Brief notes on setting up semi-high performance computing environments

October 15, 2012

We have two different computing environments for fitting demanding models to large space and/or time data sets.

1. A distributed system consists of multiple autonomous computers (nodes) that communicate through a co mputer network. A computer program that runs in a distributed system is called a distributed program. Message Passing Interface (MPI) is a specification for an Application Programming Interface (API) that allows many computers to communicate with one another (implementations in C, C++, and Fortran.)

We have two different computing environments for fitting demanding models to large space and/or time data sets.

1. A distributed system consists of multiple autonomous computers (nodes) that communicate through a co mputer network. A computer program that runs in a distributed system is called a distributed program. Message Passing Interface (MPI) is a specification for an Application Programming Interface (API) that allows many computers to communicate with one another (implementations in C, C++, and Fortran.)

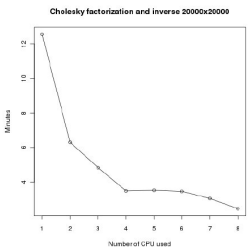
2. A shared memory multiprocessing system consists of a single computer with memory that may be simultaneously accessed by one or more programs running on multiple central processing units (CPUs). The OpenMP (Open Multi-Processing) is an API that supports shared memory multiprocessing programming (implementations in C, C++, and Fortran).

We have two different computing environments for fitting demanding models to large space and/or time data sets.

- Recent work focuses on fitting geostatistical (specifically point-referenced) models using MCMC methods. This necessitates iterative evaluation of a likelihood which requires operations on large matrices.
- A specific hurdle is factorization to computing determinant and inverse of large dense covariance matrices.
- We try to model our way out and use tools from computer science to overcome the computational complexity (e.g., covariance tapering, Kaufman et al. 2008; low-rank methods, Cressie and Johannesson 2008; Banerjee et al. 2008, etc.).
- Due to slow network communication and transport of submatrices among nodes distributed systems are not ideal for these types of iterative large matrix operations.

- My lab currently favors shared memory multiprocessing system.
- We buy rack mounted units (e.g., Sun Fire X4170 Server with 2 quad-core Intel Xeon Processor 5500 Series and 48 GB of RAM ∼10-15k) running the Linux operating systems.
- Software includes OpenMP coupled with Intel Math Kernel Library (MKL) `http://software.intel.com/en-us/non-commercial-software-development`. MKL is a library of highly optimized, extensively threaded math routines (e.g., BLAS, LAPACK, ScaLAPACK, Sparse Solvers, Fast Fourier Transforms, and vector RNGs).

So what kind of speed up to expect from threaded BLAS and LAPACK libraries. Mean computing times of dpotrf:



See `http://blue.for.msu.edu/comp-notes` for some simple examples of C++ with MKL and Rmath libraries along with associated Makefile files (I'll add more examples shortly and upon request).

- Many core and contributed packages (including *spBayes*) call Basic Linear Algebra Subprograms (BLAS) and LAPACK (Linear Algebra PACKage) Fortran libraries.
- Substantial computing gains:
  - processor specific threaded BLAS/LAPACK implementation (e.g., MKL or AMD's Core Math Library (ACML))
  - processor specific compilers (e.g., Intel's *icc/ifort*)

Compiling *R* to call MKL's BLAS and LAPACK libraries (rather than stock serial versions).

```
MKL_LIB_PATH="/opt/intel/composer_xe_2011_sp1.10.319/mkl/lib/intel64"

export LD_LIBRARY_PATH=$MKL_LIB_PATH

MKL="-L${MKL_LIB_PATH} -lmkl_intel_lp64 -lmkl_intel_thread \
       -lmkl_core -liomp5 -lpthread -lm"

./configure --with-blas="$MKL" --with-lapack
```


Time needed to collect 100 MCMC samples using *spLM* and threaded vs. non-threaded BLAS/LAPACK on a Intel Core 2 Quad processor and Ubuntu 8.10 Linux OS. *R* compiled with GNU gcc and gfortran.