run on Grace -Hopper test system to reduce the runtime fluctuation due to other users on the system
  - Grace has 12 nodes, 288 cores
  - Close-to-dedicated machine
- Applications experimented:
  - VASP, QE, LAMMPS, NAMD, NWCHEM, BerkeleyGW

VASP (5.2.12)

- Program Description
  - VASP is a Fortran code that performs atomic scale materials modeling.
- Options explored
  - Compilers and optimization flags used
    - PGI: -fastsse, -O3, -Mvect
    - Intel: -O3, -fast
    - GNU: -O3, -ffast-math
    - Cray: -O –ipa0
  - Libraries: LibSci, ACML
- Tested with 3 test cases
  - Algorithms: DIIS-RMM, Davidson, Hybrid
  - Concurrencies: 48, 96, 144; 384,768; 48,72

Cray compiler outperforms other compilers with medium sized VASP runs





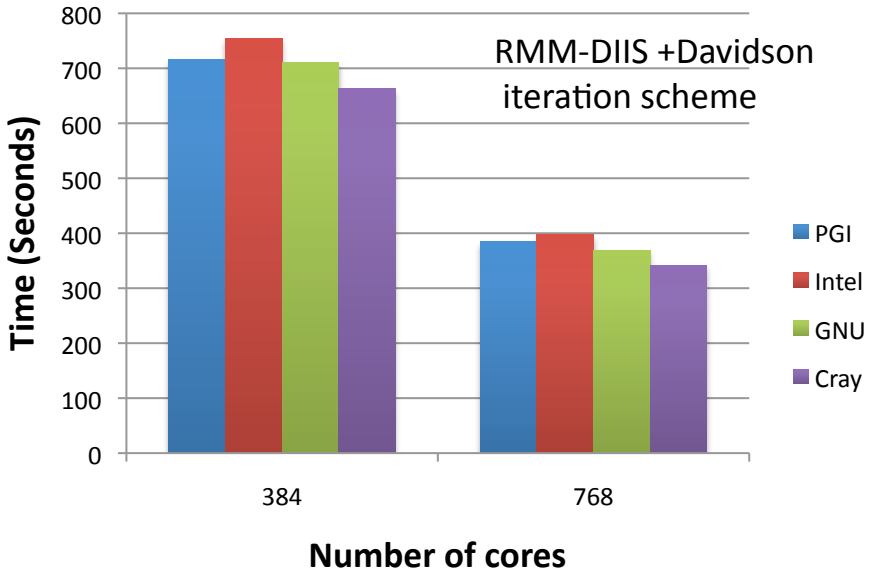
Test case 1:

- NERSC user provided test case:
- A 155 atom system
- The time to complete first 20 electronic steps were measured

Compiler	Performance gain relative to PGI compiler (%)
Intel	-12%
GNU	-6% ~ +1%
PGI	default
Cray	5.8%

VASP runs faster by 5.8% when switching to Cray compiler.

