The Abdus Salam
International Centre for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

1967-10

Advanced School in High Performance and GRID Computing

3 - 14 November 2008

Mathematical Libraries.
Part I

KOHLMEYER Axel
University of Pennsylvania
Department of Chemistry
231 South 34th Street
PA 19104 Philadelphia
U.S.A.

Mathematical Libraries (Part 1)

ICTP Advanced School in High Performance and GRID Computing

Axel Kohlmeyer
Center for Molecular Modeling
ICTP, Trieste – Italy, 05 November 2008

The Abdus Salam
International Centre for Theoretical Physics

Overview

- Opportunities to improve application performance on modern CPUs
- Why use performance libraries?
- Using / linking libraries
- Example:
  Using DGEMM in BLAS

"There are better ways to log off."

Using Cache Efficiently

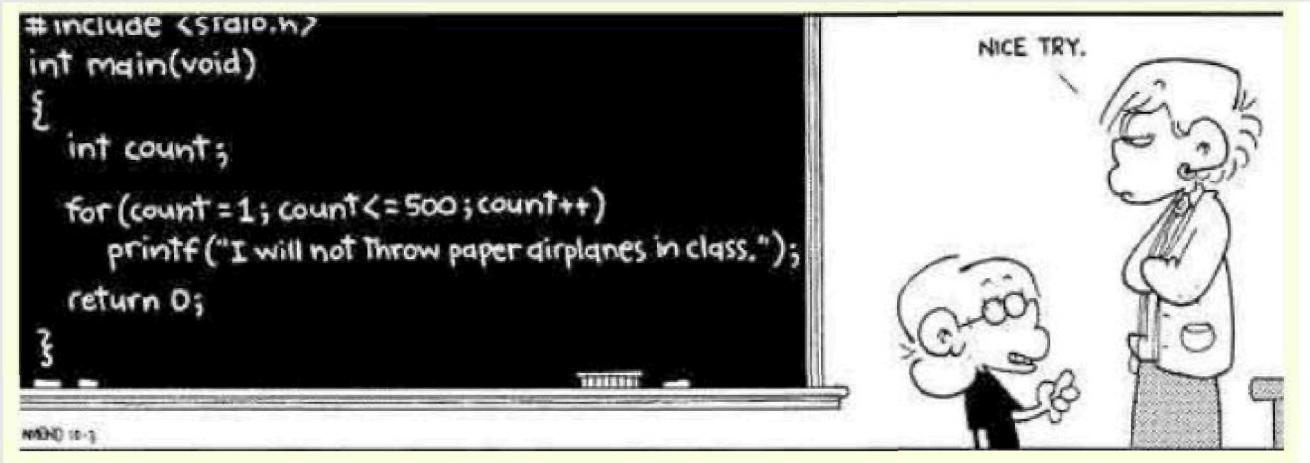
- Cache is a fast but small memory area
- Located on the CPU or close to it
- Compensate for discrepancy between CPU speed (fast) and Memory speed (slow)
- Typically transparently, and kept coherent in multi-core, multi-CPU environment.
- The key for good performance is to write code that maximized the effect of cache memory
- The optimal code structure (blocking) depend on CPU speed, model, and architecture

Using Special Instructions

- Modern CPUs contain many special purpose instructions: MMX,SSE,3d-now,Altivec
- Many allow to operate on multiple data elements in parallel: SIMD, vector instructions
- Programming in assembly required to use them efficiently
- In general not portable between different CPU architectures and models
- Significant speedups for Linear Algebra operations, signal processing.

Using Multi-core / Multi-CPU

- Modern CPUs contain multiple CPU cores
- Need parallel program (MPI, OpenMP, ...) to exploit the additional capability
- Parallelism in code need rewrites/restructuring (MPI), or instrumentation (OpenMP)



What are performance libraries?

- Routines for common (math) functions such as vector and matrix operations, fast Fourier transform etc. written in a specific way to take advantage of capabilities of the CPU.
- Each CPU type normally has its own version of the library specifically written or compiled to maximally exploit that architecture
- Make coding easier. Complicated math operations can be used from existing routines
- Increase portability of code as standard (and well optimized) libraries exist for ALL computing platforms.

Why use performance libraries?

