Some notes on efficient computing and high performance computing environments

Abhi Datta[1], Sudipto Banerjee[2] and Andrew O. Finley[3]

July 31, 2017

[1]Department of Biostatistics, Bloomberg School of Public Health, Johns Hopkins University, Baltimore, Maryland.

[2]Department of Biostatistics, Fielding School of Public Health, University of California, Los Angeles.

[3]Departments of Forestry and Geography, Michigan State University, East Lansing, Michigan.

Code implementation

Very useful libraries for efficient matrix computation:

1. Fortran BLAS (Basic Linear Algebra Subprograms, see Blackford et al. 2001). Started in late 70s at NASA JPL by Charles L. Lawson.

2. Fortran LAPACK (Linear Algebra Package, see Anderson et al. 1999). Started in mid 80s at Argonne and Oak Ridge National Laboratories.

Modern math software has a heavy reliance on these libraries, e.g., Matlab and *R*. Routines are also accessible via C, C++, Python, etc.

Many improvements on the standard BLAS and LAPACK functions, see, e.g.,

- Intel Math Kernel Library (MKL)
- AMD Core Math Library (ACML)
- Automatically Tuned Linear Algebra Software (ATLAS)
- Matrix Algebra on GPU and Multicore Architecture (MAGMA)
- OpenBLAS `http://www.openblas.net`
- vecLib (for Mac users only)

Key BLAS and LAPACK functions used in our setting.

Function	Description
`dpotrf`	LAPACK routine to compute the Cholesky factorization of a real symmetric positive definite matrix.
`dtrsv`	Level 2 BLAS routine to solve the systems of equations $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x}$ and $\mathbf{b}$ are vectors and $\mathbf{A}$ is a triangular matrix.
`dtrsm`	Level 3 BLAS routine to solve the matrix equations $\mathbf{AX} = \mathbf{B}$, where $\mathbf{X}$ and $\mathbf{B}$ are matrices and $\mathbf{A}$ is a triangular matrix.
`dgemv`	Level 2 BLAS matrix-vector multiplication.
`dgemm`	Level 3 BLAS matrix-matrix multiplication.

Computing environments

Consider different environments:

1. A distributed system consists of multiple autonomous computers (nodes) that communicate through a 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.

Consider different environments:

1. A distributed system consists of multiple autonomous computers (nodes) that communicate through a 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.

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). OpenMP (Open Multi-Processing) is an API that supports shared memory multiprocessing programming.

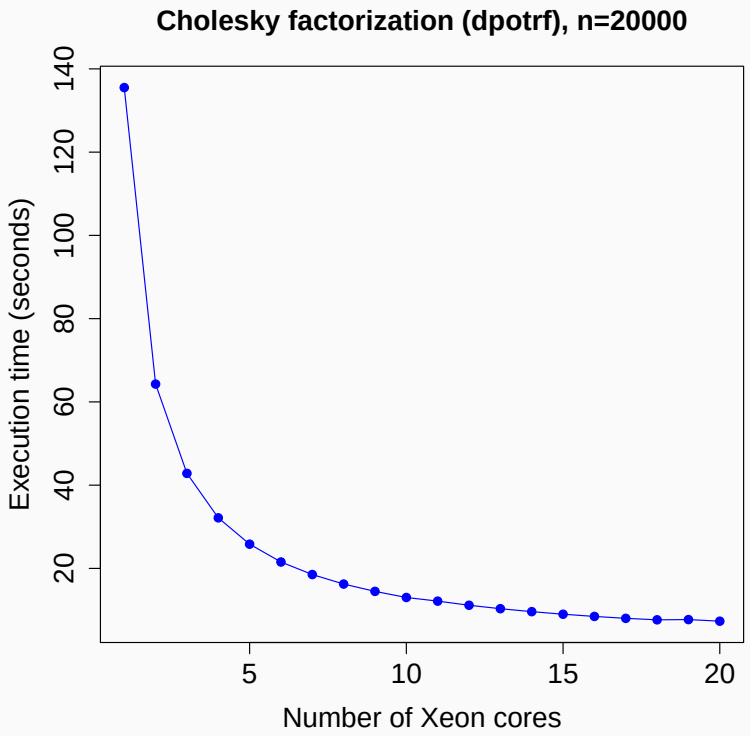
3. A heterogeneous system uses more than one kind of processor, e.g., CPU & (Graphics Processing Unit) GPU or CPU & Intel's Xeon Phi Many Integrated Core (MIC).

