Ranger User Guide

Optimization Section

General Optimization Outline

The most practical approach to enhance the performance of applications is to use use advanced compiler options, employ high performance libraries for common mathematical algorithms and scientific methods, and tune the code to take advantage of the architecture. Compiler options and libraries can provide a large benefit for a minimal amount of work. Always profile the entire application to ensure that the optimization efforts are focused on areas with the greatest return on the optimization efforts.

"Hot spots" and performance bottlenecks can be discovered with basic profiling tools like *prof* and *gprof*. It is important to watch the relative changes in performance among the routines when experimenting with compiler options. Sometimes it might be advantageous to break out routines and compile them separately with different options that have been used for the rest of the package. Also, review routines for "hand-coded" math algorithms that can be replaced by standard (optimized) library routines. You should also be familiar with general code tuning methods and restructure statements and code blocks so that the compiler can take advantage of the microarchitecture.

Code should:

- be clear and comprehensible
- provide flexible compiler support
- should be portable

It is important to avoid too many architecture-specific and compiler-specific code constructs. Use language features and restructure code so that the compiler can discover how to optimize code for the architecture. That is, expose optimization when possible for the compiler, but don't rewrite the code specifically for the architecture.

Some best practices are:

- Avoid excessive program modularization (i.e. too many functions/subroutines)
  - write routines that can be inlined
  - use macros and parameters whenever possible
- Minimize the use of pointers
- Avoid casts or type conversions, implicit or explicit
- Avoid branches, function calls, and I/O inside loops
  - structure loops to eliminate conditionals
  - move loops around subroutine into the subroutine

Compiler Options

Compiler options must be used to achieve optimal performance of any application. Generally, the highest impact can be achieved by selecting an appropriate optimization level, by targeting the architecture of the computer (CPU, cache, memory system), and by allowing for interprocedural analysis (inlining, etc.). There is no set of options that gives the highest speed-up for all applications and consequently different combinations have to be explored.

PGI compiler :

-O[n]	Optimization level, n=0, 1, 2 or 3
-tp barcelona-64	Targeting the architecture
-Mipa[=option]	Interprocedural analysis, option=fast,inline

Intel compiler :

-O[n]	Optimization level, n=0, 1, 2 or 3
-x[p]	Targeting the architecture, p=W or O
-ip, -ipo	Interprocedural analysis

See the Development section for the use of these options.

Performance Libraries

TACC provides several ISP (Independent Software Providers) and HPC vendor math libraries that can be used in many applications. These libraries provide highly optimized math packages and functions for the Ranger system.

The ACML library (AMD Core Math Library) contains many common math functions and routines (linear algebra, transformations, transcendental, sorting, etc.) specifically optimized for the AMD Barcelona processor. The ACML library also supports multi-processor (threading) FFT and BLAS routines. The Intel MKL (Math Kernel Library) has a similar set of math packages. Also, the GotoBLAS libraries contain the fastest set of BLAS routine for this machine.

The default compiler representation for Ranger consists of 32-bit ints, and 64-bit longs and pointers. Likewise for Fortran, integers are 32-bit, and the pointers are 64-bit. This is called the **LP64** mode (Long and Pointers are 64-bit, ints and integers are 32-bit). Libraries with 64-bit integers are often suffixed with an **IPL64**.

ACML and MKL libraries

The ***"AMD Core Math Library"*** and ***"Math Kernel Library"*** consists of functions with Fortran, C, and C++ interfaces for the following computational areas:

- BLAS (vector-vector, matrix-vector, matrix-matrix operations) and extended BLAS for sparse computations
- LAPACK for linear algebraic equation solvers and eigensystem analysis
- Fast Fourier Transforms
- Transcendental Functions

Note, Intel provides performance libraries for most of the common math functions and routines (linear algebra, transformations, transcendental, sorting, etc.) for their em64t and Core-2 systems. These routines also work well on the AMD Opteron microarchitecture. In addition, MKL also offers a set of functions collectively known as VML -- the **"Vector Math Library"**. VML is a set of vectorized transcendental functions which offer both high performance and excellent accuracy compared to the *libm*, for vectors longer than a few elements.

GotoBLAS library

The ***"GotoBLAS Library"*** (pronounced "goat-toe") contains highly optimized BLAS routines for the Barcelona microarchitecture. The library has been compiled for use with the PGI and Intel Fortran compilers on Ranger. The Goto BLAS routines are also supported on other architectures, and the source code is available free for academic use (download). We recommend the GotoBLAS libraries for performing the following linear algebra operations and solving matrix equations:

- BLAS (vector-vector, matrix-vector, matrix-matrix operations)
- LAPACK for linear algebraic equation solvers and eigensystem analysis

Code Tuning

Additional performance can be obtained with these techniques :

- Memory Subsystem Tuning : Optimize access to the memory by minimizing the stride length and/or employ cache

blocking techniques.
- Floating-Point Tuning : Unroll inner loops to hide FP latencies, and avoid costly operations like divisions and exponentiations.
- I/O Tuning : Use direct-access binary files to improve the I/O performance.

These techniques are explained in further detail, with examples in the Memory Subsystem Tuning guide (http://www.tacc.utexas.edu/~jwozniak/ranger/user_guide/tuningmethods.php)