Tools for Computational Physics
Week 2, Lecture 2
Mathematical Libraries

R. Rousseau
SISSA, 2-4 Via Beirut,
Trieste, Italy
Outline

• Introduction
• CPUs and Memory
• Math libs.
• Performance
• Linking Math libs
• Examples
Introduction

- The goal of this lecture is to introduce some basic understanding of how the CPU and memory work together to perform a calculation.

- Demonstrate possible bottle necks in calculations can occur and how this may be avoided by using mathematical libraries.

- Explain what these libraries contain and give a brief overview how they are incorporated by the user into their code.

- Illustrate their uses and provide general information on available software and how to choose what you need for a given project.
A CPU contains a chip where circuits are wired in to turn electrical signals into mathematical operations.

• Each CPU-type has a different architecture and a set of instructions on how to operate. (we will not discuss the +/- of a given type in any great detail).

• CPUs also require data which they keep in memory on the CPU (cache).

• Access to this memory is fast but this memory is expensive to manufacture and often there is more data than memory to store it.

• Thus the computer also has RAM memory and I/O devices where this data is stored and moved into the CPU (by the program) for computation.

Note that PC based machines all have small cache as compared with work station class machines.
memory hierarchy

• In modern computer system same data is stored in several storage devices during processing

• The storage devices can be described & ranked by their speed and “distance” from the CPU

• There is thus a hierarchy of memory objects

• Programming a machine with memory hierarchy requires optimization for that memory structure.
The Memory Hierarchy

CPU       Register      Cache                      RAM              VIRTUAL

CPU Speed

element

line

page

processor side  system side

Size
Processor-DRAM Gap (latency)

"Moore’s Law: 4X/3"

Introduction of RISC architecture

Processor-Memory Performance Gap:
(grows 50% / year)

DRAM 7%/yr.

μProc 60%/yr.

“Moore’s Law: 4X/3”
Memory Hierarchy

- Processor
  - Control
  - Registers
  - On-chip cache

- Second level cache (SRAM)
- Main memory (DRAM)
- Secondary storage (Disk)
- Tertiary storage (Disk/Tape)

Speed (ns): ~5 ~75 ~500 ~10 ms ~10 sec
Size (bytes): ~Kb ~Mb ~Gb ~Tb 100 Tb
Layout of a typical Computer: Pentium IV

- The most commonly used machines are PIVs (single or dual) which have high clock speeds for perform the FEC. 512Mb cache. Details: #more /proc/cpuinfo

- Multiprocessor PIVs can exist on the same board and share the same memory etc (SMP machine).

- The typical multiprocessor PIV machine has N CPUs on the same BUS linked via a memory controlled to the RAM and I/O devices.
Computational Bottle Necks: Athlon

- Memory access delayed by passing through northbridge
- I/O & memory compete for CPU’s FSB B/W
- B/W bottlenecks: link B/W < I/O device B/W
Compute Nodes

- Lanai 9.0 Myrinet (copper)
- 2 Broadcom Gbit ethernet.
- 2 Opteron 246 (2GHz/1MB cache)
- 4GB 400MHz Kingston memory
- Mainboard Celestica A220

![Memory access on Opteron and Xeon SMP nodes](image)
A newer paradigm in architecture is Opteron. Memory controlled on the CPU. However, RAM is ALWAYS slower than Cache. So architecture will not completely solve the problem. Note CPU freq. is lower in opteron than PIV (less power used and heat generated) but is just as fast or faster: clock speed is not everything.
Math Libraries

- Routines for common math factions such as vector and matrix operations, Fourier transform etc written in a specific way to take the most advantage of the architecture of the CPU.
- Compilers can optimize code only to a certain point (they are dumb) hence sophisticated algorithms and coding is required for the compiler to make a routine that is really efficient: naive coding won't work!
- AN ABSOLUTE NECESSITY on PC based machines due to small cache on CPU.
- Makes coding easier as intrinsic math functions can be used from canned routines.
Common Math Libraries

Groups of subroutines that are “standard” and can be modified by the machine vendor to run on their machine. This enables for codes to be more portable from one machine to the next and still be efficient.

**Linear Algebra (LA)**

**Basic Linear Algebra Subroutines (BLAS)**
- Level 1 (vector-vector operations)
- Level 2 (matrix-vector operations)
- Level 3 (matrix-matrix operations)
- Routines involving sparse vectors

**Linear Algebra PACKage (LAPACK)**
- Leverage BLAS to perform complex operations

Fast Fourier Transform (FFTW)
- Real, or Complex 1D, 2D, 3D.
Common Libraries

Standard BLAS and LAPACK (not machine specific)


AMD Math Core Library (ACML): BLAS LAPACK FFTW routines modified for best performance on x86 (athlon, opteron) based machines.

