Intel Math Kernel Library

Ramses van Zon

SciNet HPC Consortium University of Toronto

TechTalk, March 2012

SciNet compute • calcul CANADA

Introduction

- So you want to do your scientific computation, as fast and as soon as possible.
- Perhaps you are not an expert at coming up with faster algorithms that can take advantage of the specific hardware architecture available to you, or you cannot/won't spend the time to become one.
- Luckily, experts have already coded and optimized many common scientific computational tasks, and tuned them for particular hardware:

$\Rightarrow$**Vendor specific libraries**

Intel MKL

What is it and why should I use it?

- Collection og highly optimized, high-performance mathematical library for Intel chips.
- Can replace other implementations of e.g. BLAS, LAPACK and FFT.
- It's available on the GPC.
- It's easy to use (finally) when used with the intel compiler (and intelmpi for mpi).
- Take advantage of the knowledge that the Intel guys have of their own chips.

Documentation is a bit tedious, so hence this friendly TechTalk!

Introduction: MKL basic usage

```
#include <stdio.h>
#include <stdlib.h>
#include "mkl_cblas.h"

int main(int argc,char**argv)
{
  if (argc != 3)
    printf("need 2 numbers\n");
  else {
    float x = atof(argv[1]);
    float y = atof(argv[2]);
    cblas_saxpy(1,1.0,&x,1,&y,1);
    printf("Sum is:%f\n", y);
  }
}
```

Introduction: MKL basic usage

```c
#include <stdio.h>
#include <stdlib.h>
#include "mkl_cblas.h"

int main(int argc,char**argv)
{
  if (argc != 3)
    printf("need 2 numbers\n");
  else {
    float x = atof(argv[1]);
    float y = atof(argv[2]);
    cblas_saxpy(1,1.0,&x,1,&y,1);
    printf("Sum is:%f\n", y);
  }
}
```

```
$ icc example.c -mkl
```

SciNet compute • calcul CANADA

Introduction: MKL basic usage

```c
#include <stdio.h>
#include <stdlib.h>
#include "mkl_cblas.h"

int main(int argc,char**argv)
{
  if (argc != 3)
    printf("need 2 numbers\n");
  else {
    float x = atof(argv[1]);
    float y = atof(argv[2]);
    cblas_saxpy(1,1.0,&x,1,&y,1);
    printf("Sum is:%f\n", y);
  }
}
```

```
$ icc example.c -mkl   since v. 11.1
```

SciNet compute • calcul CANADA

Introduction: MKL basic usage

```c
#include <stdio.h>
#include <stdlib.h>
#include "mkl_cblas.h"

int main(int argc,char**argv)
{
  if (argc != 3)
    printf("need 2 numbers\n");
  else {
    float x = atof(argv[1]);
    float y = atof(argv[2]);
    cblas_saxpy(1,1.0,&x,1,&y,1);
    printf("Sum is:%f\n", y);
  }
}
```

```
$ icc example.c -mkl   since v. 11.1
$ ./a.out 4 5
```

SciNet compute • calcul CANADA

Introduction: MKL basic usage

```c
#include <stdio.h>
#include <stdlib.h>
#include "mkl_cblas.h"

int main(int argc,char**argv)
{
  if (argc != 3)
    printf("need 2 numbers\n");
  else {
    float x = atof(argv[1]);
    float y = atof(argv[2]);
    cblas_saxpy(1,1.0,&x,1,&y,1);
    printf("Sum is:%f\n", y);
  }
}
```

```
$ icc example.c -mkl   since v. 11.1
$ ./a.out 4 5
Sum is:9.000000
```

