Introduction to Numerical Libraries

Jerry Ebalunode
Center for Advanced Computation and Data Systems
(CACDS)
http://cacds.uh.edu

http://support.cacds.uh.edu

University of Houston
Houston, TX

@UHCACDS twitter

facebook.com/CACDSATUH

Overview

- Scientific Computing
- Why Numerical Libraries
- Random Number Generation (Intel MKL)
- Vector Arithmetic (Intel MKL)
- Fast Fourier Transforms (FFTW & MKL)
- Linear Algebra (BLAS and LAPACK)
- Open Lab and Homework

First Access Your Account

- Log into your accounts
  - Username or login = hpc_user**X**
  - Where x = sign in serial number 1 – 47
  - Password = **cacds2014**

  - Use your web browser
    - Firefox, Chromium or Google chrome
- Slides could be downloaded from URL below
  - http://129.7.249.171/workshops/intro2numlib.pdf

Getting Started

Use the terminal to download intro2numlib.zip file to your home directory

- Run the following commands

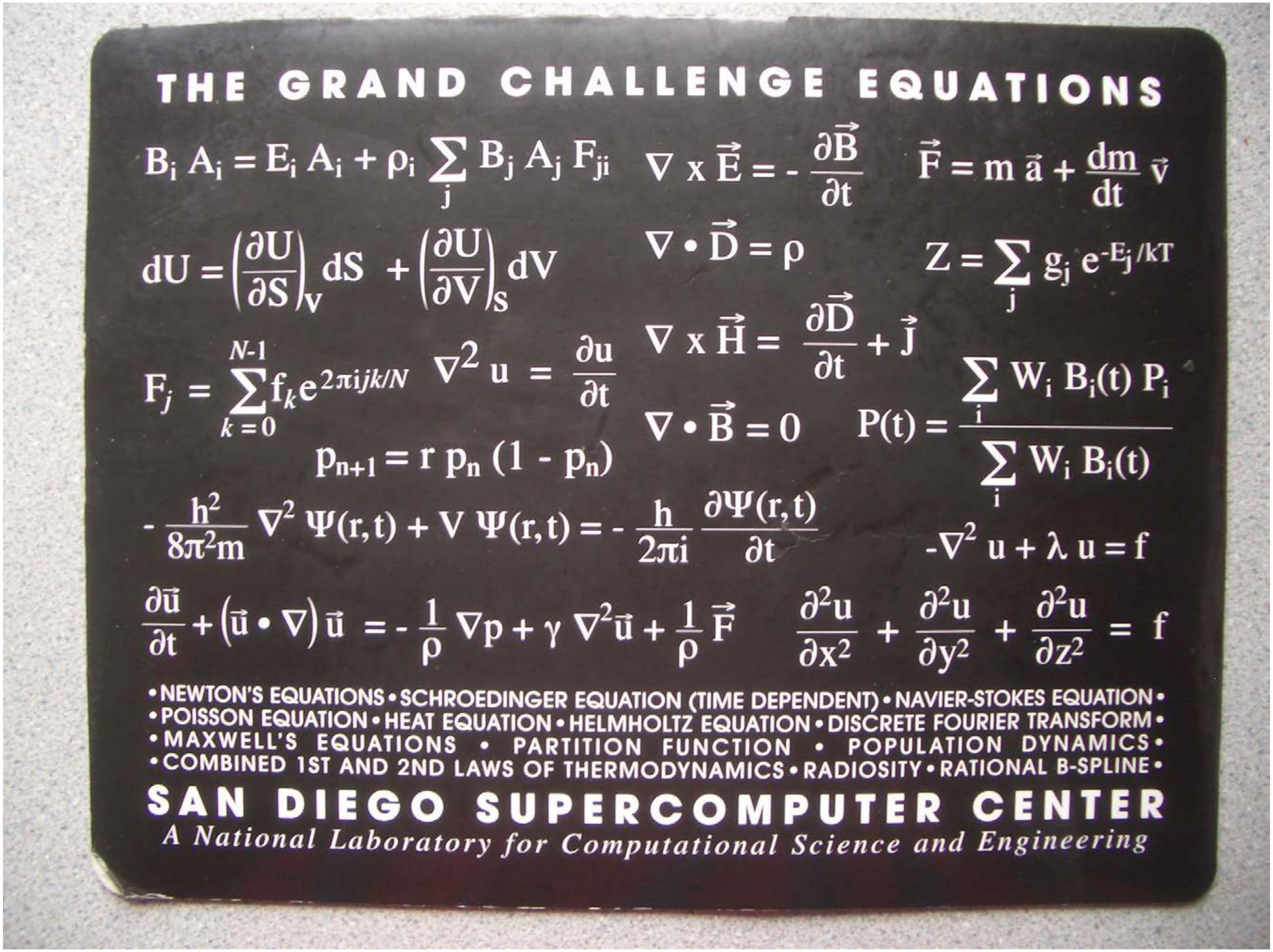
```
cd

wget   http://129.7.249.171/workshops/intro2numlib.zip

unzip    intro2numlib.zip


cd  intro2numlib
```