Cray compiler outperforms other compilers for larger test cases



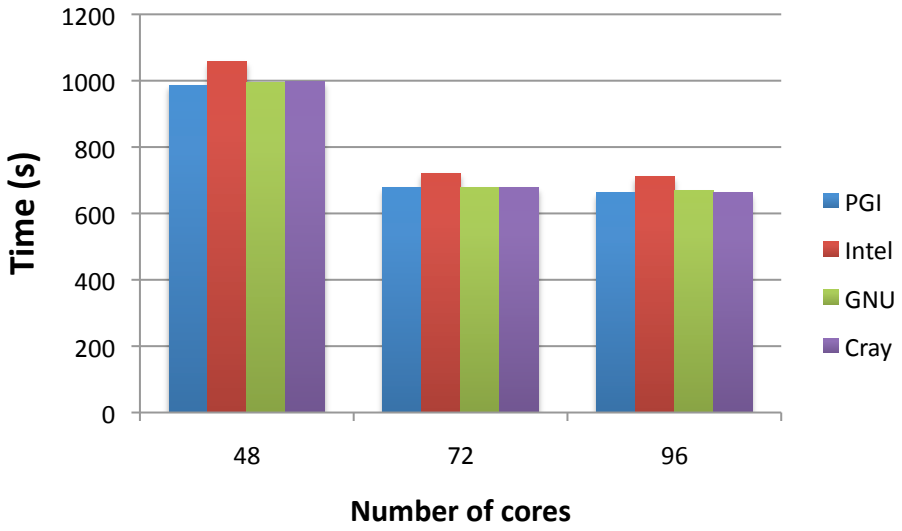
Test case 2

- NERSC user provided
- A 660 atom system
- Time for first 4 electronic steps

Compiler	Faster than the default compiler by (%)
Intel	-5%
PGI	default
GNU	4%
Cray	11%

VASP with Cray compiler runs faster by up to 11% for the larger test case.

Compiler performance varies depending on job types



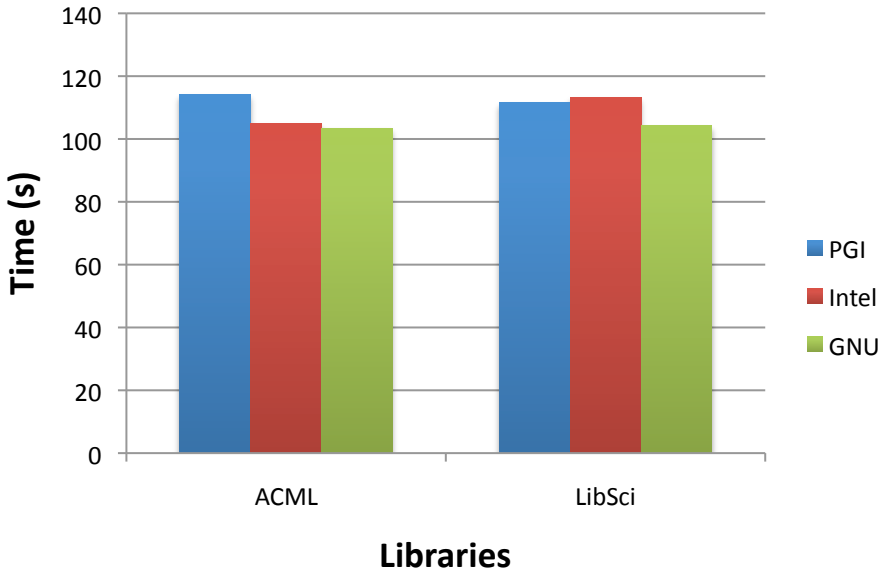
Test case 3

- Provided by NERSC users
- Hybrid calculation for a 105 atom system

Compiler	Faster than PGI compiler by (%)
Intel	-6%
Cray, GNU	0%
PGI	default

VASP with Cray compiler runs at the same speed as PGI compiler for the hybrid jobs

ACML performs slightly better than LibSci with Intel and GNU compilers with VASP



Compiler	Performance gain of (Compiler + ACML) relative to (the same Compiler + LibSci)
PGI +ACML	-2.2%
Intel +ACML	7.3%
GNU +ACML	0.8%
Cray+ACML	N/A

Test case 1:

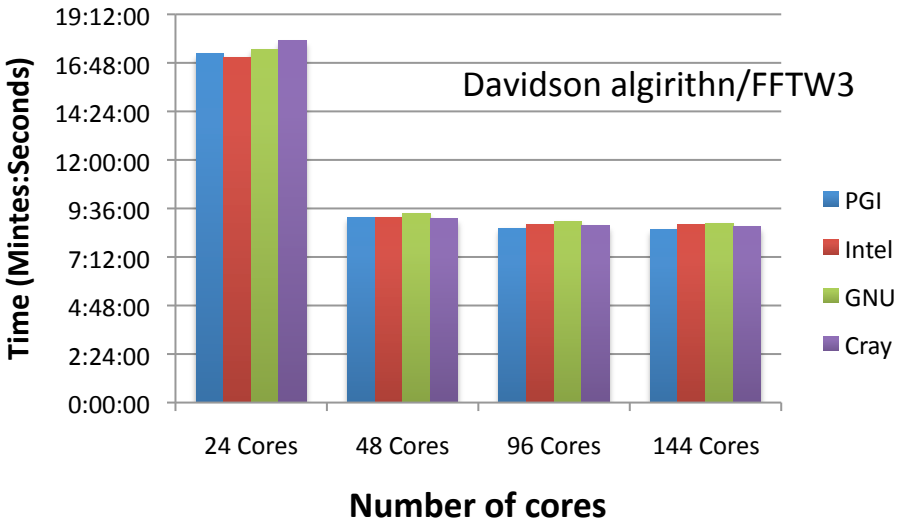
- NERSC user provided test case:
- A 155 atom system
- The time to complete first 4 electronic steps were measured

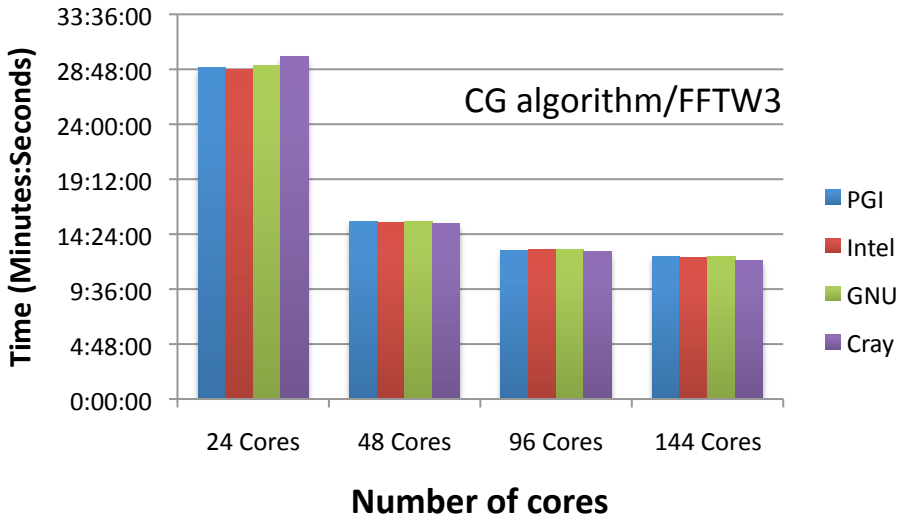
VASP runs slightly faster when it is compiled with the combination of Intel compiler and ACML library.

Quantum Espresso (4.3.2)

- Program Description
  - QE is a hybrid MPI/OpenMP Fortran code that performs atomistic simulations based on electronic structure.
- Options explored:
  - Compilers and optimization flags used
    - PGI: -fast -O3
    - Intel: -O3
    - GNU: -O3 -ffast-math
    - Cray: default
  - Libraries: LibSci/ACML, FFTW/FFTW3
- Tested with:
  - Job Types: Davidson, CG
  - Concurrencies: 24, 48, 96, 144

