Math Kernel Library

The Intel Math Kernel Library (MKL) is a library of optimized, general-purpose math software. The routines are optimized and threaded, and CISL recommends using them on Cheyenne for applications that otherwise spend substantial computational time in non-optimized routines that do the same calculations.

In addition to using the examples below, you may find these links useful.

- Math Kernel Library Link Line Advisor
- MKL Documentation

Check the version numbers to confirm that they are compatible with those of the Intel compiler and MKL you are using.

Page contents

Prerequisites

Before running the examples below, load these environment modules:

- **ncarenv**
- **intel**
- **ncarcompilers**
- **mkl**
- **mpt**

Loading the MKL module gives you access to many MKL examples. Examples 2 and 3 below show how to copy the files and untar them in a directory of your own.

> The examples below assume that you are using SGI's Message Passing Toolkit (MPT) MPI library and that you compiled the executables with an Intel Fortran compiler loaded with a Cheyenne modulefile. For help with adapting them for other cases, contact the NCAR Research Computing help desk.

Example 1: Sequential program calling an MKL routine

This example shows how to use the Intel compiler **ifort** to compile a program with MKL dependencies. Though this program is sequential, i.e. contains no OpenMP directives, there is great advantage to compiling and linking it with ifort's **-qopenmp** option, because the MKL library is in fact threaded and will run faster if you use that option.

The program in this example calls BLAS subroutine DGEMM, which is included in MKL. To begin, create a file named **tdgemm.f90** with the following source code.

```
program main
! This program uses subroutine DGEMM
! to perform matrix-matrix operation
! c := alpha*op( a )*op( b ) + beta*c
!
 implicit none
  integer :: i, j
  character*1 ::   transa, transb
  data transa, transb/'T', 'T'/
  double precision :: alpha, beta
  data alpha, beta/0.50d0, -1.20d0/
  integer :: m, n, k
  parameter (m=16000, k=16000, n=16000)
  double precision :: a(m,k), b(k,n), c(m,n)
!
  print *, "This example computes real matrix C=alpha*A*B+beta*C"
  print *, "using Intel® MKL function DGEMM, where a, b, and c"
  print *, "are matrices and alpha and beta are double precision "
  print *, "scalars"
  print *, ""

  print *, "Initializing data for matrix multiplication C=A*B for "
  print 10, " matrix a(",m," x",k, ") and matrix b(", k," x", n, ")"
  print *, ""
  alpha = 1.0
  beta = 0.0
!
  print *, "Intializing matrix data"
  a = 1.0
  b = 1.1
  c = 0.0
!
  print *, "Computing matrix product using Intel® MKL DGEMM "
  print *, "subroutine"
  call DGEMM('n','n',m,n,k,alpha,a,m,b,k,beta,c,m)
  print *, "Computations completed."
  print *, ""
!
  print *, "Top left corner of matrix A:"
  print 20, ((a(i,j), j = 1,min(k,6)), i = 1,min(m,6))
  print *, ""
  print *, "Top left corner of matrix B:"
  print 20, ((b(i,j),j = 1,min(n,6)), i = 1,min(k,6))
  print *, ""
  print *, "Top left corner of matrix C:"
  print 30, ((c(i,j), j = 1,min(n,6)), i = 1,min(m,6))
  print *, ""
!
 10   format(a,i5,a,i5,a,i5,a,i5,a)
 20   format(6(f12.0,1x))
 30   format(6(es12.4,1x))
!
  print *, "Example completed."
end program main
```

Be sure you have the necessary environment modules loaded. (See *Prerequisites* above.) Compile the program as shown here:

```
ifort -O2 -qopenmp -o tdgemm.exe tdgemm.f90
```

Next, create a batch script following the example here and submit it with a **qsub** command. It will run on one node using 8 processors.

```
#!/bin/tcsh
#PBS -N omp
#PBS -A project_code
#PBS -l walltime=00:10:00
#PBS -q regular
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -l select=1:ncpus=1:ompthreads=8

mkdir -p /glade/scratch/$USER/temp
setenv TMPDIR /glade/scratch/$USER/temp

### Run the executable
omplace -nt $OMP_NUM_THREADS ./tdgemm.exe

#endjob
```

The expected output includes lines like this:

```
  1.7600E+04   1.7600E+04   1.7600E+04   1.7600E+04   1.7600E+04   1.7600E+04
  1.7600E+04   1.7600E+04   1.7600E+04   1.7600E+04   1.7600E+04   1.7600E+04
```

It is a good exercise to rerun the job with this alternative for the select statement, then compare the timings of the two jobs:

```
#PBS -l select=1:ncpus=1:ompthreads=1
```

Timings in one comparison showed the job with 8 threads ran in 38 seconds and the other took 4 minutes, 20 seconds. To get your own timings:

- include the **-m** and **-M** arguments in your batch script to get the information by email, or
- prepend your executable command with **/usr/bin/time -v** and results will be included in your output file.

Example 2: Intel sequential program calling an MKL routine

The official MKL example uses a program named dgemmx.f, which takes input data from the file dgemmx.d. After you get the program by following the instructions below, you can study the Fortran code to verify that it contains error-checking and error-reporting code. The shorter, heuristic program in Example 1 does not offer these advantages, even though both examples effect the same matrix-multiply calculation by calling DGEMM.

Here's how to get the example program and dependencies on Cheyenne.

1. Load the minimum recommended set of environment modules. See *Prerequisites* above.
2. Create a clean directory on Cheyenne, such as **/glade/scratch/username/mkl_dgemm**, for example.
3. Change to that directory with the **cd** command, then run these commands.

```
cp $MKLROOT/examples/examples_core_f.tgz .
gunzip examples_core_f.tgz;  tar xvf examples_core_f.tar
cd blas
make libintel64 function=dgemm
cd _results/intel_lp64_parallel_intel_iomp5_intel64_lib/
cat dgemmx.res
```

Expected numerical output (last four lines):

```
OUTPUT DATA
ARRAY C
     8.950   8.950   7.750   7.750     7.750
    19.110  19.110  17.910  17.910    17.910
```

Example 3: Using MKL with the MPI parallel programming model

This example uses MKL's implementation of Scalapack routines to perform QR factorization of a general matrix, and an MKL makefile to build a binary executable file.

Here's how to get the example program and dependencies on Cheyenne:

1. Load the minimum recommended set of environment modules. See *Prerequisites* above.
2. Create a clean directory, such as **/glade/scratch/$USER/mkl_examples**.
3. Change to that directory with the **cd** command, then run the following commands.

```
cp $MKLROOT/examples/examples_cluster_f.tgz .
gunzip examples_cluster_f.tgz;  tar xvf examples_cluster_f.tar
cd scalapackf
make libintel64 example=pcgetrfx mpi=custom
```

The **make** command in this example will fail to run the executable. You can run it yourself from a PBS interactive job or by submitting a batch script (example below) within your **scalapackf** directory.

```
#!/bin/tcsh
#PBS -N mkl_example
#PBS -A project_code
#PBS -l walltime=00:10:00
#PBS -q regular
#PBS -j oe
#PBS -k eod
#PBS -l select=1:ncpus=36:mpiprocs=36
#PBS -m abe
#PBS -M email_address

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Load modules; Intel assumed + MKL
module load mkl
module li

### Run the executable
mpiexec_mpt _results/intel_custom_lp64_intel64_lib_parallel/pcgetrfx.exe in/pcgetrfx.in output.$PBS_JOBID

### End job
```

Finally, check the results of the run. If your test is successful, your job's output file will include a 10-column table that includes a PASSED column. Each entry should be a "1" in that column.