Incorporating Parallel Libraries in Your Applications

Kent Milfeld

UP / TACC
May 28, 2009

Top 3 Approaches to Optimization

1. Compiler Options ⟸ **A Few Slides**
2. Libraries ⟸ **Our Focus**
3. Code Restructuring ⟸ **For Another Day**

Compiler Options

Three important Categories

- Optimization Level
- Architecture Specification
- Interprocedural Optimization

You should always have at least one option from each category!

Compiler Options

- Compilers can perform significant optimization
  - The compiler follows your lead!
  - Structure code to make apparent what the compiler should do (so that the compilers and others can understand it).
  - Use simple language constructs (e.g. don't use pointers, or OO code).
- Experiment with different options.
- May need routine-specific options.

Optimization Level: –O*n*

- -O0 no optimization, fast compilation
- -O1 optimize for speed, but disables optimizations which increase code size
- -O2 often default optimization
- -O3: aggressive prefetching, loop transformations, etc.

Architecture Specification

X87 instruction sets are now replaced by SSE “Vector” instruction sets.

(S)SSE = (Supplemental) Streaming SIMD Extension

SSE instructions sets are chip dependent

(SSE instructions pipeline and simultaneously execute independent operations to get multiple results per clock period.)

The –x<codes> { code = W, P, T, O, S}

directs the compiler to use most advanced SSE instruction set for the target hardware.

Architecture Specification

Intel

Processor-specific options (all do SSE and SSE2):

-xT includes SSE3 & SSSE3 instructions for EM64T (Lonestar, v. 10.1, SSSE only on Intel Chips)

-xW **no supplemental** Instructions (Ranger, v. 10.1)

-xO includes SSE3 Instructions (Ranger, v. 10.1)

PGI

-tp barcelona-64 uses barcelona instruction set (IS)

GCC

-mtune=barcelona -march=barcelona uses barcelona IS

Interprocedural Optimization (IP)

- Most compilers will handle IP within a single file (option –ip)
- The Intel -ipo compiler option does more
  - It adds additional information to each object file.
  - Then, during loading, the code is recompiled and IP among ALL objects is performed.
  - May take much more time: Code is recompiled during linking
  - It is **Important** to include options in link command (-ipo –O# -xW, etc.) (special Intel xild loader replaces ld)
  - When archiving in a library, you must use xiar, instead of ar.

TACC

Interprocedural Optimization (IP)

Intel

-ip	enable single-file interprocedural (IP) optimizations (within files). Line numbers produced for debugging
-ipo	enable multi-file IP optimizations (between files)

PGI

-Mipa=fast,inline	Interprocedural Optimization

Other Intel Compiler Options

Other options:

-g	debugging information, generates symbol table
-vec_report[#]	{#=0-5}, controls vector diagnostic reporting
-C	enable extensive runtime error checking (-CA, -CB, -CS, -CU, -CV)
-convert <kwd>	specify file format keyword: big_endian, cray, ibm, little_endian, native, vaxd
-openmp	enable the parallelizer to generate multi-threaded code based on the OpenMP directives.
-openmp_report	controls level of diagnostic reporting
-static	create a static executable for serial applications. MPI applications compiled on Lonestar cannot be built statically.

TACC

Other PGI Compiler Options

Processor-specific optimization options:

-fast	-O2 -Munroll=c:1 -Mnoframe -Mlre -Mautoinline -Mvect=sse -Mscalarsse -Mcache_align -Mflushz
-mp	thread generation for OpenMP directives
-Minfo=mp,ipa	OpenMP/Interprocedural Opt. reporting

Compilers - Best Practice

- Normal compiling for Ranger

intel **icc/ifort** **-O3 -ipo -xW** **prog.c/cc/f90**
pgi **pgcc/pgcpp/pgf95** **-fast -tp barcelona-64**
-Mipa=fast,inline **prog.c/cc/f90**
gnu **gcc** **–O3 –fast –xipo** **-mtune=barcelona **
-march=barcelona **prog.c**

- O2 is default opt, if this breaks, then most likely code has a bug.

- The effects of -xW and -xO options may vary

- Don't include debug options for a production compile!

ifort –O2 ~~–g~~ ~~–CB~~ test.c

Libraries

- Use libraries Optimized for specific Architectures
- Use library routines instead of hand-coding your own
  In "hot spots", never write library functions by hand.
- Vendors supply HPC libs for their platforms.
  IBM: ESSL/PESSL, Intel: MKL for x86-64, AMD: ACML,
  Cray: libsci, SGI:SCSL for SGI, etc.
- Libs can be 100x faster than Numerical Recipes codes.

