Making Effective Use of Compilers at NERSC

Michael Stewart
NERSC User Services Group
August 15, 2012
Introduction

- Description of the Hopper compiling environment.
- Strengths and weaknesses of each compiler.
- Advice on choosing the most appropriate compiler for your work.
- Comparative results on benchmarks and other codes.
- How to use the compilers effectively.
- Carver compiling environment.
- Plans for the new Cray Cascade system (NERSC 7) compiling environment.
- Your feedback.
Why So Many Compilers on Hopper?

- NERSC5 (Cray Franklin XT) was delivered in 2006 with the only commercially available compiler, PGI.
- GNU compilers were on Franklin, but at that time GNU Fortran optimization was poor.
- Next came Pathscale because of superior optimization for Franklin's AMD Opteron processors.
- Cray ported their well optimized compiler to the Opteron so it was added next.
- Intel was popular on Carver, and it produced highly optimized codes on Hopper.
- PGI is still the default, but this is not a NERSC recommendation. Cray's current out of the box compiler is the Cray compiler, but we kept PGI as the default to avoid disruption.
How to Change Compilers on Hopper

- Use the Cray wrappers ftn, cc, and CC to invoke the compiler to get the proper libraries and not the compiler specific invocation, e.g. gcc, pgf90, ifort.
- By default PGI will be used with the wrappers.
- To use other compilers simply swap in the appropriate PrgEnv module:
  - module swap PrgEnv-pgi PrgEnv-cray
  - module swap PrgEnv-pgi PrgEnv-intel
  - module swap PrgEnv-pgi PrgEnv-gnu
- Nothing else needs to be done to use the new compiler to build codes.
PGI

● Strengths
  ○ Available on a wide variety of platforms making codes very portable.
  ○ Because of its wide usage, it is likely to compile almost any valid code cleanly.
  ○ Very well supported. Bugs are fixed relatively quickly and there are frequent bugfix releases.

● Weaknesses
  ○ Does not optimize as well as compilers more narrowly targeted to AMD architectures.

● Optimization recommendation:
  ○ -fast
  ○ Default optimization level: -O1
Cray

● Strengths
  ○ Fortran is well optimized for the Hopper architecture.
  ○ Uses Cray math libraries for optimization.
  ○ Well supported.
  ○ Very good at standards compliance and adoption.

● Weaknesses
  ○ Compilations can take much longer than with other compilers and create much larger executables.
  ○ Not very comfortable with C++ codes.
  ○ Very picky about standard compliance.

● Optimization recommendations:
  ○ Compile with no explicit optimization arguments. The default level of optimization is very high.
Intel

● Strengths
  ○ Optimizes C and Fortran codes very well.
  ○ Supports C++ very well.

● Weaknesses
  ○ Occasional problems in porting codes to this compiler.
  ○ -fast optimization level can be problematic.
    ■ Can take a very long time or fail.
    ■ Occasionally has produced incorrect results.

● Optimization recommendations:
  ○ Compile with no explicit optimization arguments. The default level of optimization is very high.
GNU/GCC

● Strengths
  ○ Available on a wide variety of platforms for free.
  ○ Exposure to a wide variety of codes, so any given code is likely to compile cleanly.
  ○ Very good at C++ optimization.
  ○ Optimizes Fortran codes as well as PGI on the average.

● Weaknesses
  ○ Not a commercial product, so no guarantee of bug fixes.
  ○ Does not optimize as well as architecture targeted compilers like Intel and Cray.

● Optimization recommendation:
  ○ -O3 -ffast-math
Pathscale

- This compiler is no longer supported by Cray.
- Contact consultants for assistance in converting to a different compiler (consult@nersc.gov).

Strengths
- Good optimization, generally not as good as Intel or Cray.

Weaknesses
- Support level and future of the product are questionable.
- Cray is withdrawing library support for this compiler.

Optimization recommendation:
- -O3
- Default optimization is -O2.
## Fortran Standards Compliance

- **Fortran 2003 - Object Oriented extensions.**
  - Cray and Intel almost fully compliant.
  - PGI and GNU mostly compliant.

- **Fortran 2008 - Coarray support.**
  - No one close to being fully compliant.
  - Cray, Intel, GNU have many features, PGI comparatively few.
Which Compiler to Use?

- Porting a code to Hopper.
  - Use the existing compiler if it is on Hopper, since relatively minor changes should be necessary to the Makefile or configure script.