SciNet compute • calcul CANADA

Introduction: MKL basic usage

```c
#include <stdio.h>
#include <stdlib.h>
#include "mkl_cblas.h"

int main(int argc,char**argv)
{
  if (argc != 3)
    printf("need 2 numbers\n");
  else {
    float x = atof(argv[1]);
    float y = atof(argv[2]);
    cblas_saxpy(1,1.0,&x,1,&y,1);
    printf("Sum is:%f\n", y);
  }
}
```

```
program example
real :: a,x,y
character(len=32) :: s1,s1
if(command_argument_count().ne.2)
then
  print *, "need 2 numbers"
else
  call get_command_argument(1,s1)
  call get_command_argument(2,s2)
  read (s1,*) x
  read (s2,*) y
  call saxpy(1,1.0,x,1,y,1)
  print *, "Sum is:", y
end if
end program example
```

```
$ icc example.c -mkl   since v. 11.1
$ ./a.out 4 5
Sum is:9.000000
```

SciNet compute • calcul CANADA

Introduction: MKL basic usage

```c
#include <stdio.h>
#include <stdlib.h>
#include "mkl_cblas.h"

int main(int argc,char**argv)
{
  if (argc != 3)
    printf("need 2 numbers\n");
  else {
    float x = atof(argv[1]);
    float y = atof(argv[2]);
    cblas_saxpy(1,1.0,&x,1,&y,1);
    printf("Sum is:%f\n", y);
  }
}
```

```
$ icc example.c -mkl   since v. 11.1
$ ./a.out 4 5
Sum is:9.000000
```

```
program example
real :: a,x,y
character(len=32) :: s1,s1
if(command_argument_count().ne.2)
then
  print *, "need 2 numbers"
else
  call get_command_argument(1,s1)
  call get_command_argument(2,s2)
  read (s1,*) x
  read (s2,*) y
  call saxpy(1,1.0,x,1,y,1)
  print *, "Sum is:", y
end if
end program example
```

```
$ ifort example.f90 -mkl  since v. 11.1
$ ./a.out 4 5
 Sum is:  9.000000000000000
```

SciNet compute • calcul CANADA

Introduction: MKL Mathematical Functionality

A subset of what can it do:

1. Basic Linear Algebra
   BLAS, Sparse BLAS, PBLAS
2. Linear solvers
   LAPACK, ScaLAPACK1, Sparse Solver routines
3. Vector Mathematical & Statistical functions
   VML, VSL
4. Fourier Transform functions
   incl. MPI versions and FFTW wrappers

Introduction: MKL Technical Functionality

Programming Languages

1. Fortran 77
2. Fortran 90/95
3. C/C++ (sometimes through linking Fortran routines)

Works best (and easiest) with intel compilers and intelmpi.

Introduction: MKL Technical Functionality

Programming Languages

1. Fortran 77
2. Fortran 90/95
3. C/C++ (sometimes through linking Fortran routines)

Works best (and easiest) with intel compilers and intelmpi.

Parallelism

1. Sequential: -mkl=sequential
2. Threaded: -mkl=parallel (default)
3. MPI: -mkl=cluster

1. Basic Linear Algebra

There is a fortran standard for basic linear algebra subroutines.

MKL is only one of the implementations, and has

- BLAS
- CBLAS
- Sparse BLAS
- PBLAS

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 8 & 9 \\ 2 & 7 & 10 \\ 3 & 6 & 11 \\ 4 & 5 & 12 \end{bmatrix} = \begin{bmatrix} 50 & 94 & 178 \\ 60 & 120 & 220 \end{bmatrix}$$

SciNet compute • calcul CANADA

1. Basic Linear Algebra

BLAS

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} \times \begin{vmatrix} 2 & 7 & 10 \\ 3 & 6 & 11 \\ 4 & 5 & 12 \end{vmatrix} = \begin{bmatrix} 50 & 94 & 178 \\ 60 & 120 & 220 \end{bmatrix}$$

SciNet compute • calcul CANADA

1. Basic Linear Algebra

BLAS

Level 1: vector-vector operations

ex. $\vec{\mathbf{y}} \leftarrow \alpha\vec{\mathbf{x}} + \vec{\mathbf{y}}$: ?axpy,?=s,d,c,z