TACC

Linux x86-64 (Lonestar/Ranger) Libraries - 3rd Party Applications

Performance

- gprof
- TAU
- PAPI
- DDT
- …

Math Libs

- SPRNG
- Metis/parmetis
- FFTW (2/3)
- MKL
- GSL
- GotoBLAS

Method Libs

- PETSc
- PLAPACK
- SCALAPACK
- SLEPc
- …

Applications

- Amber
- NAMD
- GROMACS
- Gamess
- NWchem
- …

I/O

- NetCDF
- HDF (4/5)
- Parallel I/O
- GridFTP
- …

Intel MKL 10.0 (Math Kernel Library)

- Optimized for the IA32, x86-64, IA64 architectures
- Supports both Fortran and C interfaces
- Includes functions in the following areas:
  - BLAS (levels 1-3)
  - LAPACK
  - FFT routines
  - … others
  - Vector Math Library (VML)

Intel MKL 10.0 (Math Kernel Library)

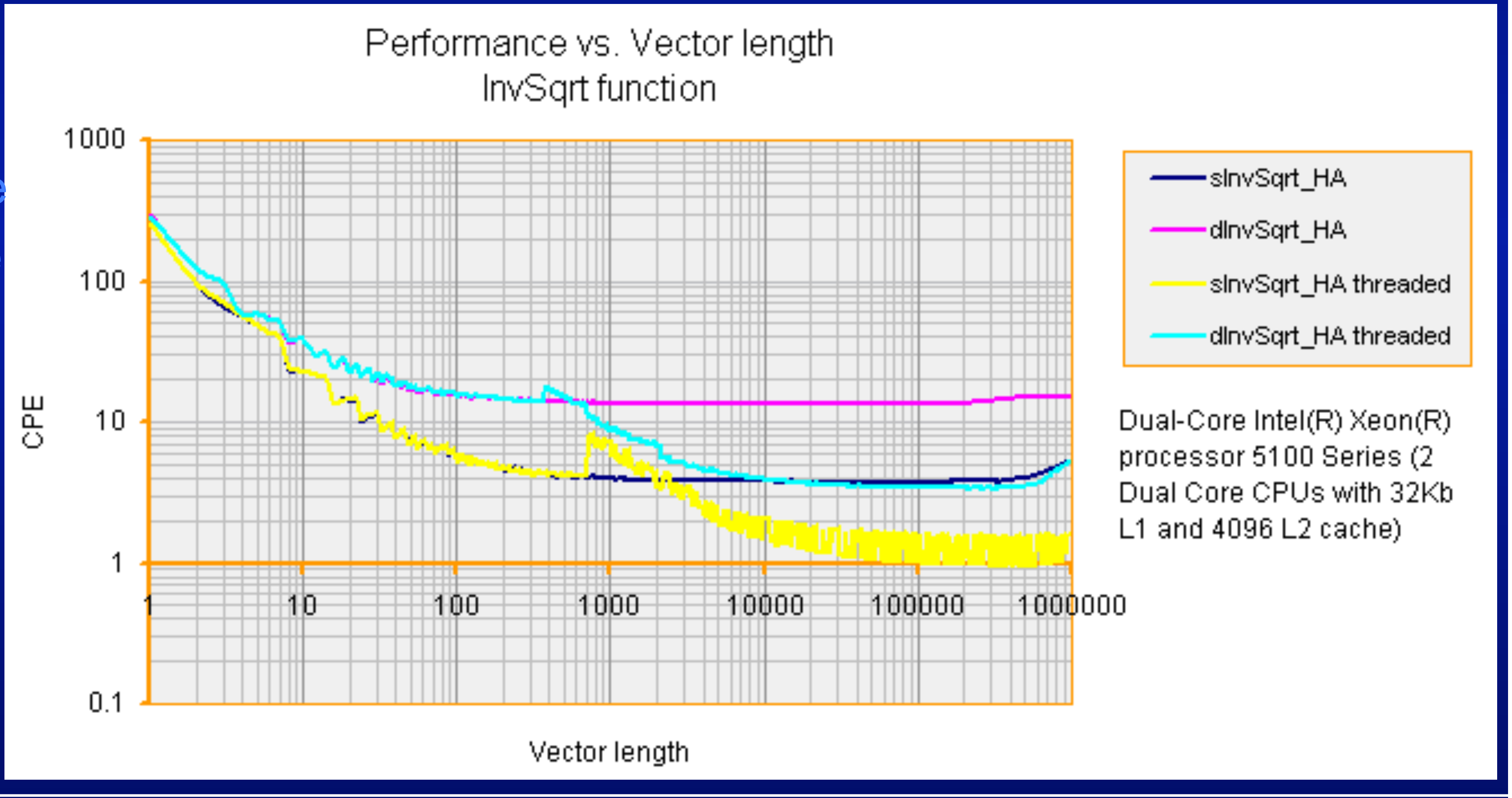
- Components:

BLAS	Basic Linear Alg.
BLAS 95	F95 interface
Sparse BLAS	
PBLAS	Parallel
LAPACK	Lin. Eq., Lst. Sqr., Eigen
LAPACK 95	F95 interface
ScaLAPACK	Parallel
VML	Vector Math Lib
VSL	Vector Stat. Lib.
DSS/PARDISO*	Parallel Direct Sparse Solver
ISS solvers	
Optimization (Trust-Region) solvers	
FFT	
FFTW	Faster FFT
Cluster FFT	Parallel
PDE	Trigonometric Trans. / Poisson Solvers
GMP	GNU Multi-Precision Functions

Intel MKL 10.0 (Math Kernel Library)

- Vector Math Library is often overlooked as a resource for optimization. The reciprocal (inverse) of the square root function is a case, in point:

Sequences above 10 elements have a large dividend.



Library Headaches

- Where the problems begin!

```
% ifort bcholf.f90
bcholf.o: In function `MAIN__':
bcholf.f90:(.text+0x898): undefined reference to `dpotrf_'
bcholf.f90:(.text+0x969): undefined reference to `dtrsm_'
bcholf.f90:(.text+0xa0d): undefined reference to `dsyrk_'
```

So, let’s take a look at how to incorporate libraries first.

Incorporating Libraries

- Static Libraries
  - end in `.a` (e.g. libalgos.a)
  - are included in executable
  - can be seen in code with `nm` command
- Dynamic Libraries
  - End in `.so` (e.g. libalgos.so)
  - Are not included in executable
  - finds entries at run time from path from env. or system

Linking Libraries

Using Compiler/Linker

mpicc and mpif90 can be used for compiling and/or linking (often called loading).

e.g.

mpicc –c a.c

mpicc –c fun.c

mpicc –o a.exe a.o fun.o

mpif90 –c a.f90

mpif90 –c fun.f90

mpif90 –o a.exe a.o fun.o

TACC

Loading Libraries

- Provides list and terse explanation of all options

  % mpicc/f90 --help

  % ld --help

  % view `which mpirun` (if mpirun is a script)

- Displays the compiler and loader commands/options

  % mpicc/f90 –v my_complete_app.c/f90

  e.g. <Compiler> -Ox -D… -I… etc.

  ld -dynamic-linker …–L… -l… etc.

Loading Libraries

- Loader options are prefixed by –Wl
  (careful, unknown options are silently dropped)

  % mpicc/f90 –Wl,<*options*>   my_complete_app.c/f90

  E.g. -Wl,--verbose,-s, …
  - Very much detail with verbose
  - s removes symbols from executable (reduces executable size)

```
% mpif90  mpihello.f90
% ls -l a.out
-rwx... 1 joeuser G-25072 11984 May 14 10:35 a.out
% mpif90 -Wl,-s mpihello.f90
% ls -l a.out
-rwx... 1 joeuser G-25072  8160 May 14 10:35 a.out
```

TACC

Libraries

- Compilers create objects (.o files) that can be grouped together in an archive (.a files)

  Commands:
  % icc/pgcc/gcc   –c funa.c funb.c   → creates funa.o & funb.o
  % ar –cr –o libfun.a funa.o funb.o → creates libfun.a

Static --Loading Libraries

Two ways to load static libraries.

1.) Specify full path:

```
% mpicc/f90  -o a.exe a.c     $HOME/libs/libfun.a
```

2.) Loader option with path-list, & abbreviated lib name:

```
% mpicc/f90  -o a.exe a.c  -L$HOME/libs  -lfun
```

Assume location of libfun.a is $HOME/libs

TACC

Static Loading Libraries

- The files (libfun.a, b.o, and libgun.a) are read in order.

  mpicc/f90 -o a.exe a.c –L$HOME/libs –lfun b.o –lgun

- If the linker cannot recognize the format of an object file (object or archive, .o or .a), it will assume that it is a linker script. More on this later.

Dynamic Loading Libraries

- Use path-list for dynamic libraries (same as static)

  % mpicc/f90 ___ -o a.exe a.c -L$HOME/libs -lfun

  Loader searches for .so first (then .a file). Often .so files are linked to a specific version numbered .so file (e.g. libfun.so.1 or libfun.so.2).

  BUT, at run time loader must know where to find the library!

Assume location of libfun.a is $HOME/libs

Dynamic Loading Libraries

- Path at runtime is resolved from
  1. path specified by rpath loader option
  2. ld searches the system library directories, as defined by /etc/ld.so.conf (configuration file for ldconfig)
  3. Finally, paths in $LD_LIBRARY_PATH are searched (colon separated list).