- Developing a code on Hopper.
  - For C++ use Intel or GNU.
  - Targeted for Cray systems? The Cray Fortran and Intel compilers are likely to produce the fastest code.
  - Will it be ported to other systems? GNU will produce relatively fast code and can be ported more easily to other architectures.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Science Area</th>
<th>Algorithm</th>
<th>Concurrency (Scaling)</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTC</td>
<td>Fusion</td>
<td>PIC, Finite Difference</td>
<td>2048 (weak)</td>
<td>f90</td>
</tr>
<tr>
<td>IMPACT-T</td>
<td>Accelerator Physics</td>
<td>PIC, FFT</td>
<td>1024 (strong)</td>
<td>f90</td>
</tr>
<tr>
<td>MILC</td>
<td>Lattice Gauge Physics</td>
<td>Conjugate Gradient, FFT, Sparse Matrix</td>
<td>1024 (weak)</td>
<td>c, assembly</td>
</tr>
<tr>
<td>PARATEC</td>
<td>Material Science</td>
<td>DFT, FFT, BLAS</td>
<td>1024 (strong)</td>
<td>f90</td>
</tr>
<tr>
<td>Benchmark</td>
<td>Full Name</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>Block Tridiagonal</td>
<td>Solve a synthetic system of nonlinear PDEs using a block tridiagonal algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CG</td>
<td>Conjugate Gradient</td>
<td>Estimate the smallest eigenvalue of a large sparse symmetric positive-definite matrix using the inverse iteration with the conjugate gradient method as a subroutine for solving systems of linear equations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP</td>
<td>Embarrassingly Parallel</td>
<td>Generate independent Gaussian random variates using the Marsaglia polar method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>Fast Fourier Transform</td>
<td>Solve a three-dimensional PDE using FFT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU</td>
<td>Lower-Upper Symmetric Gauss-Seidel</td>
<td>Solve a synthetic system of nonlinear PDEs using a symmetric successive over-relaxation algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MG</td>
<td>MultiGrid</td>
<td>Approximate the solution to a three-dimensional discrete Poisson equation using the V-cycle multigrid method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>Scalar Pentadiagonal</td>
<td>Solve a synthetic system of nonlinear PDEs using a scalar pentadiagonal algorithm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hopper Benchmark Performance
Normalized to PGI Performance

Wall clock time ratio, lower is better.
Compiling USG Supported Applications

- **VASP** - performs *ab initio* quantum-mechanical molecular dynamics (MD) using pseudopotentials and a plane wave basis set. (f90)
- **QE (QuantumEspresso)** - an integrated suite of computer codes for electronic structure calculations and materials modeling at the nanoscale. (f90)
- **NAMD** - a molecular dynamics (MD) program designed for parallel computation. (C/C++)
- **LAMMPS** - a large scale classical molecular dynamics code. (C++)
- **BerkeleyGW** - calculates the quasiparticle properties and the optical responses of a large variety of materials. (f90)
- **NWChem** - a computational chemistry package. (f90)
## Building and Running NERSC Applications

**Percent Performance Decrease/Improvement over PGI**

<table>
<thead>
<tr>
<th>Program</th>
<th>Intel</th>
<th>GNU</th>
<th>Cray</th>
<th>Best Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>VASP</td>
<td>12% to 5%</td>
<td>6% to 4%</td>
<td>0% to 11%</td>
<td>Cray</td>
</tr>
<tr>
<td>QE</td>
<td>2%</td>
<td>1%</td>
<td>7%</td>
<td>Intel</td>
</tr>
<tr>
<td>NAMD</td>
<td>14%</td>
<td>18%</td>
<td>Failed</td>
<td>GNU</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>5% to 17%</td>
<td>5% to 9%</td>
<td>6% to 4%</td>
<td>Intel</td>
</tr>
<tr>
<td>BerkeleyGW</td>
<td>0%</td>
<td>13%</td>
<td>8%</td>
<td>PGI/Intel</td>
</tr>
<tr>
<td>NWChem</td>
<td>12% to 34%</td>
<td>9% to 28%</td>
<td>Failed</td>
<td>Intel</td>
</tr>
</tbody>
</table>