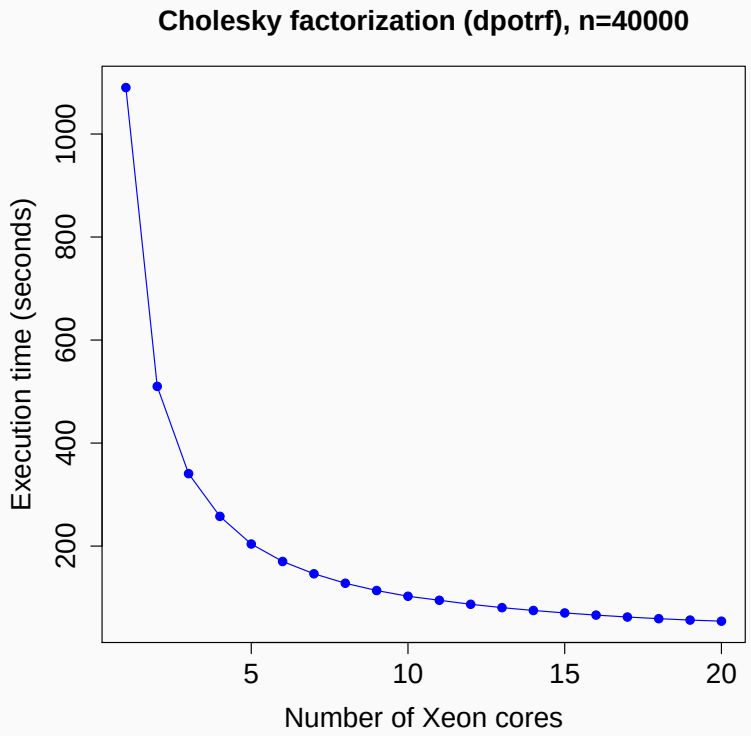
Which environments are right for large n settings?

- MCMC necessitates iterative evaluation of the 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 computing tools to overcome the 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 and heterogeneous systems.
- Newest unit is a Dell Poweredge with 384 GB of RAM, 2 threaded 10-core Xeon CPUs, and 2 Intel Xeon Phi Coprocessor with 61-cores (244 threads) running a Linux operating systems.
- Software includes OpenMP coupled with Intel MKL. MKL is a library of highly optimized, extensively threaded math routines designed for Xeon CPUs and Phi coprocessors (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.





R and threaded BLAS and LAPACK

- Many core and contributed packages (including *spBayes*) call BLAS)and LAPACK Fortran libraries.

- Compile *R* against threaded BLAS and LAPACK provides substantial computing gains:
  - processor specific threaded BLAS/LAPACK implementation (e.g., MKL or ACML)
  - processor specific compilers (e.g., Intel's *icc/ifort*)

For Linux/Unix: compiling *R* to call MKL's BLAS and LAPACK libraries (rather than stock serial versions). Change your `config.site` file and `configure` call in the R source code directory.

My `config.site` file

```
CC=icc
CFLAGS="-g -O3 -wd188 -ip -mp"
F77=ifort
FLAGS="-g -O3 -mp -openmp"
CXX=icpc
CXXFLAGS="-g -O3 -mp -openmp"
FC=ifort
FCFLAGS="-g -O3 -mp -openmp"
ICC_LIBS=/opt/intel/composerxe/lib/intel64
IFC_LIBS=/opt/intel/composerxe/lib/intel64
LDFLAGS="-L$ICC_LIBS -L$IFC_LIBS -L/usr/local/lib"
SHLIB_CXXLD=icpc
SHLIB_CXXLDFLAGS=-shared
```

Compiling *R* to call MKL's BLAS and LAPACK libraries (rather than stock serial versions). Change your `config.site` file and `configure` call in the R source code directory.