TACC

Dynamic Loading Libraries

- ld rpath option embeds path explicitly in executable:

```
mpicc/f90  -Wl,-rpath,$HOME/libs   a.c  -L$HOME/libs  -lfun
```

- For multiple libs in $DIRa, $DIRb, $DIRc (etc.):

```
mpicc/f90  -Wl,-rpath=$DIRa:$DIRb:$DIRc  a.c  \
           -L$DIRa  -lfuna -L$DIRb  -lfunb -L$DIRc  -lfunc
```

Assume location of libfun.a is $HOME/libs

TACC

Dynamic Loading Libraries

- Use objdump to see the rpath variables:

```
% objdump -p a.out | grep RPATH
RPATH   /opt/apps/intel10_1/mvapich/1.0.1/lib/shared:/opt/apps/intel/10.1/fc/lib:/opt/…
```

- Use ldd to print ld (loader) dependencies

```
% ldd a.out
libmpich.so.1.0 => /opt/apps/intel10_1/mvapich/1.0.1/lib/libmpich.so.1.0  (0x00002a95557000)
libfun.so       => /users/jouser/libfun.so                                 (0x00002a9587a000)
…
```

TACC

Ldconfig's dynamic linker run time bindings

% cat /etc/ld.so.conf contains list of directories

```
include ld.so.conf.d/*.conf
/lib64
/usr/lib64
/usr/kerberos/lib64
/opt/nmi/lib
/usr/lib64/qt-3.1/lib
/usr/lib64/mysql
/usr/X11R6/lib64
```

```
ibutils.conf
…
ofed.conf
qt-x86_64.conf
xorg-x11-i386.conf
xorg-x11-x86_64.conf
```

% ldconfig --print-cache lists libraries found in ld.so.conf

```
1082 libs found in cache `/etc/ld.so.cache'
    libzvt.so.2 (libc6,x86-64) => /usr/lib64/libzvt.so.2
    libz.so.1 (libc6,x86-64) => /usr/lib64/libz.so.1
    libz.so.1 (libc6) => /usr/lib/libz.so.1
    libz.so (libc6,x86-64) => /usr/lib64/libz.so
    libz.so (libc6) => /usr/lib/libz.so
```

TACC

System & Multiple HPC Math Libraries

- MPI Considerations
  - Many systems have more than one compiler
  - Many systems have more than one version of MPI
    use modules to manage dependencies
- Multi-library codes
  - Requires ordering
  - Check to make sure correct library is loaded

MPI Considerations -- Libraries

- Compiler and compiled-MPI must match.
  - (pairs: PGI/mvapich, Intel/mvapich – they are dependent)
- MPI environment comes with mpirun
  - mpirun sets up environment variables (for numa, polling, etc.) for specific libraries at run time
- Other libraries may be dependent on compiler/MPI

Changing Environments -- Libraries

- module unload <mpia>
- module swap <compilera> <compilerb>
- module load <mpib>

- TACC defines general environment variables in modules. Syntax: TACC_<pkg>_LIB
  - E.G.

```
% module load mkl
% echo $TACC_MKL_LIB
/opt/apps/intel/mkl/10.0.1.014/lib/em64t
```

Changing Environments -- Libraries

- E.G.

% mpif90 mpihello.f90 {mpif90 uses a –Wl,-rpath}

% objdump -p a.out | grep RPATH

RPATH /opt/apps/pgi7_2/mvapich/1.0.1/lib/shared:/share/apps/pgi/7.2…

% module unload mvapich

% module swap pgi intel

% module load mvapich

% mpif90 mpihello.f90 {mpif90 uses a –Wl,-rpath}

% objdump -p a.out | grep RPATH

RPATH /opt/apps/intel10_1/mvapich/1.0.1/lib/shared:/opt/apps/intel/10.1/fc/lib: /opt/…

Use the same environment in batch jobs, even when using rpath→ mpirun!

TACC

Library Examples

- GotoBLAS
- Intel MKL
- Multiple Libs
- Scalapack
- PETSc

Goto Libraries (static, dynamic)

- LP64 and ILP64
  - Library uses LP64 and ILP64 in name **libgoto_ilp64…** **libgoto_lp64…**
- Multi-threaded Issues
  - Uses mp in name for multi-threaded version **libgoto_lp64.a** **libgoto_lp64_mp.a**
- Has both static and "dynamic" libraries **.so and .a files**

Examples:

% module load gotoblas

% mpicc $TACC_GOTOBLAS_LIB/libgoto_lp64.a prog.c

% mpif90 –L$TACC_GOTOBLAS_LIB/ -lgoto_lp64   prog.f90

See next section on groups.