Now, you can begin working with tutorial files on your terminal



[Courtesy of San Diego Supercomputer Center]

Scientific Computing

- Why should we care about scientific computing?
  - Computational research has emerged to complement experimental methods in basic research, design, optimization, and discovery in all facets of engineering and science
  - In certain cases, computational simulations are the only possible approach to analyze a problem:
    - Experiments may be cost prohibitive (e.g. *flight testing a 1,000 fuselage/wing-body configurations for a modern fighter aircraft*)
    - Experiments may be impossible (e.g. *interaction effects between the International Space Station and Shuttle during docking*)
  - Simulation capabilities rely heavily on the underlying compute power (e.g. amount of memory, total compute processors, and processor performance)
    - Fostered the introduction and development of *super-computers* starting in the 1960's
    - Large-scale compute power is tracked around the world via the *Top500 List* (more on that later)

[Courtesy of San Diego Supercomputer Center]

Scientific Computing: a definition

- "The efficient computation of constructive methods in applied mathematics"
  - Applied math: getting results out of application areas
  - Numerical analysis: results need to be correctly and efficiently computable
  - Computing: the algorithms need to be implemented on modern hardware

[Courtesy of San Diego Supercomputer Center]

Case Study
Random Number Generation

- A random number generator (RNG) is a computational device designed to generate a sequence of numbers or symbols that lack any pattern, i.e. appear random.
- RNGs are widely used in Monte Carlo-method simulations, molecular dynamics simulations, cryptography etc.

Random Number Generation using GNU C library

```
#include <iostream>
#include <omp.h>
using namespace std;
int main()
{
  double t1,t0,elapsed;
  const size_t N = 1<<29L;
  //const size_t F = sizeof(float);
  float* A = new float [N];
  srand(0); // Initialize RNG
  t0=omp_get_wtime();
  for (int i = 0; i < N; i++)
  {
    A[i]=(float)rand() / (float)RAND_MAX;
  }
  t1=omp_get_wtime();
  elapsed=t1-t0;

  cout << "\nGenerated  "<< N<< "  random numbers in "<<elapsed << " seconds\n\nHere is a sample\n";

   for (int i = 0; i < 10; i++)
   {
     cout <<endl<<A[i];
   }
    delete  [] A;
 }
```

Random Number Generation

module add intel

icpc random_gen.cpp -openmp -o random_gen

./random_gen

Generated 536,870,912 random numbers in 5.49038 seconds

Here is a sample

0.242578

Numerical Libraries

- Software libraries used in application development for performing numerical calculations.
  - Robust and efficient algorithms
- Benefits
  - Helps scientists from reinventing the wheel
  - Scientist can focus more on the original problem
- Example
  - OpenSource
    - Blas, Lapack, GSL, GMP, FFTW, ACML, SuitSparse, Magma
  - Commercial
    - Intel Math Kernel Library, NAG, IMSL, ALGLIB

Intel Math Kernel Library

- MKL improves performance with math routines for software applications that solve large computational problems.
- MKL provides:
  - BLAS and LAPACK linear algebra routines
  - Fast Fourier transforms
  - Vectorized math functions
  - Random number generation functions
  - Tools for solving partial differential equations

Random Number Generation using Intel Math Kernel Library

A typical algorithm for MKL random number generators is as follows:

- Create and initialize stream/streams. Functions vslNewStream
- Call one or more RNGs.
- Process the output.
- Delete the stream/streams. Function vslDeleteStream.