My `configure` script

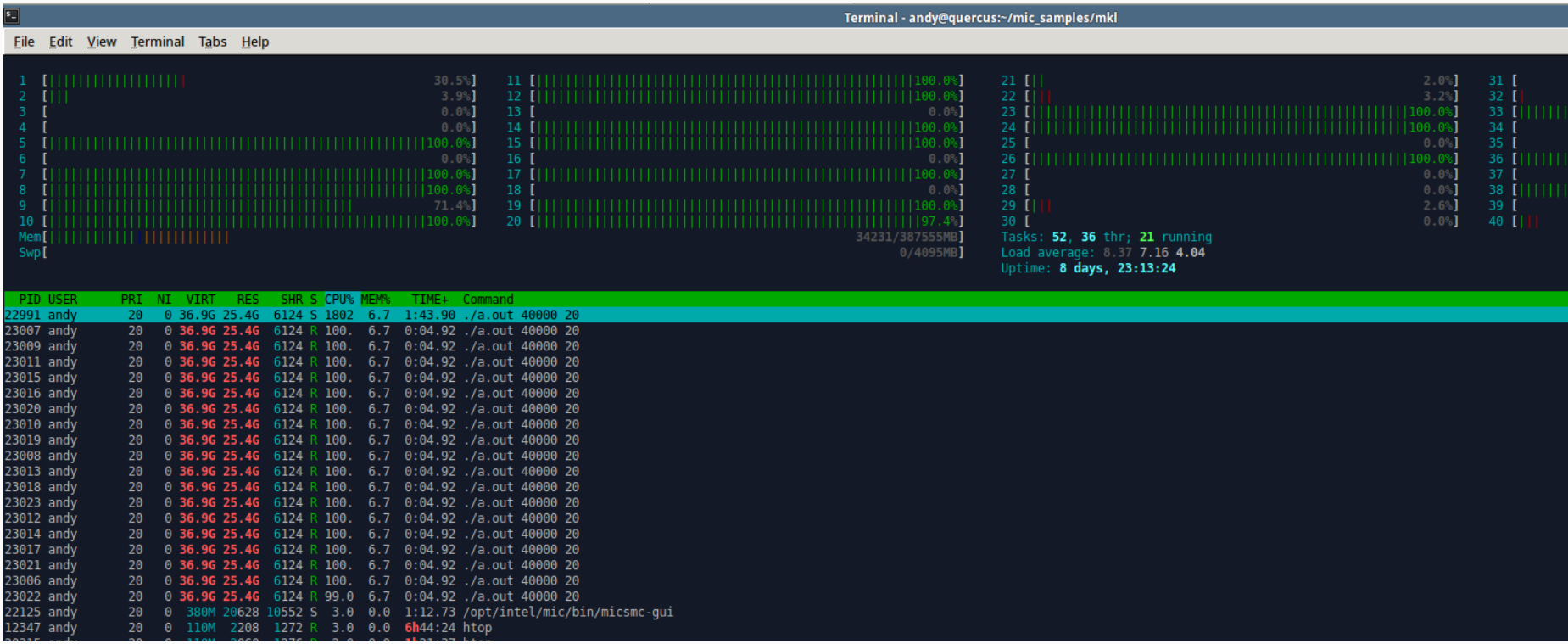
```
MKL_LIB_PATH="/opt/intel/composerxe/mkl/lib/intel64"

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
            --prefix=/usr/local/R-mkl
```

For many BLAS and LAPACK functions calls from *R*, expect near linear speed up ...



Mac users, see some vecLib linking hints at
`http://blue.for.msu.edu/GWEDA17/BLASHints.txt`

Linux/Unix users, compile R with MKL as described above, or compile OpenBLAS and just redirect R's default libRblas.so to libopenblas.so, e.g.,

```
/usr/local/lib/R/lib/libRblas.so ->
                    /opt/OpenBLAS/lib/libopenblas.so
```

See `http://blue.for.msu.edu/comp-notes` for some simple examples of C++ with MKL and Rmath libraries along with associated Makefile files.