MKL Libraries (static, dynamic)

- LP64 and ILP64 – you don't have to worry about this.
- Multi-threaded
  - Make sure you specify $OMP_NUM_THREADS 1 or not 1

Examples:

% module load mkl

% mpicc –L$TACC_MKL_LIB –lmkl –lguide -lpthreads prog.c –lm

% mpif90 –L$TACC_MKL_LIB –lmkl –lguide -lpthreads prog.f90

dependency

ILP64/LP64 set here

Intel "Legacy" OMP
Use -liomp5 for latest version

MKL Libraries (groups)

- The ld loader now accepts groups.
- An ".so" file can have text with a group of lib names:

```
% cat $TACC_MKL_LIB/libmkl.so
GROUP (libmkl_intel_lp64.so
       libmkl_intel_thread.so
       libmkl_core.so)

% cat $TACC_GOTOBLAS_LIB/libgoto_lp64.so
GROUP (libgoto_lp64.a  /share/apps/intel/10.1/fc/lib/libimf.so
   /share/apps/intel/10.1/fc/lib/libifcore.so.5
   /share/apps/intel/10.1/fc/lib/libsvml.so
   /share/apps/intel/10.1/fc/lib/libintlc.so.5 -ldl)
```

MAPS and Libraries

- Use –Map ld option to acquire a “memory” map
  - Object and common symbols mapped to memory
  - How common symbols are allocated
  - What symbols caused archive members to be extracted
- Syntax: … -Wl,-Map=mymap …

e.g.

Archive member included because of file (symbol) **loaded from archive**

/opt/apps/intel10_1/gotoblas/1.26/libgoto_lp64.a(dtrsm.o)

bcholf.o (dtrsm_) **Object file (program) and call (symbol)**

MAPS and Libraries

- With .PLTs external reference resolved at runtime!
  - .PLT (Procedure Load Table) -- used by MKL
  - run-time ld is responsible for relocating external reference

e.g.

```
.plt      0x0000000000402e90     0x5b0 /usr/lib/gcc/…/lib64/crt1.o
          0x0000000000402eb0           printf@@GLIBC_2.2.5
           …
          0x0000000000403290           dtrsm_
          0x00000000004032a0           dpotrf_
```

- Use --cref cross-reference option:

```
dtrsm_  /opt/apps/intel/mkl/10.0.1.014/lib/em64t/libmkl_intel_lp64.so
dsyrk_  /opt/apps/intel/mkl/10.0.1.014/lib/em64t/libmkl_intel_lp64.so
…
```

cross-ref. map listing

TACC

Multiple Libraries (static, dynamic)

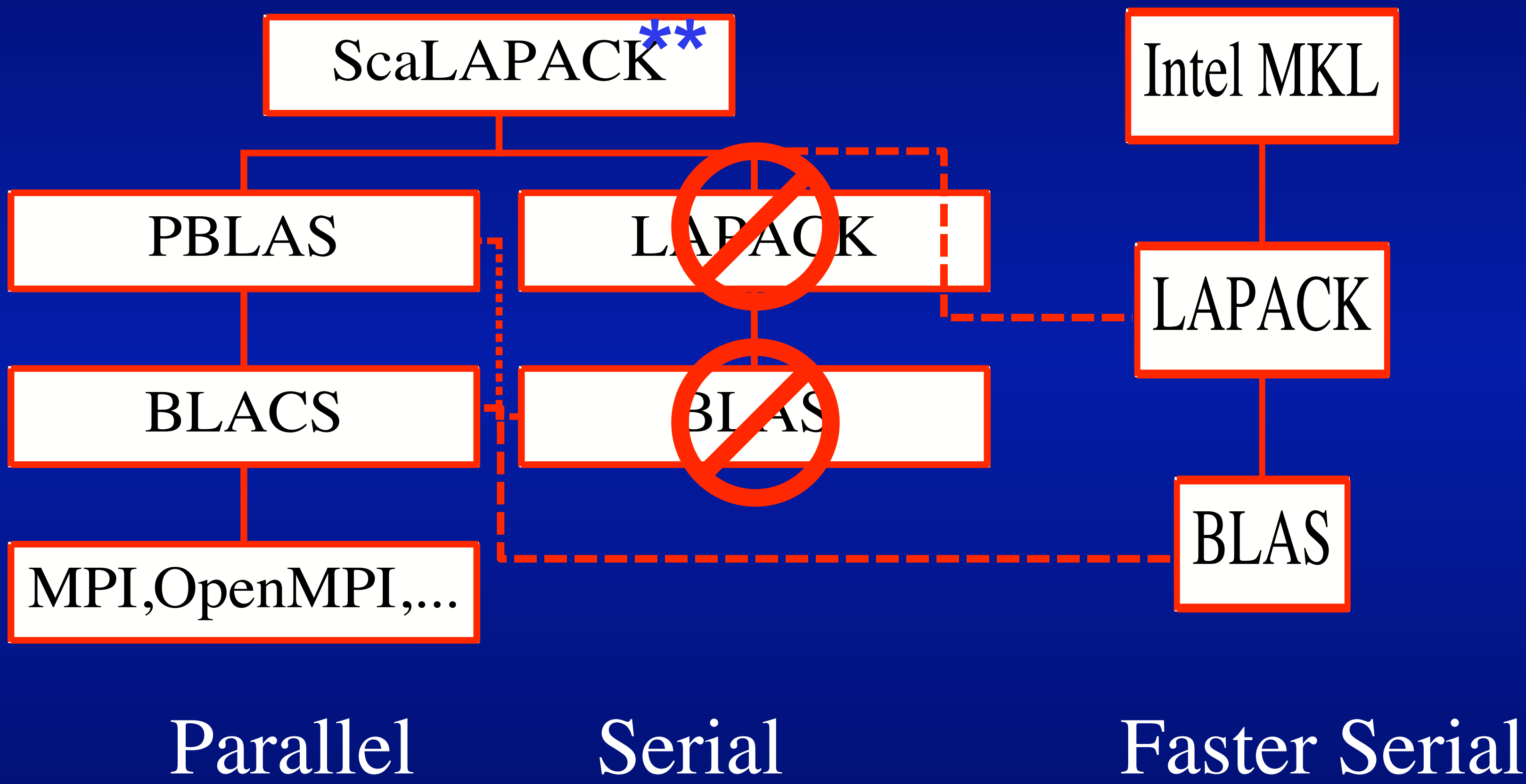
- E.G. LAPACK from gotoBLAS, all else from MKL
  - -- dtrsm, dpotrf, dsyrk from gotoBLAS and vdInvSqrt from MKL.

```
% module load gotoblas mkl
% mpif90 –O3 cholesky.f90 \
        -Wl,--cref \
        -Wl,-rpath, $TACC_MKL_LIB \
        -Wl,-Map=mymap \
        -L${TACC_GOTOBLAS_LIB} -lgoto_lp64 \
        -L${TACC_MKL_LIB}          -lmkl -lguide -lpthread
```