- Reference:
  - https://software.intel.com/en-us/node/521842

Random Number Generation using Intel Math Kernel Library

```
#include <iostream>
#include <mkl_vsl.h>
#include <omp.h>
using namespace std;
int main()
{
  double t1,t0,elapsed;
  const size_t N = 1<<29L;
  //const size_t F = sizeof(float);
  float* A = new float [N];

  VSLStreamStatePtr rnStream;
  vslNewStream(&rnStream, VSL_BRNG_MT19937, 1 );     ; // Initialize RNG

  t0=omp_get_wtime();
  vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD, rnStream, N, A, 0.0f, 1.0f);
  t1=omp_get_wtime();
  elapsed=t1-t0;

 cout << "\nGenerated  "<< N<< "  random numbers in "<<elapsed << " seconds\n\nHere is a sample\n";
   for (int i = 0; i < 10; i++)
   {
     cout <<endl<<A[i];
   }
   delete  [] A;
   vslDeleteStream( &rnstream );

 }
```

Random Number Generation

icc -openmp -mkl=sequential random_gen_numlib.cpp -o random_gen_numlib

./random_gen_numlib

Generated 536,870,912 random numbers in **1.57736** seconds

~4X speedup

Here is a sample

0.134364

Vector Arithmetic

```
for (int i = 0; i < N; i++)
{
 C[i]= A[i] + B[i];
}
```

Vector Add

```
icpc vecadd.cpp -openmp -o vecadd
```

```
./vecadd
```

Vector Arithmetic with Intel MKL

```
vsAdd( N, A, B, C)
```

Vector Add

```
icpc    vecadd_numlib.cpp   -openmp    -o
vecadd_numlib      -mkl=sequential

./vecadd_numlib
```

Fast Fourier Transform

- FFT is an algorithm to compute Discrete Fourier Transforms (DFT)in a fast way
- FFT employed in simulation of periodic systems
  - periodic systems are very common in scientific computing
    - many body systems in empirical and *ab initio* molecular dynamics
- DFT is simply the computation of the coefficients ***ck***, the integrals, using the trapezoidal integration formula, that is, if *x0, x1, . . . , xN−1* are N complex numbers that represents *f(n/N) = xn*, then

$$\sum_{k=-\infty}^{\infty} c_k e^{2\pi k i \theta} \quad \Bigg| \quad c_k = \int_{\mathbb{T}} e^{-2\pi k i \theta} f(\theta)\, d\theta \quad \Bigg| \quad c_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{-ik\frac{2\pi n}{N}}$$

Fast Fourier Transform
1D DFT

```
int main(int argc, char* argv[])
{

  double t1,t0,elapsed;
  const size_t N = 1<<25L;
  fftw_complex *in, *out;
  fftw_plan p;
  int i;
  float *real = new float [N];
  float *imag = new float [N];
  VSLStreamStatePtr rnStream;
  vslNewStream(&rnStream, VSL_BRNG_MT19937, 1 );     ; // Initialize RNG
  in  = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*N);
  vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD, rnStream, N, real, 0.0f, 1.0f);
  vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD, rnStream, N, imag, 0.0f, 1.0f);
  for(i=0;i<N;i++)
  {
   in[i][0]=real[i];
   in[i][1]=imag[i];
  }
  out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*N);
  //! STEP ONE
  p = fftw_plan_dft_1d(N,in,out,FFTW_FORWARD,FFTW_ESTIMATE);
```

Fast Fourier Transform

```
  //! STEP TWO
  t0=omp_get_wtime();
  fftw_execute(p);
  t1=omp_get_wtime();
  elapsed=t1-t0;

  printf("1-D FFT is:\n");
  for(i=0;i<10;i++)
  {
   cout << out[i][0]<<"\t"<< out[i][1] <<endl;
  }
  //! STEP THREE
  fftw_destroy_plan(p);
  fftw_free(in);
  fftw_free(out);
  delete [] real;
  delete [] imag;
  vslDeleteStream(&rnStream);
  return 0;
}
```

FFT example

icpc fftw_example.cpp -openmp -o fftw_example -mkl=sequential

./fftw_example

BLAS
Basic Linear Algebra Subprograms

- The Basic Linear Algebra Subprograms (BLAS) define a set of fundamental operations on vectors and matrices which can be used to create optimized higher-level linear algebra functionality.
- Tuned Linear Algebra Programs
- **Level 1**
  - Vector operations, e.g. *y = \alpha x + y*
- **Level 2**
  - Matrix-vector operations, e.g. *y = \alpha A x + \beta y*
- **Level 3**
  - Matrix-matrix operations, e.g. *C = \alpha A B + C*

BLAS

The BLAS routines and functions are divided into the following groups according to the operations they perform:

- BLAS Level 1 Routines perform operations of both addition and reduction on vectors of data. Typical operations include scaling and dot products.
- BLAS Level 2 Routines perform matrix-vector operations, such as matrix-vector multiplication, rank-1 and rank-2 matrix updates, and solution of triangular systems.
- BLAS Level 3 Routines perform matrix-matrix operations, such as matrix-matrix multiplication, rank-k update, and solution of triangular systems.