Provided by Zhengji Zhao, Megan Bowling and Jack Deslippe from CUG 2012
Set of benchmarks to test how well a compiler optimizes C++ operations and language features. Not a test of floating point performance.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepanov_abstraction</td>
<td>Sorting and summing values wrapped in curly braces.</td>
</tr>
<tr>
<td>stepanov_vector</td>
<td>Replacing pointers with vector iterators and using reverse iterators.</td>
</tr>
<tr>
<td>functionobjects</td>
<td>Instantiation of simple functors and the relative performance of function pointers, functors and inline operators.</td>
</tr>
<tr>
<td>simple_types_constant_folding</td>
<td>Folding constant math expressions on simple data types.</td>
</tr>
<tr>
<td>simple_types_loop_invariant</td>
<td>Moving loop invariant calculations out of the loop.</td>
</tr>
<tr>
<td>loop_unroll</td>
<td>Unrolling loops to hide instruction latency.</td>
</tr>
</tbody>
</table>
## C++ Benchmark Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>PGI</th>
<th>Cray</th>
<th>Intel</th>
<th>GNU</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepanov_abstraction</td>
<td>300.88</td>
<td>169.78</td>
<td>39.37</td>
<td>50.11</td>
</tr>
<tr>
<td>stepanov_vector</td>
<td>Did not compile.</td>
<td>233.44</td>
<td>67.99</td>
<td>84.97</td>
</tr>
<tr>
<td>functionobjects</td>
<td>36.93</td>
<td>38.44</td>
<td>31.15</td>
<td>31.11</td>
</tr>
<tr>
<td>simple_types_constant_folding</td>
<td>1413.96</td>
<td>7856.88</td>
<td>1571.74</td>
<td>509.68</td>
</tr>
<tr>
<td>simple_types_loop_invariant</td>
<td>1041.58</td>
<td>2366.86</td>
<td>863.62</td>
<td>889.20</td>
</tr>
<tr>
<td>loop_unroll</td>
<td>5014.56</td>
<td>1323.57</td>
<td>363.53</td>
<td>866.86</td>
</tr>
</tbody>
</table>

Times are in seconds, lower is better.
Compiling for OpenMP on Hopper

- Cray compiler: -Oomp (on by default)
- PGI: -mp=nonuma
- Intel: -openmp
- GNU: -fopenmp
- Pathscale: -mp
Running with OpenMP on Hopper

● Run time all compilers:
  ○ - set OMP_NUM_THREADS to number of threads
  ○ aprun -d numthreads ...
● Pathscale run time - set PSC_OMP_AFFINITY to FALSE.
● Intel run time - use "-cc none" or "-cc numa_node" arguments to aprun.
OpenMP/Hybrid Run Time Optimization

- Each 24 core Hopper compute node consists of 4 6 core "numa nodes".
- Best hybrid code performance when you allocate 1 MPI process with 6 threads to each of these numa nodes and use their local memory.
- Single node parameters:
  - export OMP_NUM_THREADS=6
  - aprun -d 6 -N 4 -S 1 -ss ......
- For more details see https://www.nersc.gov/users/computational-systems/hopper/performance-and-optimization/using-openmp-effectively-on-hopper/
OpenACC Support on Hopper

- OpenACC - A standard is designed to simplify parallel programming of heterogeneous CPU/GPU systems.
- Accomplished with #pragma's and compiler directives in the source code like OpenMP.
- Currently supported by the Cray compiler.
- Coming in version 12.6 of the pgi compiler.
PGAS Support on Hopper

- PGAS (Partitioned Global Address Space) - allows the programmer to view a single shared partitioned address space where each variable is associated with a single processor, but can be directly read and written by any processor.
  - UPC - Unified Parallel C.
  - CAF - Coarray Fortran.
- Available with Cray, Berkeley UPC, and Intel Compilers.
- See https://wwwnerscgov/users/training/online-tutorials/introduction-to-pgas-languages/#toc-anchor-2
Carver Compiler Environment

- Default compiler is PGI for Franklin/Hopper consistency, not as a NERSC recommendation.
- Intel compiler is available as a module, and generally produces much faster code than PGI.
- GNU compiler has comparable performance to PGI.
- Optimization recommendations are the same as on Hopper:
  - Intel: default, no optimization arguments.
  - PGI: -fast
  - GNU: -O3 -ffast-math
NERSC7 Compiler Environment

- NERSC7 will be a Cray Cascade Intel Xeon based system.
- Default compiler will be Intel.
- The Cray and GNU compilers will be available as modules.
- There are no plans to have PGI or Pathscale on the system.
- NERSC will do extensive performance analysis on benchmarks before the system becomes available to users.
Questions and Comments