- Compilers can optimize code only to a certain point (they are dumb). Effective programming needs deep knowledge of the platform
- Performance libraries are designed to use the CPU in the most **efficient** way, which is not necessarily the most straightforward way.
- It is normally best to use the libraries supplied by or recommended by the CPU vendor
- On modern hardware they are hugely important, as they most efficiently exploit caches, special instructions and parallelism

Standardization

- Subroutines have a standardized layout
- BLAS is documented in the source code
- Man pages exist
- Vendor supplied docs
- Different BLAS implementations have the same calling sequence

```
      SUBROUTINE DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
     $                   BETA, C, LDC )
*     .. SCALAR ARGUMENTS ..
      CHARACTER*1        TRANSA, TRANSB
      INTEGER            M, N, K, LDA, LDB, LDC
      DOUBLE PRECISION   ALPHA, BETA
*     .. ARRAY ARGUMENTS ..
      DOUBLE PRECISION   A( LDA, * ), B( LDB, * ), C( LDC, * )
*     ..
*
*  PURPOSE
*  =======
*
*  DGEMM  PERFORMS ONE OF THE MATRIX-MATRIX OPERATIONS
*
*     C := ALPHA*OP( A )*OP( B ) + BETA*C,
*
*  WHERE  OP( X ) IS ONE OF
*
*     OP( X ) = X   OR   OP( X ) = X',
*
*  ALPHA AND BETA ARE SCALARS, AND A, B AND C ARE MATRICES, WITH OP( A )
*  AN M BY K MATRIX,  OP( B )  A  K BY N MATRIX AND  C AN M BY N MATRIX.
*
*  PARAMETERS
*  ==========
*
*  TRANSA - CHARACTER*1.
*           ON ENTRY, TRANSA SPECIFIES THE FORM OF OP( A ) TO BE USED IN
*           THE MATRIX MULTIPLICATION AS FOLLOWS:
*
*              TRANSA = 'N' OR 'N',  OP( A ) = A.
*
*              TRANSA = 'T' OR 'T',  OP( A ) = A'.
*
*              TRANSA = 'C' OR 'C',  OP( A ) = A'.
*
*           UNCHANGED ON EXIT.
*
*  TRANSB - CHARACTER*1.
*           ON ENTRY, TRANSB SPECIFIES THE FORM OF OP( B ) TO BE USED IN
*           THE MATRIX MULTIPLICATION AS FOLLOWS:
*
*              TRANSB = 'N' OR 'N',  OP( B ) = B.
*
*              TRANSB = 'T' OR 'T',  OP( B ) = B'.
*
```

How to use libraries in code

```
[rroussea@samson qmc_code]$ more alldet.f90
        subroutine alldet(iopt,indt,nelorb,nelup,neldo,ainv,winv
     1,ainvupb,derl)
        implicit none
        integer iopt,nelorb,nelup,neldo,nelt,i,j,nelorb5
     1,indt
        real*8 ainv(nelup,*),ainvupb(nelup,*)
     1  ,derl(nelorb,*),winv(nelorb,0:indt+4,*)

         nelorb5=(indt+5)*nelorb



        call dgemm('N','T',nelup,nelorb,nelup,1.d0,ainv,nelup
     1,winv(1,0,1),nelorb5,0.d0,ainvupb,nelup)

        call dgemm('N','N',nelorb,nelorb,neldo,1.d0,winv(1,0,nelup+1)
     1,nelorb5,ainvupb,nelup,0.d0,derl,nelorb)
```

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 matrix-matrix multiplication