TACC

ScaLAPACK Libraries

Use Netlib Parallel calls,and Intel Serial libs.



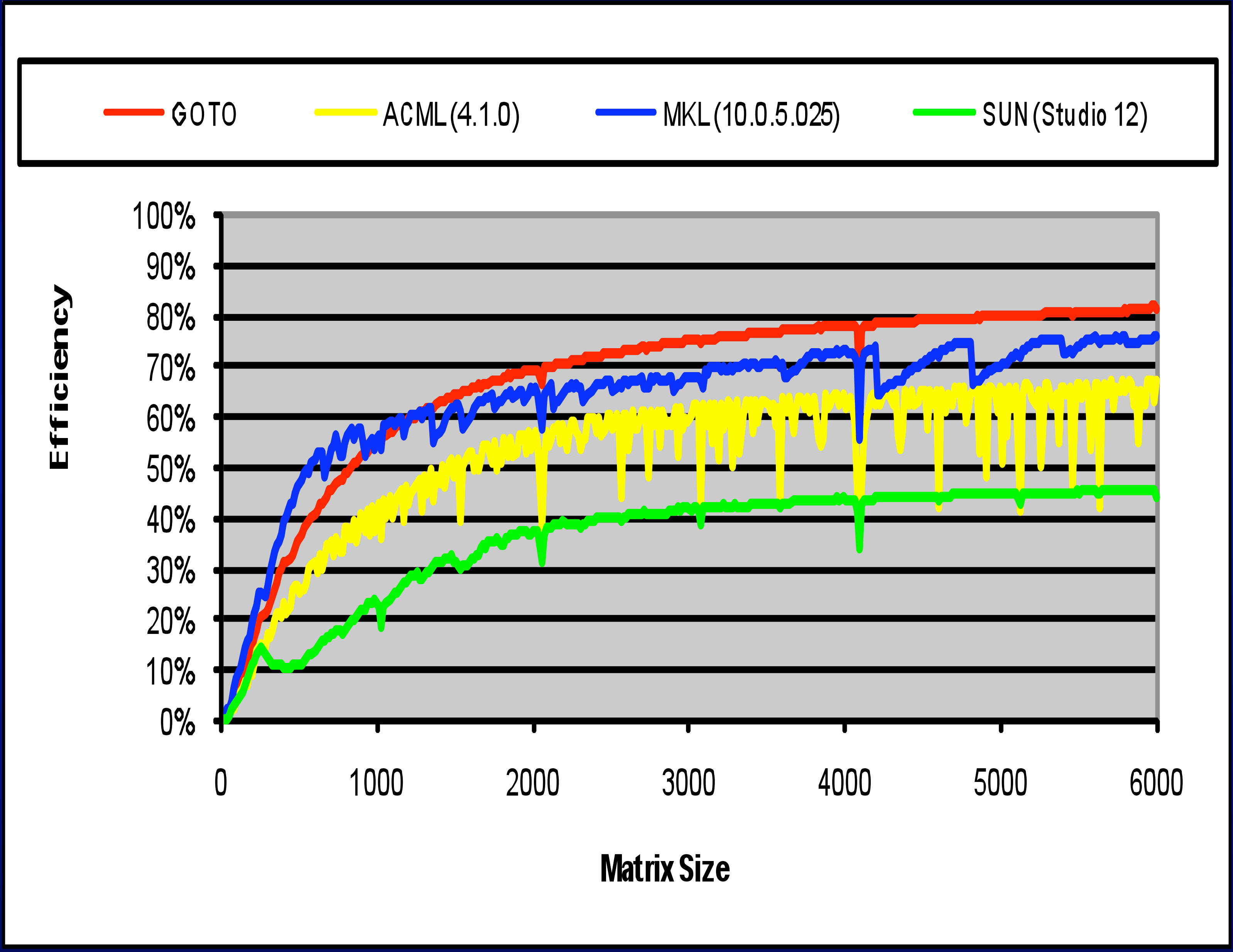
**www.netlib.org/scalapack/index.html

Netlib ScaLAPACK with MKL LAPACK/BLAS

```
% module load scalpack mkl
% mpif90 –O3 psgesv.f90 \
     -Wl,--cref -Wl,-Map,mymap \
     -Wl,-rpath=TACC_MKL_LIB:$TACC_SCALAPACK_LIB \
     -L${TACC_SCALAPACK_LIB}                         \
     ${TACC_SCALAPACK_LIB}/blacs_MPI-LINUX-0.a       \
     -lscalapack                                     \
     ${TACC_SCALAPACK_LIB}/blacsCinit_MPI-LINUX-0.a \
     ${TACC_SCALAPACK_LIB}/blacs_MPI-LINUX-0.a       \
     -L${TACC_MKL_LIB} -lmkl -lmkl_lapack -lguide
```

complications with interface, blacs loaded twice.

GotoBLAS Performance



Cactus, PTETSc & MKL Libraries

Things can get complicated!

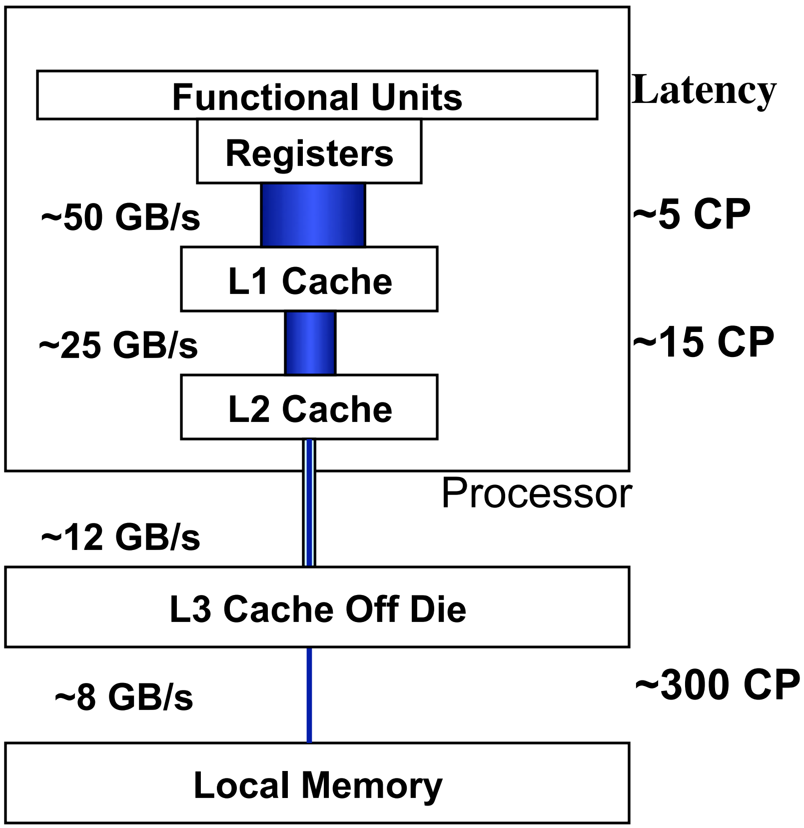
```
mpicxx -o SandTank -O2 datestamp.o
-L$CACTUS/lib -lthorn_Cactus -lthorn_CactusBindings -lthorn_BlackOilEvolve -