Performance of compilers are very similar when FFTW3 is used for QE





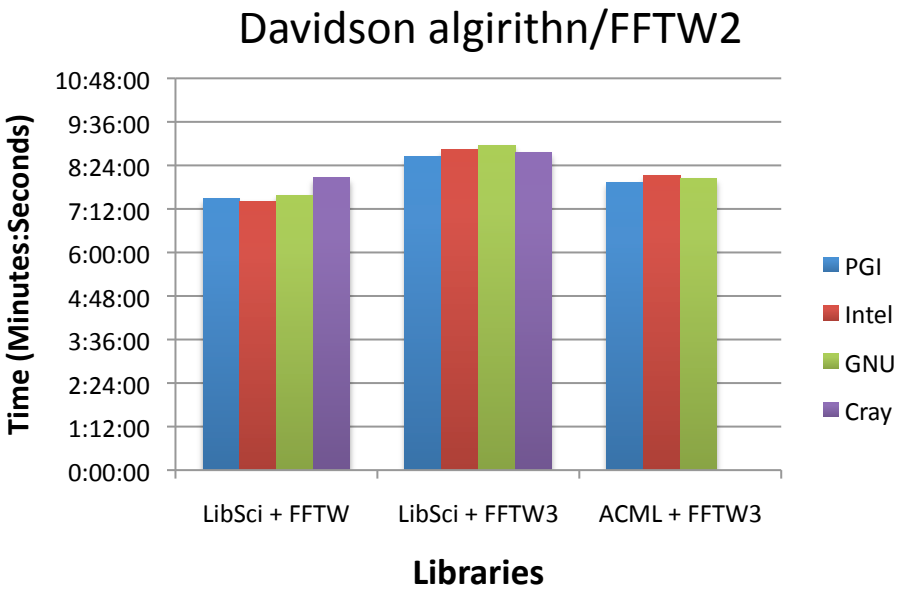
Test case:

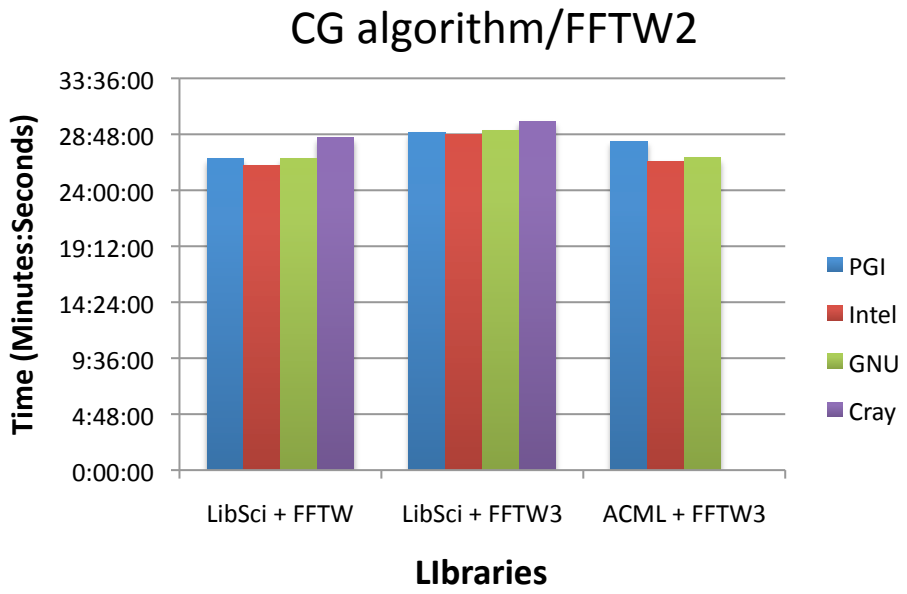
- a (8,0) single walled-carbon nanotube, an 80 Ry wavefunction cutoff in an 11041 au^3 unit cell.
- A self-consistent field calculation

Compiler	Faster than the default compiler by (%)
PGI	default
Intel	0.5%
GNU	-0.2%
Cray	0.3%

QE runs at a similar speed across different compilers + FFTW3.