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} \times \begin{vmatrix} 2 & 7 & 10 \\ 3 & 6 & 11 \\ 4 & 5 & 12 \end{vmatrix} = \begin{bmatrix} 50 & 94 & 178 \\ 60 & 120 & 220 \end{bmatrix}$$

SciNet compute • calcul CANADA

1. Basic Linear Algebra

BLAS

Level 1: vector-vector operations

ex. $\vec{\mathbf{y}} \leftarrow \alpha\vec{\mathbf{x}} + \vec{\mathbf{y}}$: ?axpy,?=s,d,c,z

Level 2: matrix-vector operations

ex. $\vec{\mathbf{y}} \leftarrow \alpha\mathbf{A}\vec{\mathbf{x}} + \beta\vec{\mathbf{y}}$: ?gemv

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} \times \begin{vmatrix} 2 & 7 & 10 \\ 3 & 6 & 11 \\ 4 & 5 & 12 \end{vmatrix} = \begin{bmatrix} 50 & 94 & 178 \\ 60 & 120 & 220 \end{bmatrix}$$

SciNet compute • calcul CANADA

1. Basic Linear Algebra

BLAS

Level 1: vector-vector operations

ex. $\vec{\mathbf{y}} \leftarrow \alpha\vec{\mathbf{x}} + \vec{\mathbf{y}}$: ?axpy,?=s,d,c,z

Level 2: matrix-vector operations

ex. $\vec{\mathbf{y}} \leftarrow \alpha\mathbf{A}\vec{\mathbf{x}} + \beta\vec{\mathbf{y}}$: ?gemv

Level 3: matrix-matrix operations

ex. $\mathbf{C} \leftarrow \mathbf{AB} + \alpha\mathbf{C}$: ?gemm,?symm

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} \times \begin{vmatrix} 2 & 7 & 10 \\ 3 & 6 & 11 \\ 4 & 5 & 12 \end{vmatrix} = \begin{bmatrix} 50 & 94 & 178 \\ 60 & 120 & 220 \end{bmatrix}$$

SciNet compute • calcul CANADA

1. Basic Linear Algebra

BLAS

Level 1: vector-vector operations

ex. $\vec{\mathbf{y}} \leftarrow \alpha\vec{\mathbf{x}} + \vec{\mathbf{y}}$: ?axpy,?=s,d,c,z

Level 2: matrix-vector operations

ex. $\vec{\mathbf{y}} \leftarrow \alpha\mathbf{A}\vec{\mathbf{x}} + \beta\vec{\mathbf{y}}$: ?gemv

Level 3: matrix-matrix operations

ex. $\mathbf{C} \leftarrow \mathbf{AB} + \alpha\mathbf{C}$: ?gemm,?symm

- Standard but bit cryptic (saxpy,dgemm,...) and 'very fortran'

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} \times \begin{vmatrix} 2 & 7 & 10 \\ 3 & 6 & 11 \\ 4 & 5 & 12 \end{vmatrix} = \begin{bmatrix} 50 & 94 & 178 \\ 60 & 120 & 220 \end{bmatrix}$$

SciNet compute • calcul CANADA

1. Basic Linear Algebra

BLAS

Level 1: vector-vector operations

ex. $\vec{\mathbf{y}} \leftarrow \alpha\vec{\mathbf{x}} + \vec{\mathbf{y}}$: ?axpy,?=s,d,c,z

Level 2: matrix-vector operations

ex. $\vec{\mathbf{y}} \leftarrow \alpha\mathbf{A}\vec{\mathbf{x}} + \beta\vec{\mathbf{y}}$: ?gemv

Level 3: matrix-matrix operations

ex. $\mathbf{C} \leftarrow \mathbf{AB} + \alpha\mathbf{C}$: ?gemm,?symm

- Standard but bit cryptic (saxpy,dgemm,...) and 'very fortran'
- For 'full' matrices.
  For sparse vectors/matrices, MKL has separate routines.