BLAS
SAXPY

```
#include "mkl_cblas.h"

#include <omp.h>
using namespace std;

int main(int argc, char* argv[])
{

  double t1,t0,elapsed;
  float alpha=3.0f;
  MKL_INT i, N = 1<<25L;
  //! STEP ZERO
  float *x = new float [N];
  float *y = new float [N];
  VSLStreamStatePtr rnStream;
  vslNewStream(&rnStream, VSL_BRNG_MT19937, 1 );     ; // Initialize RNG

  vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD, rnStream, N, x, 0.0f, 1.0f);
  vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD, rnStream, N, y, 0.0f, 1.0f);

 //! STEP ONE
  t0=omp_get_wtime();
  cblas_saxpy(N, alpha, x, 1, y, 1);
  t1=omp_get_wtime();
```

BLAS CALL

BLAS
SAXPY

```
  t1=omp_get_wtime();
  elapsed=t1-t0;

  printf("SAXPY is:\n");
  for(i=0;i<10;i++)
  {
   cout << y[i] <<endl;
  }
  //! STEP TWO
  delete [] x;
  delete [] y;
  vslDeleteStream(&rnStream);
  return 0;
}
~
```

SAXPY example

icpc saxpy.cpp -openmp -o saxpy -mkl=sequential

./saxpy

LAPACK

- The original versions of LAPACK from which that part of Intel MKL was derived can be obtained from http://www.netlib.org/lapack/index.html. The authors of LAPACK are E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen.
- The LAPACK routines can be divided into the following groups according to the operations they perform:
  - Routines for solving systems of linear equations, factoring and inverting matrices, and estimating condition numbers (see LAPACK Routines: Linear Equations).
  - Routines for solving least squares problems, eigenvalue and singular value problems, and Sylvester's equations (see LAPACK Routines: Least Squares and Eigenvalue Problems).
  - Auxiliary and utility routines used to perform certain subtasks, common low-level computation or related tasks (see LAPACK auxiliary and utility routines).

LAPACK Routines

- The library includes LAPACK routines for both real and complex data. Routines are supported for systems of equations with the following types of matrices:
  - general
  - banded
  - symmetric or Hermitian positive-definite (full, packed, and rectangular full packed (RFP) storage)
  - symmetric or Hermitian positive-definite banded
  - symmetric or Hermitian indefinite (both full and packed storage)
  - symmetric or Hermitian indefinite banded
  - triangular (full, packed, and RFP storage)
  - triangular banded
  - tridiagonal
  - diagonally dominant tridiagonal.

Routine Naming Conventions

- To call one of the routines from a FORTRAN 77 program, you can use the LAPACK name.
  - LAPACK names have the structure ?yyzzz or ?yyzz, where the initial symbol ? indicates the data type:

s real, single precision

c complex, single precision

d real, double precision

z complex, double precision

Eigen value an Eigen vectors using LAPACK Drivers

- ssyev: computes eigenvalues and, optionally, the eigenvectors of a square real symmetric matrix *A*

- *interface*
- ssyev(*jobz, uplo, n, a, lda, w, work, lwork, info*)

Eigen value an Eigen vectors using LAPACK Drivers

```
ssyev( "Vectors", "Upper", &n, a, &lda, w,
&wkopt, &lwork, &info );
    lwork = (int)wkopt;
    work = (float*)malloc( lwork*sizeof(float) );
    /* Solve eigenproblem */
ssyev( "Vectors", "Upper", &n, a, &lda, w, work,
&lwork, &info );
```

SSYEV example

```
icpc  ssyev.cpp   -o  ssyev   -mkl=sequential
```

```
./ssyev
```

Parallel Numerical Library

- Three flavors
  - Multi-threaded support for multicore platforms
    - OpenMP for shared memory systems
  - Distributed memory support
    - MPI enabled libraries
      - SCALAPACK
- GPGPU i.e. support Accelerators and Co-Processors
  - Xeon Phi Intel MKL offload support
  - Magma library http://icl.cs.utk.edu/magma/
    - BLAS and LAPACK support on NVIDIA and Intel GPGPUs
    - open source
  - CuBLAS, CuFFT, CuSparse
    - Support NVIDIA GPUs

Parallel Numerical Library
Multi-core Platforms

Multi-threaded support for multicore platforms

- OpenMP for shared memory systems

- Example with ssyev

```
icpc  ssyev.cpp   -o  ssyev.parallel   -mkl=parallel

export OMP_NUM_THREADS=#CORES_ON_SERVER
./ssyev
```

HPC Support Page

- http://support.cacds.uh.edu/