The optimal combination for QE is Intel compiler with LibSci and internal FFTW2



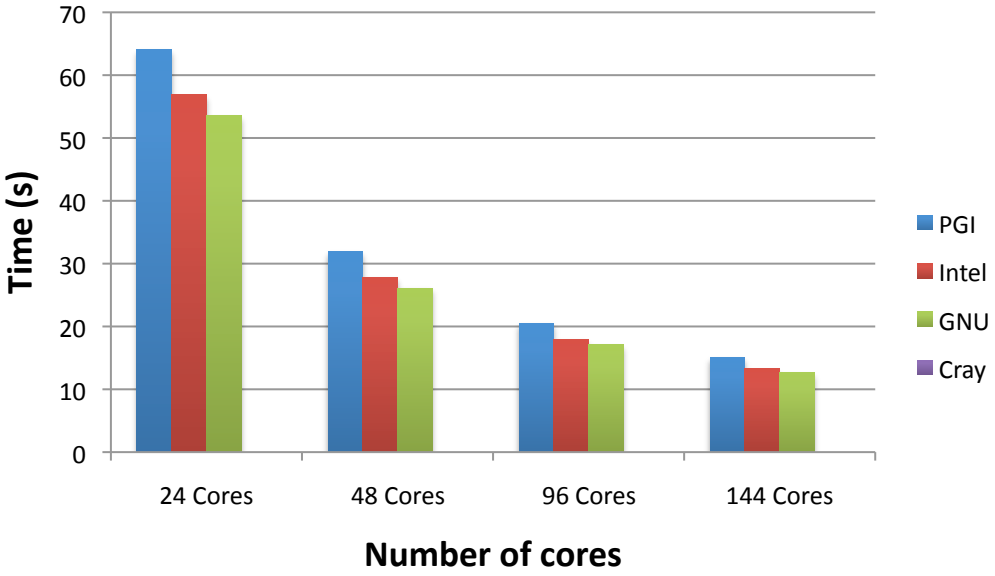


QE runs faster by up to 13.6% if using the internal FFTW2 library and Intel compiler than the current build (PGI+ FFTW3).

NAMD (2.8)

- Program Description
  - NAMD is a C++ molecular dynamics simulator.
- Options explored
  - Compilers and optimization Flags used
    - PGI: -fastsse -O3
    - Intel: -O2, -ip
    - GNU: -O3, -ffast-math, -fexpensive-optimization
  - Libraries used: FFTW2, TCL; Charm++
- Tested with a standard benchmark
  - Job Type: Particle Mesh Ewald (PME)
  - Concurrencies: 24, 48, 96, 144

GNU compiler performs best for NAMD



Compiler	Faster than GNU compiler by (%)
PGI	-22%
Intel	-5%
GNU	default
Cray	Failed to build

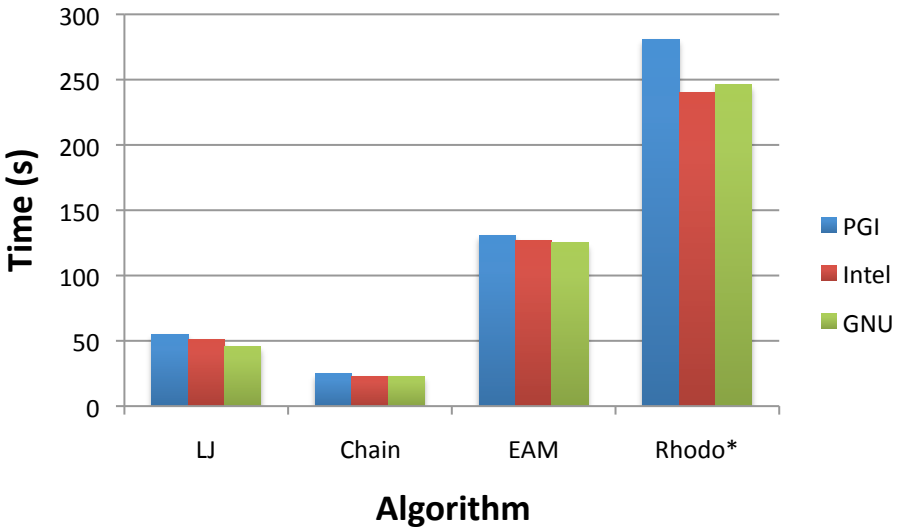
Test case:
A standard benchmark, APoa1, 92,424 atoms (PME)

The GNU compiler was confirmed as the best compiler for C++ code.