Automatically Tuned Linear Algebra Software (ATLAS): some BLAS and LAPACK routines that can be compiled on PC based machines to obtain better maximum performance by tuning machine specific parameters.

All can be downloaded free from the web.
Library Performance: Variance with Machine

DGEMM using ACML 2.5.0 on different architectures

- Xeon 3.4 EM64T
- Opteron 250 Sun v20z
- Opteron 246 (celestica)

Mflop/second vs. matrix size (N x N)
How to include these libraries in your code.

Within your code you simply need to call the BLAS/LAPACK routines
As if they are subroutines you would normally write.
Note check the BLAS/LAPACK manuals to know the name of routine
And what variables need to be passed to them and in what order.
NB DGEMM double generic matric-matrix multiplication
Ubiquitous Nomenclature

As long as you stick to using the libraries in a standard way diverse software can all use the SAME crucial routines:

```
SUBROUTINE STATE_ORTHO(C1,CB,CS,UMAT,IROOT,NSTATE,MSUB,SCR,LSCR)
  !-------------------------------------------
  IMPLICIT NONE
  INCLUDE 'system.h'
  ! Arguments
  INTEGER    IROOT,NSTATE,MSUB,LSCR
  COMPLEX*16  C1(NGW,NSTATE,*) ,CB(NGW,*) ,CS(NGW,*)
  REAL*8     UMAT(NSTATE,MSUB,*) ,SCR(*)
  ! Variables
  INTEGER    IR
  REAL*8     SO,DOTPS
  !-------------------------------------------
  DO IR=1,IROOT-1
    CALL DGERM('T', 'N', MSUB, MSUB, NSTATE, 1.0D0, UMAT(1,1,IR), NSTATE,
               & UMAT(1,1,IROOT), NSTATE, 0.0D0, SCR, MSUB)
    CALL DGERM('N', 'N', 2*NGW, MSUB, MSUB, 1.0D0, C1(1,1,IR), 2*NGW,
               & SCR, MSUB, 0.0D0, CS, 2*NGW)
    SO=DOTPS(NGW, MSUB, CS, CB)
    CALL GLOSUM(1,SO)
    CALL DAXPY(2*NGW*MSUB,-SO,CS,1,CB,1)
  ENDDO
  !-------------------------------------------
  RETURN
END
```
How to include these libraries in your code.

Normally, the math library exists on the machine as a precompiled library object file.
How it is compiled determines how you will link into it to Your code.
The location of the library, how it is compiled (weather or not you need to tell the compiler where to find it) etc is EXTREMELY dependent on the System administrators.
How to include these libraries in your code: linking

```bash
# options for opteron: by rr

# compiler
F77=ifort
F90=$<F77>
FLAGS90=$<FLAGS>
FLAGS=-pc64 -tpp7 -03 -w95 -FI -unroll
#FLAGS = -pc64 -tpp7 -03 -unroll

#pgi
#F77=pgf90
#FLAGS=-c -Mfixed -tp k8-64 -fast -Mvect
#LFLAGS=-Mfixed -tp k8-64 -Bstatic -L.
#LINK= -L. -latlas_amd64
#LINK= -L/opt/acml2.0/pgi64/lib/. -lacml -lg2c

#Math libs static linking
LINK= -static-libcxa ./libatlas_amd64.a

# amcl
#LINK= -static-libcxa /opt/acml2.0/gnu64/lib/libamcl.a -lg2c

#MKL
#LINK= mpi/mpif.o randomP.o -L /opt/intel/mkl61/lib/32/ \ 
  -lmkl_lapack -lmkl_p4 -lguide -lpthread
```
Example: CPMD w/wo atlas

As a simple example we will consider the cpmd 3.7 code compiled with pgf on a 2.0GHz single PIV:
A With standard LAPACK/BLAS
B,With a BLAS/LAPACK/ATLAS combo library optimized fro a PIV.

```bash
# SHELL = /bin/sh
#
# Default Configuration for PC-PGI
SRC = .
DEST = .
BIN = .
#QMMM_FLAGS = -D__QMECHCOUPL
#QMMM_LIBS = -L. -1mm
FFLAGS = -Mr8 -pc64 -Msingnnextend
#LFLAGS = -L. -latlas_p4_2 $(QMMM_LIBS)
LFLAGS = \n
    -llapack -lblas -lpge -lpgeftnrtl
CFLAGS =
CPP = /lib/cpp -P -C -traditional
CPPFLAGS = -D__Linux -D__PGI -DLAPACK -DFFT_DEFAULT
NOOPT_FLAG =
CC = gcc -02 -Wall
FC = pgf77 -c -fast -tp piv
LD = pgf77 -fast -tp piv
AR =
```

```bash
#----------------------------------------------
```
**Example: CPMD w/wo atlas**

