Compiler Optimization
STAT 624: Big Data and Databases
Professor: Henrik Schmiediche
Student: Carson Hanel

This is a quick and easy runthrough of being able to assess compilation options on the Terra cluster computing system utilized by Texas A&M University for high performance research computing. I'd like to go through this analysis with emphasis on the commands utilized, and the performance boosts (or lack thereof) gained. Let's start with the most basal Fortran compiler, namely GCC.

Basic Compiler Flags

To load the compiler modules, use: module load intel/2018b

When running the compiled code, the syntax is as follows: time ./brem < thik.inp &> dev/null/

- **Command:** gfortran -O0 -o brem brem.f gammln.f
  - 2 minutes and 35.743 seconds
- **Command:** gfortran -O1 -o brem brem.f gammln.f
  - 2 minutes and 13.546 seconds
- **Command:** gfortran -O2 -o brem brem.f gammln.f
  - 2 minutes and 6.774 seconds
- **Command:** gfortran -O3 -o brem brem.f gammln.f
  - 2 minutes and 5.009 seconds
- **Command:** gfortran -Ofast -o brem brem.f gammln.f
  - 2 minutes and 6.407 seconds
- **Assessment:** It would seem that Fortran's compiler, though equipped with optimization technology, gives brem.f minimal speedup with the various flags. GCC also has a mirror to Intel's -fast (namely -Ofast), which is still vastly inferior to Intel compilation methods.

Now, let's check out speeds attained from the same file under Intel's compiler

- **Command:** ifort -o brem brem.f gammln.f
  - 48.054 seconds
- **Command:** ifort -O0 -o brem brem.f gammln.f
  - 1 minute and 27.724 seconds
- **Command:** ifort -O1 -o brem brem.f gammln.f
  - 49.968 seconds
- **Command:** ifort -O2 -o brem brem.f gammln.f
  - 48.007 seconds
- **Command:** ifort -O3 -o brem brem.f gammln.f
  - 47.340 seconds

- **Command:** ifort -fast -o brem brem.f gammln.f
  - 16.181 seconds
- **Assessment:** From the looks of it, Intel's compiler by default utilizes the -O2 compilation flag, which allows intel to really optimize to the teeth, but avoids using the -O3 flag, which aggressively changes code behaviour. Compiling with the -O0 flag is akin to compiling in Debug mode in Visual Studio, whereas compiling with the default -O2 flag is more akin to Release mode.

Intermediate Compiler Flags

Now that it's been shown how much more powerful the Intel compiler is for this application, let's take a look at the various intel optimization flags. As a reminder, the base Intel speed was 48.054 seconds. Let's see how the sub-flags of -fast improve individually on that baseline.

- **Command:** ifort -xHost -o brem brem.f gammln.f
  - **Function:** Targets the current CPU architecture's AVX instruction set for optimization.
  - **Speed:** 47.671 seconds
  - **NOTE:** If you'd like to target a specific AVX architecture, rather than the current Host's (to distribute code), utilize **-axCORE-AVX2**
- **Command:** ifort -ipo -o brem brem.f gammln.f
  - **Function:** Targets interprocess optimization, allowing the compiler to better schedule the program's multithreading.
  - **Speed:** 16.727 seconds
- **Command:** ifort -no-prec-div -o brem brem.f gammln.f
  - **Function:** Reduces the precision on division, minimizing required computation for floating point operations at the expense of, of course, precision.
  - **Speed:** 48.044 seconds
- **Assessment:** It seems as though the reduction in precision and targeting a specific AVX instruction set didn't have a large impact, however, -fast's -ipo, or rather interprocess optimization, allows for much better scheduling of jobs - eliminating busy spinning, and optimizing the way multithreaded code runs on the CPU. I'd assume that brem.f is a resource hungry multithreaded program with some easily fixable bugs in it. So, to put it plainly, -ipo is the heavyweight compiler flag for the file brem.f.

Advanced Compiler Flags

Finally, let's check out profile guided optimization with the default Intel compiler, then see how it performs on the compiler's -O2 flag.

- **Command Sequence: -O2**
  - **Original speed:** 48.007 seconds
  - Ifort -prof-gen -02 -o brem brem.f gammln.f

- ./brem &> /dev/null
- Ifort -prof-use -02 -o brem brem.f gammln.f
- **First run of -prof-use :** 54.165 seconds
- **Second run of -prof-use:** 50.008 seconds
- **Third run of -prod-use :** 49.800 seconds
- **Assessment:** This program did not benefit greatly beyond the baseline from Intel's performance profiler. The profiler is infinitely useful on code that branches often, but, it would seem that Intel's O2 flag already, potentially, took care of that optimization. As you can see, over more runs of the profile, the speed converged to roughly what it was without the profiler. As a further note, -O2 is akin to the default Intel compilation flag.

- **Command Sequence: -fast**
  - **Original speed:** 16.181 seconds
  - Ifort -prof-gen -fast -o brem brem.f gammln.f
  - ./brem &> /dev/null
  - Ifort -prof-use -fast -o brem brem.f gammln.f
  - **First run of -prof-use :** 15.492 seconds
  - **Second run of -prof-use:** 15.266 seconds
  - **Third run of -prod-use :** 15.254 seconds
  - **Assessment:** So, oddly, Intel's -fast flag still benefits, in a noticeable way, the speed even though it's already been optimized so heavily. At this point, it's almost akin to a very weak AI slowly realizing the most optimal way to run the program. After 5 runs of -prod-use, though minimal, brem continued to see speed gains until finally regressing on the 6th run.