LAMMPS (Mar 15, 2012)

- Program Description
  - LAMMPS is a C++ classical molecular dynamics code.
- Options explored
  - Compilers and optimization flags used
    - PGI: -fastsse
    - Intel: -O3 -ip -unroll0
    - GNU: -O3 -ffast-math -fexpensive-optimization
    - Cray: -O ipa0
  - Libraries used: FFTW2
- Tested with
  - Four standard benchmark: LJ, Charm, EAM, Rhodo
  - 4 cores used for each test

GNU and Intel Compilers Have Similar Performance with LAMMPS



* To fit Rhodo runtime in the figure, time shown for Rhodo tests were the real runtime – 600 seconds.

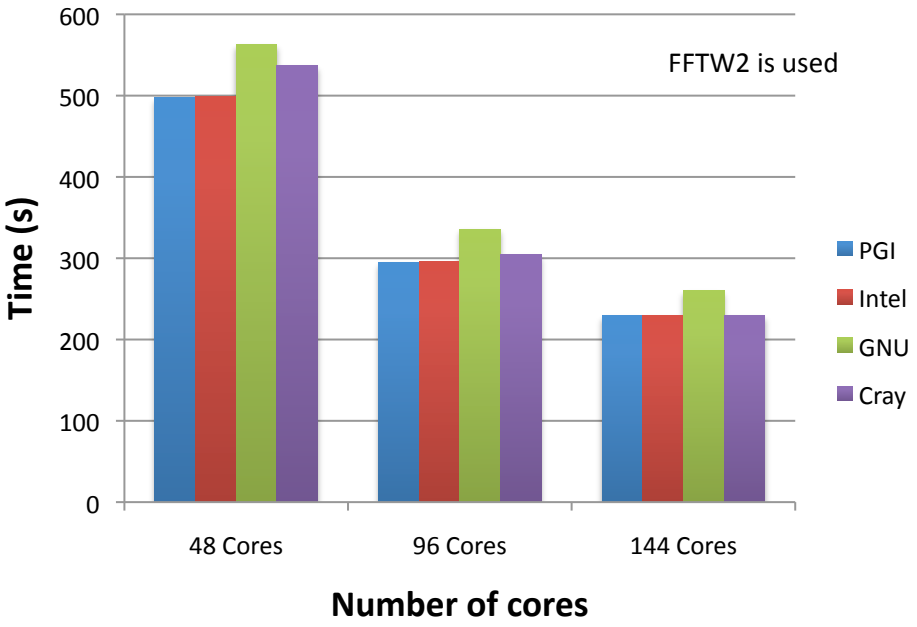
Compiler	Faster than GNU compiler by (%)
PGI	-5% ~ -20%
Intel	-10% ~ +2%
GNU	default
Cray	Failed to build

LAMMPS performs best with most of the test cases with GNU compiler.

BerkeleyGW (1.0.x)

- Program Description
  - BerkeleyGW is a Fortran, parallel computer package that calculates the quasiparticle properties and the optical response of materials.
- Options explored
  - Compilers and optimization flags used
    - PGI: -fast
    - Intel: -fast
    - GNU: -O3 -ffast-math -fexpensive-optimizations
    - Cray: default
  - Libraries: LibSci/ACML, FFTW2/FFTW3
- Tested with
  - Job Type: Epsilon
  - Concurrencies: 24, 48, 96, 144