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} \times \begin{vmatrix} 2 & 7 & 10 \\ 3 & 6 & 11 \\ 4 & 5 & 12 \end{vmatrix} = \begin{bmatrix} 50 & 94 & 178 \\ 60 & 120 & 220 \end{bmatrix}$$

SciNet compute • calcul CANADA

BLAS example

```
program dsymmtest
  integer, parameter ::  size = 4096
  real*8, allocatable, dimension(:,:)  ::  a,b,c
  allocate(a(size,size))
  allocate(b(size,size))
  allocate(c(size,size))
  a = 2.0d0
  b = -3.0d0
  c = 0.0d0
  call dsymm('L','U',size,size,1.0d0,a,size,b,size,0.0d0,c,size)
  do j=1,size
    do i=1,size
      if (2.0d0*-3.0d0*size.ne.c(i,j)) then
        print *, "Test failed"
      end if
    end do
  end do
  deallocate(a)
  deallocate(b)
  deallocate(c)
```

Why not use MATMUL?

- Matmul is slower.
- Intel compiler does not replace matmul with call to BLAS.
- Matmul also does not get parallelized.

Why not use MATMUL?

- Matmul is slower.
- Intel compiler does not replace matmul with call to BLAS.
- Matmul also does not get parallelized.

Parallization

- On-node parallelization (threaded) is easy:

  `-mkl=parallel`

- Can be used together with other OpenMP, but not nested.
- Obeys `OMP_NUM_THREADS`
- Using hyperthreading (`OMP_NUM_THREADS`=16) can hurt MKL performance.
- Off-node: PBLAS (Distributed data)
  Fortran only. From C, you can call these, with some twists.
  Note: **`mkl_pblas.h`** at least declares the Fortran functions.

Calling Fortran functions from C

First off: this is somewhat compiler specific, but we're already using intel's, so that seems okay.

- Function name in C appears with appended underscore _.
- Translate types
- Function arguments are all pointers. Passing constants requires storing them in a variable an using a pointer to it.
- All arrays should be contiguous in memory. Default C layout of 2d arrays is the transposed of the default Fortran layout.
- Character strings as fortran function arguments are special, in C the length of the string must be passed as a separate parameter following the `char *`.
- May need to link with **`ifort`**.

Do this when you must, but don't if there is a C interface.

SciNet compute • calcul CANADA

CBLAS

CBLAS Is a C interface to the Fortran BLAS calls

- Requires inclusion of header file **`mkl_cblas.h`**
- Function names are **`cblas_<blasname>`** (still cryptic).
- No need to worry about mixed language calls.
- Support for row-major matrices.

CBLAS

CBLAS Is a C interface to the Fortran BLAS calls

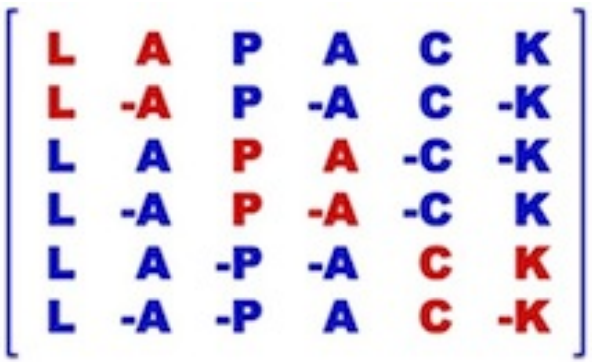
- Requires inclusion of header file **mkl_cblas.h**
- Function names are **cblas_<blasname>** (still cryptic).
- No need to worry about mixed language calls.
- Support for row-major matrices.

```
#include <stdio.h>
#include <stdlib.h>
#include "mkl_cblas.h"
int main(int argc,char**argv){
  if (argc != 3) printf("need 2 numbers\n");
  else {
    float x = atof(argv[1]), y = atof(argv[2]);
    cblas_saxpy(1,1.0,&x,1,&y,1);
    printf("Sum is:%f\n", y);
  }
```

2. Linear Solvers

Linear Algebra PACKage (LAPACK)

- Linear algebra stuff like solving $\mathbf{A}\vec{\mathbf{x}} = \vec{\mathbf{b}}$
- Standard but again bit cryptic (dgels,...) and 'very fortran'
- For sparse vectors/matrices, MKL has separate routines.