lthorn_PUGHInterp -lthorn_IOBasic -lthorn_TimerReport -lthorn_Time -lthorn_SymBase -
lthorn_LocalReduce -lthorn_LocalInterp -lthorn_Boundary -lthorn_IOJpeg -lthorn_jpeg6b -
lthorn_HTTPDExtra -lthorn_HTTPD -lthorn_Socket -lthorn_IOASCII -lthorn_IOUtil -
lthorn_IDBlackOil -lthorn_CartGrid3D -lthorn_CoordBase -lthorn_Sandtank -lthorn_LocalToGlobal
-lthorn_BlackOilBase -lthorn_NaNChecker -lthorn_PUGHReduce -lthorn_PUGHSlab -
lthorn_PUGH -lthorn_Cactus -lthorn_CactusBindings  \
-L/opt/apps/pgi7_2/mvapich/1.0.1/lib -L/opt/ofed//lib64/ \
-L/opt/apps/pgi7_2/mvapich1_1_0_1/petsc/2.3.3/lib/barcelona  \
-L/usr/X11R6/lib \
-L/opt/apps/pgi7_2/mvapich/1.0.1/lib \
-lmpich -libverbs -libumad -lpthread  -lrt  \
-lpetscts -lpetscsnes -lpetscksp -lpetscdm -lpetscmat -lpetscvec -lpetsc \
-lmpich -libverbs -libumad -lpthread -lrt -lcrypt \
-lpgf90 -lpgf90rtl -lpgftnrtl -lpgf90_rpm1 -lpghpf2 -lpgc -lm
-L/opt/apps/intel/mkl/10.0.1.014/lib/em64t/ -lmkl_em64t -lmkl -lguide -lpthread -lm
```