CPU time/step of SCF Wavefuncntion cycle:

<table>
<thead>
<tr>
<th>NFI</th>
<th>GEMAX</th>
<th>CNORM</th>
<th>ETOT</th>
<th>DETOT</th>
<th>TCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.507E+00</td>
<td>3.242E-02</td>
<td>-1945.730309</td>
<td>0.000E+00</td>
<td>1052.29</td>
</tr>
<tr>
<td>2</td>
<td>1.607E+00</td>
<td>2.549E-02</td>
<td>-1957.729373</td>
<td>-1.200E+01</td>
<td>1051.90</td>
</tr>
<tr>
<td>3</td>
<td>1.765E+00</td>
<td>1.739E-02</td>
<td>-1973.700217</td>
<td>-1.597E+01</td>
<td>1053.51</td>
</tr>
<tr>
<td>4</td>
<td>1.933E+00</td>
<td>1.721E-02</td>
<td>-1988.400667</td>
<td>-1.470E+01</td>
<td>1051.56</td>
</tr>
<tr>
<td>5</td>
<td>2.083E+00</td>
<td>2.347E-02</td>
<td>-2004.029210</td>
<td>-1.563E+01</td>
<td>1049.70</td>
</tr>
</tbody>
</table>

**Atlas**

<table>
<thead>
<tr>
<th>NFI</th>
<th>GEMAX</th>
<th>CNORM</th>
<th>ETOT</th>
<th>DETOT</th>
<th>TCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.507E+00</td>
<td>3.242E-02</td>
<td>-1945.730309</td>
<td>0.000E+00</td>
<td>355.12</td>
</tr>
<tr>
<td>2</td>
<td>1.607E+00</td>
<td>2.549E-02</td>
<td>-1957.729373</td>
<td>-1.200E+01</td>
<td>353.56</td>
</tr>
<tr>
<td>3</td>
<td>1.765E+00</td>
<td>1.739E-02</td>
<td>-1973.700217</td>
<td>-1.597E+01</td>
<td>352.52</td>
</tr>
<tr>
<td>4</td>
<td>1.933E+00</td>
<td>1.721E-02</td>
<td>-1988.400667</td>
<td>-1.470E+01</td>
<td>354.49</td>
</tr>
<tr>
<td>5</td>
<td>2.083E+00</td>
<td>2.347E-02</td>
<td>-2004.029210</td>
<td>-1.563E+01</td>
<td>353.35</td>
</tr>
</tbody>
</table>

**Atlas + new ELPO**

<table>
<thead>
<tr>
<th>NFI</th>
<th>GEMAX</th>
<th>CNORM</th>
<th>ETOT</th>
<th>DETOT</th>
<th>TCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.747E+00</td>
<td>3.427E-02</td>
<td>-1971.981392</td>
<td>0.000E+00</td>
<td>204.24</td>
</tr>
<tr>
<td>2</td>
<td>1.781E+00</td>
<td>2.642E-02</td>
<td>-1984.557400</td>
<td>-1.258E+01</td>
<td>204.16</td>
</tr>
<tr>
<td>3</td>
<td>1.817E+00</td>
<td>1.864E-02</td>
<td>-1997.649315</td>
<td>-1.309E+01</td>
<td>204.32</td>
</tr>
<tr>
<td>4</td>
<td>1.844E+00</td>
<td>1.545E-02</td>
<td>-2008.287456</td>
<td>-1.064E+01</td>
<td>204.56</td>
</tr>
<tr>
<td>5</td>
<td>1.860E+00</td>
<td>1.553E-02</td>
<td>-2018.243548</td>
<td>-9.956E+00</td>
<td>206.69</td>
</tr>
</tbody>
</table>

CPU time (s)
Conclusions

The best performance from a computer can be obtained by including “canned” software from a library.

Avoids memory bottlenecks and makes coding easier.

Can lead to many times the speed up of software especially on small cache CPU machines.

For legacy codes it is a huge job to BLASify them and if the code is I/O it may not be worth the effort. (always ask yourself “is it worth it?”)

Another type of library is required for codes to run on more than 1 CPU. At a time the common paradigm is Message Passing interface (MPI).