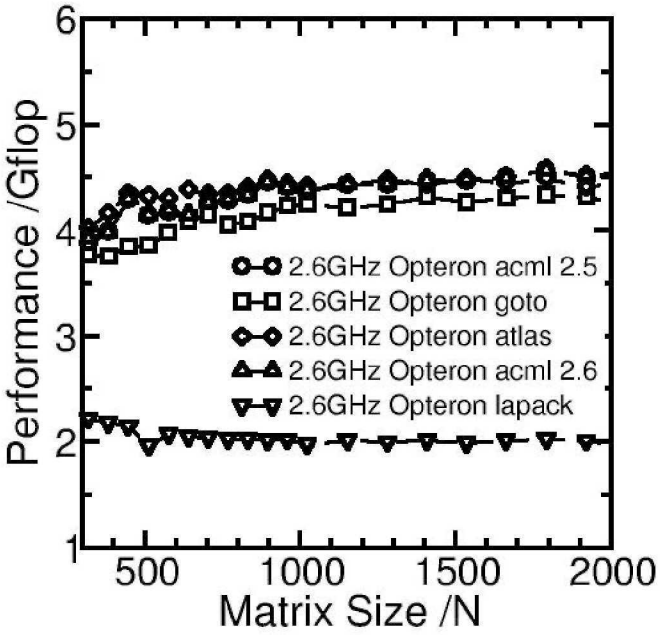
BLAS/LAPACK Implementations

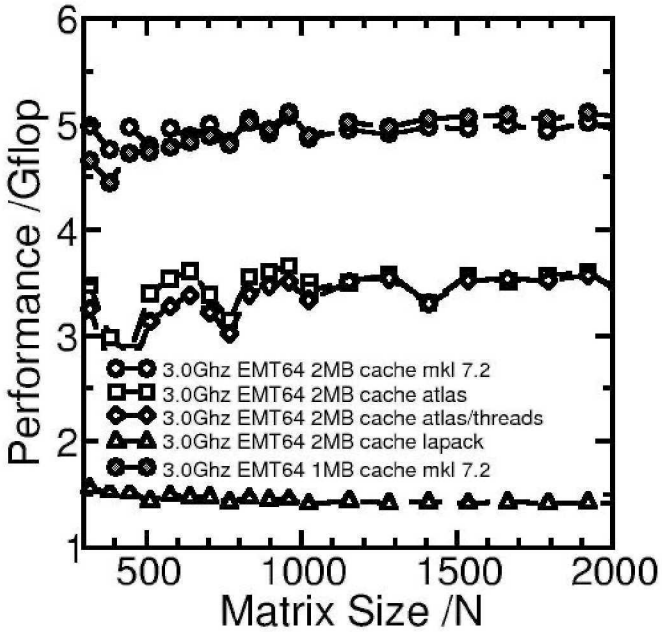
- BLAS and LAPACK reference (portable Fortran77)
- Intel Math Kernel Library (MKL): BLAS/LAPACK (+FFT) routines modified for best performance on IA32/x86, x86_64/AMD64/EM64t, and IA64 based machines.
- AMD Math Core Library (ACML): BLAS LAPACK (+FFT) routines modified for best performance on AMD x86 and x86_64 based machines
- Automatically Tuned Linear Algebra Software (ATLAS): BLAS and LAPACK routines that use empirical tests to tune machine specific parameters.
- GOTO BLAS, a BLAS implementation with special optimization for modern (x86/x86_64) CPU architectures

Linking Your Code to Libraries

- Linker flags: -L/some/other/dir -lm
  -> search for libm.so/libm.a also in /some/dir
- Order matters. Ex.: LAPACK uses BLAS
  => -L/usr/local/lib -llapack -lblas
- ATLAS is written C with f77 wrappers:
  => -L/opt/atlas/lib -lf77blas -latlas
- MKL uses “collections”.
  Using -lmkl links: libmkl_intel_lp64.so,
  libmkl_intel_thread.so libmkl_core.so
  => check with “ldd”
- Check LD_LIBRARY_PATH with shared libs

Library Performance Example





- The absolute and relative performance of a library is very platform dependent
- Vendor optimized libraries are most of the time faster.

Comments on LAPACK and BLAS

- BLAS Levels: 1) vector, 2) vector/matrix, 3) matrix
- BLAS Level 1, has N memory operations for N CPU operations. Whereas Level 3 has $2N^2$ memory/N^4 CPU => more efficient. Thus combining operations over vectors into matrix-matrix operations is useful
- Reference LAPACK/BLAS are better than “coding by hand”, BUT are less efficient than the vendor supplied versions of the same library or ATLAS.
- LAPACK relies heavily on BLAS so the key to good performance is a fast BLAS.

Beware! Multi-threaded Libraries

- FFTW: Thread parallelization through additional library, needs special management
- ATLAS: fixed number of threads compiled in
- MKL: Threading with OpenMP
  - Uses by default **all** available cores
    => conflicts with MPI, NUMA-SMP workstations
  - Modular, can be tuned to compiler, linker, threads
  - Default linking sequence for ifort with Intel threads
    => add -openmp to linker/compiler command
  - Serial: -lmkl_intel_lp64 -lmkl_sequential -lmkl_core
  - GNU: -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core
  - See documentation for details (always RTFM!).