Cactus Libs
MPI Libs
PETSc Libs
X11 Libs
PGI Libs
MKL Libs

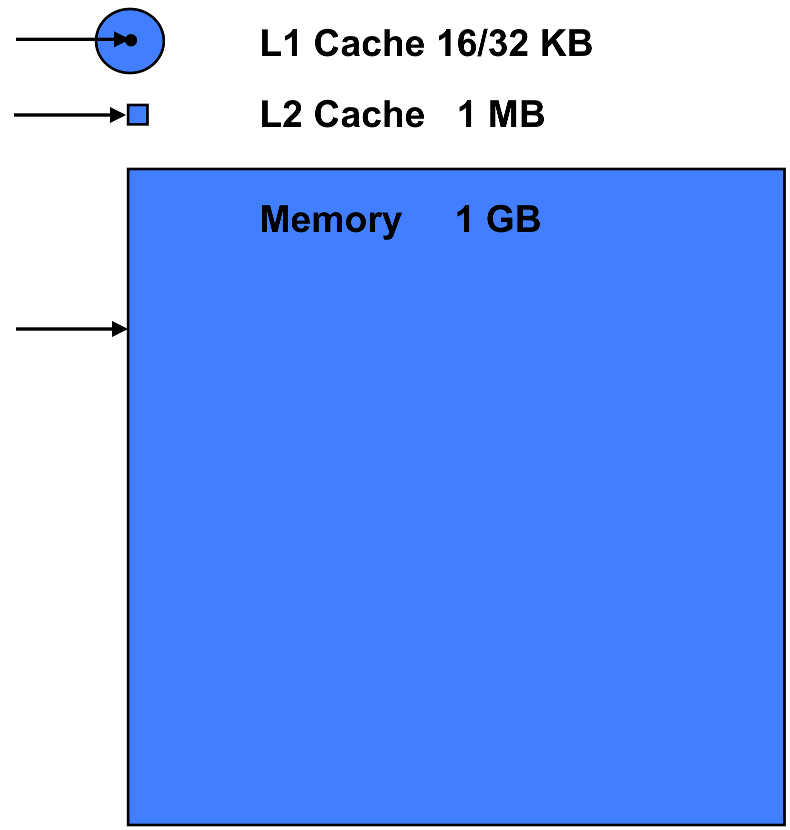
mpi reference twice.

Approx. Memory Bandwidths & Sizes

Relative Memory Bandwidths



Relative Memory Sizes



References

- Books

  High Performance Computing by Kevin Dowd and Charles Severance (O'Reilly book) -- general study of high performance computing

  Performance Optimization of Numerically Intensive Codes by Stefan Goedecker and Adolfy Hoisie (Siam book, Society for Industrial and Applied Mathematics)

- TACC User Guides

  www.tacc.utexas.edu/services/userguides/ranger/
  www.tacc.utexas.edu/services/userguides/lonestar/

- Compilers

  www.intel.com/cd/software/products/asmo-na/eng/compilers/278607.htm
  www.intel.com/cd/software/products/asmo-na/eng/compilers/279831.htm
  www.pgroup.com/doc/pgiug.pdf

- Optimization

  http://cache-www.intel.com/cd/00/00/21/92/219281_compiler_optimization.pdf

References

- Libraries

  GotoBLAS   www.tacc.utexas.edu/resources/software/

  Dense and band matrix software (**ScaLAPACK**)

    www.netlib.org/scalapack/scalapack_home.html

  Large sparse eigenvalue software (**PARPACK** and **ARPACK**)

    www.caam.rice.edu/software/ARPACK/

- Math Functions (MKL, VML)

http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation

http://www.intel.com/software/products/mkl/data/vml/functions/_listfunc.html

Perguntas