Intel and PGI compilers have the best performance with BerkeleyGW

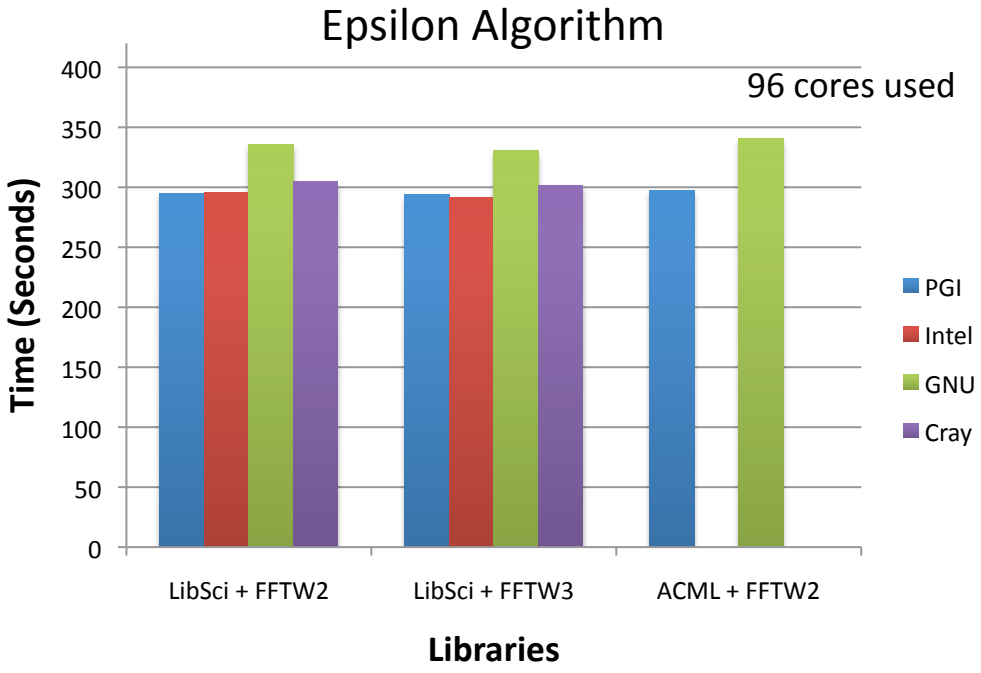


Compiler	Faster than the default compiler by (%)
PGI	default
Intel	0%
GNU	-13%
Cray	-8%

- Benchmark:
  - A (8,0) single walled-carbon nanotube, with a 80 Ry. wavefunction cutoff, 12 Ry. dielectric cutoff, and 240 empty states.

BerkeleyGW runs slowest with GNU compiler

The LibSci + FFTW3 library combination showed the best performance



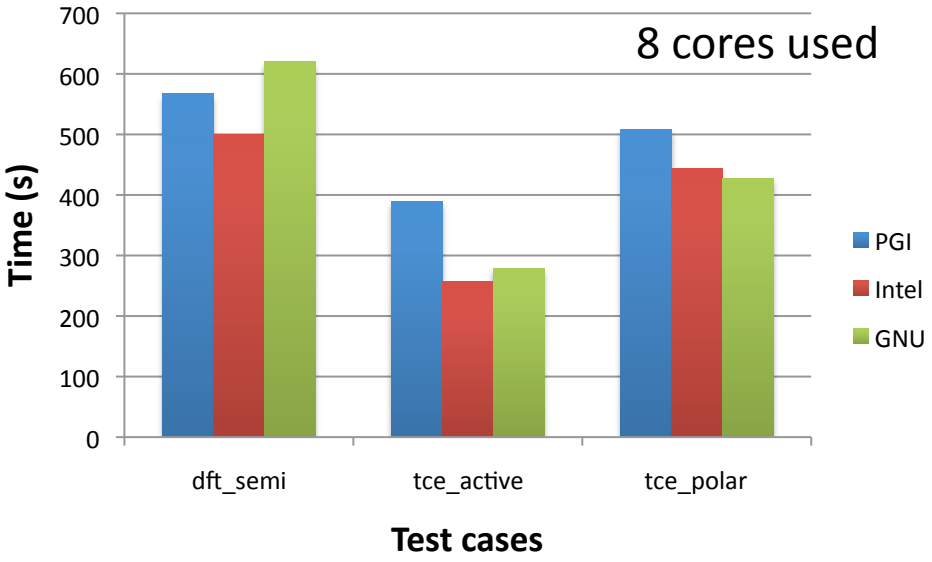
Compiler with LibSci+FFTW3	Performance gain of (Compiler + LibSci+FFTW2) relative to (the same Compiler + LibSci)
PGI	0%
Intel	1.2%
GNU	-12.5%
Cray	-3%

The Intel compiler outperformed the PGI compiler with LibSci + FFTW3 configuration by 1.2%.