SciNet
compute • calcul
CANADA

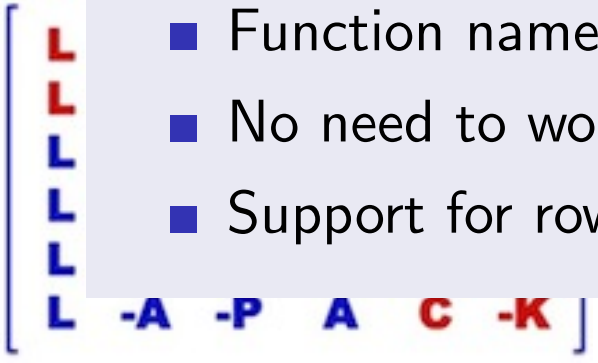
2. Linear Solvers

Linear Algebra PACKage (LAPACK)

- Linear algebra stuff like solving $\mathbf{A}\vec{\mathbf{x}} = \vec{\mathbf{b}}$
- Standard but again bit cryptic (dgels,...) and 'very fortran'
- For sparse vectors/matrices, MKL has separate routines.

LAPACKE

- Is a C interface to the Fortran LAPACK calls.
- Requires inclusion of header file **`mkl_lapacke.h`**
- Function names are **`LAPACKE_<lapackname>`**
- No need to worry about mixed language calls.
- Support for row-major matrices.



2. Linear Solvers

Scalable LAPACK 1 (ScaLAPACK1)

- Distributed memory version of LAPACK
  (Built on parallel fortran packages PBLAS and BLACS)
- No C interface: have to call the fortran functions:
  - defined in **`mkl_scalapack.h`**

- To build with distributed memory, use the mpi versions of the compilers (**`module load intelmpi`**), and compile and link with mpifort or mpicc using:

  `-mkl=cluster`

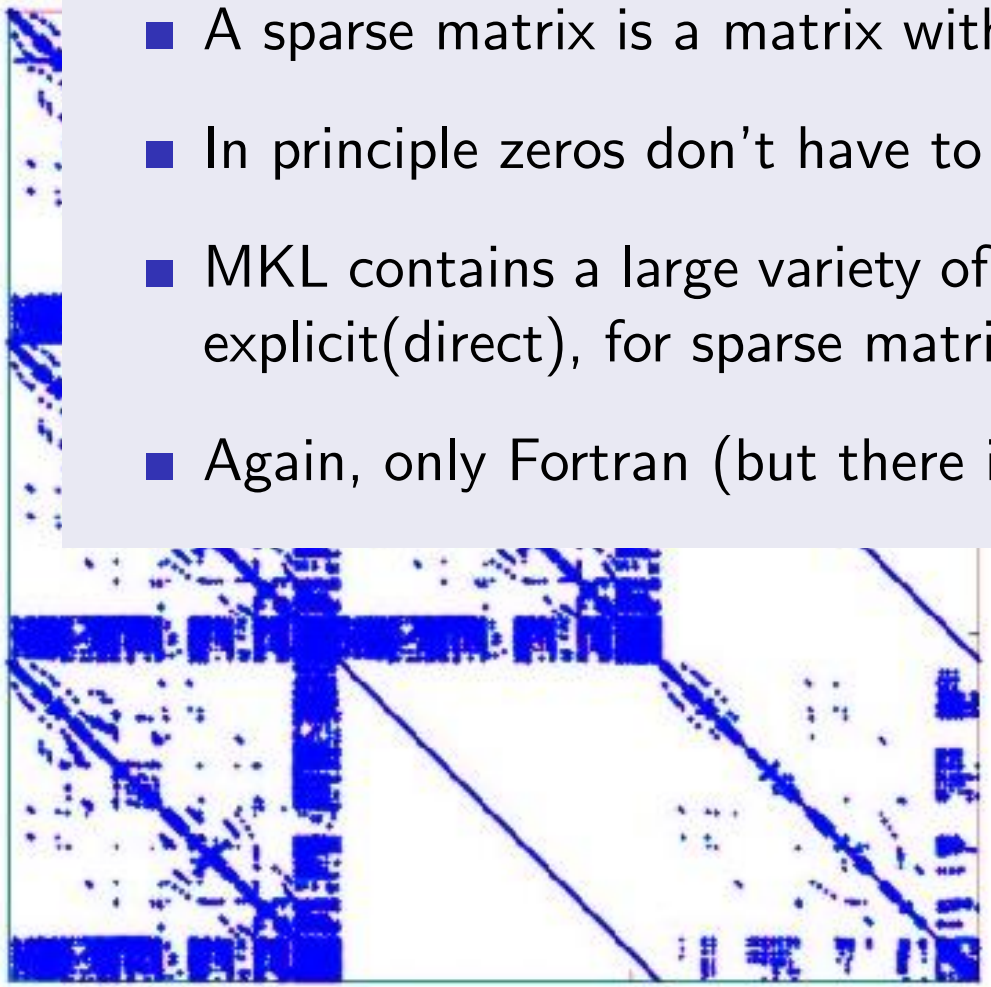
- Does not support ScaLAPACK2 (but the gpc module scalapack/2.0.1-intel-intelmpi does).

SciNet compute • calcul CANADA

2. Linear Solvers

Sparse Solver routines

- A sparse matrix is a matrix with a lot of zero elements
- In principle zeros don't have to be stored or computed with.
- MKL contains a large variety of solvers, implicit (iterative) and explicit(direct), for sparse matrices, as well as preconditioners.
- Again, only Fortran (but there is **`mkl_solver.h`**)



SciNet compute • calcul CANADA

3. Vector Mathematical & Statistical Functions

Vector Mathematical Library (VML)

- Most modern processors have a small degree of vector parallelism on each cpu (SIMD).
- E.g. 4 additions can be done at the same time.
- Hard to program for explicitly, but compilers can use them for simple loops.
- Beyond that, the VML offers functions for computing e.g. power, trigonometric, exponential, error functions, ...
- Act on **n** numbers at a time.
- C and Fortran

3. Vector Mathematical & Statistical Functions

Vector Statistical Library functions (VSL)

- Random numbers!
  - Various pseudo random number generators.
    Congruential generators, Mersenne Twister
  - Some are very suitable for parallelization.
  - Commonly used distributions:
    Bernoulli, Poisson, uniform, exponential, log-normal, beta, ...
- Convolutions and correlations as well.

3. Vector Mathematical & Statistical Functions

Vector nature of VSL

- Generate **n** random numbers at a time.
- Can lead to speed up over one-number-at-a-time methods.
- Uses VML for non-uniform distributions.

3. Vector Mathematical & Statistical Functions

Vector nature of VSL

- Generate **n** random numbers at a time.
- Can lead to speed up over one-number-at-a-time methods.
- Uses VML for non-uniform distributions.

How about parallelization?

- In parallel applications, one may need a sequence of random number generators for each thread or process.
- Should independently generate independent numbers

3. Vector Mathematical & Statistical Functions

Vector nature of VSL

- Generate **n** random numbers at a time.
- Can lead to speed up over one-number-at-a-time methods.
- Uses VML for non-uniform distributions.

How about parallelization?