NWChem (6.1)

- Program Description
  - NWChem is a computational chemistry software designed for parallel, high performance compute systems.
- Variables Tested
  - Compiler Optimization Flags
    - PGI: -fastsse, -O3, -Kieee, -Mipa
    - Intel: -O3, -prefetch, -unroll
    - GNU: -O3, -ffast-math, -mfpmath=sse
  - Libraries used: Libsci, GA
- Tested with 3 standard benchmark cases from NWChem distribution
  - Benchmark: tce_polar_ccsd_big.nw, tce_active_ccsdt.nw, dft_semidirect.nw
  - Concurrencies: 4, 8, 24 cores.

Intel compiler is arguably the optimal compiler for NWChem



Compiler	Faster than PGI compiler by (%)
PGI	default
Intel	12% ~ 34%
GNU	-9% ~ 28%
Cray	Built, but failed to run

- The Intel compiler showed the best performance on two benchmark cases, and the GNU compiler showed the best performance on one benchmark case.
- **The highest performance increase, 34%, was seen with the Intel compiler compared to PGI.**

Results Summary

Program	Default Compiler	Best Compiler	Default Library	Best Library	Performance Increase
VASP	PGI	Cray	LibSci	LibSci	11.2%
QE	PGI	Intel	LibSci + FFTW3	LibSci + FFTW	13.6%
NAMD	GNU	GNU	FFTW	-	0.0%
LAMMPS	GNU	GNU/Intel	FFTW	-	0.0%
BerkeleyGW	PGI	PGI/Intel	FFTW2	FFTW3	1.2%
NWChem	PGI	Intel	BLAS/ScaLAPACK	-	34.2%

Compiler versions tested	Library versions tested
PGI 11.9.0	libsci 11.0.03
GNU 4.6.2	acml 4.4.0
Intel 12.1.2.273	FFTW2 2.1.5.3; 1.2 (internal)
Cray cce/8.0.1	FFTW3 3.2.2.1

Answer to the charging question

- Different applications have different optimal compilers
  - No performance gain was found for the two C++ codes tested, NAMD and LAMMPS, we have confirmed that the GNU compiler is the optimal compiler for them.
  - Intel compiler is arguably the optimal compiler for NWChem and brings 12%~34% of performance increase compared to the current PGI build.
  - Combination of Intel compiler and Internal FFTW brings QE up to 13.5% of performance gain compared to the current build
  - Cray compiler is the optimal compiler for VASP which allows VASP to run faster by 5%~11% reliably with various tests.
  - BerkeleyGW performs slightly faster (~1%) with Intel compiler than with PGI compiler.
- Need further tests in all use cases (algorithms, job types, concurrencies) to confirm and conclude the optimal compiler for each application.

Acknowledgment

- The Center for Science and Engineering Education at Lawrence Berkeley National Laboratory and the Student Undergraduate Laboratory Internship of the U.S Department of Energy.
- Katie Antypas who mentored Megan during her internship
- David Turner, Mike Stewart and other members of User Services Group who helped Megan to get started at NERSC

About the authors

Megan Bowling

Megan Bowling was supported by the Center for Science and Engineering Education at Lawrence Berkeley National Laboratory and the Student Undergraduate Laboratory Internship of the U.S Department of Energy.

Jack Deslippe

Jack Deslippe is a HPC consultant at NERSC. He is the main developer of the BerkeleyGW code.

Zhengji Zhao

Zhengji Zhao is a HPC consultant at NERSC