- In parallel applications, one may need a sequence of random number generators for each thread or process.
- Should independently generate independent numbers

This can be accomplished by

- Using different parameters for same type of rng (MT)
- Skipping ahead or leapfrog (works for some of the VSL rngs)

VSL example

```
#include <stdio>
#include "mkl_vsl.h"
#define size 100000
#define repeat 1000
int main() {
  VSLStreamStatePtr str;
  vslNewStream(&str,VSL_BRNG_MT19937,777);
  double s = 0;
  for (int i=0; i<repeat; i++) {
    double r[size];
    vdRngGaussian(VSL_METHOD_DGAUSSIAN_ICDF,
                  str,size,r,5,2);
    for (int j=0; j<size; j++)
      s += r[j];
  }
  s /= size*(double)repeat;
  vslDeleteStream(&str);
  printf(Sample mean of normal distribution=%lf\n,s);
}
```

4. Fourier Transform functions

- Fast Fourier transforms for real and complex data
- Serial, threaded and distributed (cluster) versions.
- Workflow:
  1. Build a descriptor once
  2. Commit a descriptor (lets mkl try out some tweaks)
  3. Execute fft several times.
  4. Destroy descriptor.

4. Fourier Transform functions

- Fast Fourier transforms for real and complex data
- Serial, threaded and distributed (cluster) versions.
- Workflow:
  1. Build a descriptor once
  2. Commit a descriptor (lets mkl try out some tweaks)
  3. Execute fft several times.
  4. Destroy descriptor.

FFTW wrappers

- FFTW is a good free implementation of the FFT.
- MKL comes with wrappers that make MKL act like FFTW.
- Can be a bit tricky, especially if you don't have 'standard' usage. There are exception cases on what's supported for which you have to read the fine print of the MKL manual.

FFT example 1

```
#include <complex.h>
#include "mkl_dfti.h"
void mkl_transform(int N,
                   float complex *in,
                   float complex*out,
                   int s)
{
  DFTI_DESCRIPTOR_HANDLE handle;
  DftiCreateDescriptor(&handle,DFTI_SINGLE,DFTI_COMPLEX,1,N);
  DftiSetValue(handle,DFTI_PLACEMENT,DFTI_NOT_INPLACE);
  if (s > 0) {
    DftiCommitDescriptor(handle);
    DftiComputeForward(handle,in,out);
  } else {
    DftiSetValue(handle,DFTI_BACKWARD_SCALE,1.0f/N);
    DftiCommitDescriptor(handle);
    DftiComputeBackward(handle,in,out);
  }
  DftiFreeDescriptor(&handle);
}
```

FFT example 2 (fftw wrapper)

```
#include <complex.h>
#include "fftw3_mkl.h"
void fftw_transform(int N,
                    float complex* in,
                    float complex* out,
                    int s)
{
  int dir = isign>0?FFTW_FORWARD:FFTW_BACKWARD;
  fftw_plan p;
  p=fftw_plan_dft_1d(N,(fftw_complex*)in,
                     (fftw_complex*)out,dir,
                     FFTW_ESTIMATE);
  fftw_execute(p);
  if (isign<0)
    for (int i=0;i<N;i++)
      out[i] /= N;
  fftw_destroy_plan(p);
}
```

Links

Linear algebra:

BLAS: www.netlib.org/blas/faq.html
LAPACK: www.netlib.org/lapack
ScaLAPACK: www.netlib.org/scalapack

MKL:

software.intel.com/sites/products/documentation/hpc/mkl/userguides/mkl_userguide_lnx.pdf
software.intel.com/sites/products/documentation/hpc/mkl/vslnotes/index.htm
software.intel.com/en-us/articles/intel-mkl-link-line-advisor

Lecture slides

wiki.scinethpc.ca/wiki/images/a/ad/Numerics.pdf (rngs)
wiki.scinethpc.ca/wiki/images/0/0a/Linearalgebra.pdf
wiki.scinethpc.ca/wiki/images/8/8c/SCLecture8.pdf (fft)