Scientific Libraries Usage

Dr. Dimitris Dellis

GRNET

- What is a Library ?
  - A collection of routines that perform certain tasks

- What is a Library ?
  - A collection of routines that perform certain tasks
- Motivation to use libraries ?
  - Already available. No need to reinvent the wheel.
  - Usually ported to various architectures
  - Usually well tested for correctness
  - Well optimized
  - Fewer bugs
  - More portable code : You can use GPUs for example if the corresponding library is ported on GPUs.
  - Usable from various languages (C, Fortran, Python...)

Types of libraries

- I/O Libraries
  - MPI-I/O, HDF5, NetCDF
- Numerical Libraries
  - Dense and Sparse Linear Algebra
  - Fourier Transforms
  - Differential equations
- Domain Specific Libraries
  - openMM for MD on GPUs
  - libint for compute integrals in quantum mechanics
  - Packages functionality libraries. It is common in recent years, multifunctional packages to provide a library with which you can use parts of the binary/ies functionality.
  - Examples
    - **Tinker** Classical Molecular Dynamics package provides a library that is used by ab-initio packages to perform ab-initio MD (**gamess-US, cpmd**)
    - **mopac, Gaussian, Gamess-UK, Orca** abinitio packages libraries are used by classic MD packages (gromacs) to perform part of interaction calculations at ab-initio level.
    - Many packages libraries can be used for example from python to create post-processing tools, graphical interfaces etc.

I/O Libraries

Why they are useful ?

- HPC data usually are large and complex.
- Needed for Parallel I/O
- Need for portable data files between architectures
- Solutions : HDF5, NetCDF. Both are build on top of MPI-IO.
    - HDF5 is a data model and file format for storing multidimensional data.
    - HDF5 Files are portable between architectures (essentially portable between little and big endian architectures.
    - NetCDF and parallel NetCDF are usually built on top of HDF5 (and MPI-IO).
    - NetCDF is extensively used in geosciences (need to reuse large datasets on various architectures)

Numerical Libraries

- Linear Algebra
- Fourier Transforms

Numerical Libraries : Linear Algebra

BLAS/LAPACK is a standard.

Many variants use a common API

Other non-standard libraries :

- PLASMA : Parallel Linear Algebra for Scalable Multi-core Architectures http://icl.cs.utk.edu/plasma/
- MAGMA : Matrix Algebra on GPU and Multicore Architectures http://icl.cs.utk.edu/magma/
- Many more

Numerical Libraries : Linear Algebra
BLAS/Lapack Hierarchy

- Driver Routines : Solve certain problems using lower level routines, like Matrix Decomposition, Linear Equations Systems, Eigenvalue equations for dense, triangular, real and complex matrices.
- Level 3 : Matrix-Matrix operations
- Level 2 : Matrix-Vector operations
- Level 1 : vector-vector operations

Numerical Libraries : Linear Algebra
Matrix-Matrix Multiplication

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1K} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2K} \\ .. & .. & \cdots & .... \\ \alpha_{M1} & \alpha_{M2} & \cdots & \alpha_{MK} \end{bmatrix} x \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1N} \\ b_{21} & b_{22} & \cdots & b_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ b_{K1} & b_{K2} & \cdots & b_{KN} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1N} \\ c_{21} & c_{22} & \cdots & c_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ c_{M1} & c_{M2} & \cdots & c_{MN} \end{bmatrix}$$

Requires : K multiplications $+$ K additions for each element of B.

Total $2N \cdot M \cdot K$ floating point operations.

For M=N=K=1000 : $2 \cdot 10^9$ operations.

If a machine is able to perform the multiplication in 1 sec then its performance is 2 GFLOPS.

In next sections, we'll use the GFlops rate instead of time for comparisons.

Numerical Libraries : Linear Algebra
Matrix-Matrix Multiplication
We'll see three variants

- Simple Code for NxN matrices multiplication in Fortran

```
do i = 1, N
    do j = 1, N
         c(i,j)= 0.00000000000000000
         do ij = 1, N
           c(i,j)=c(i,j)+a(i,ij)*b(ij,j)
         enddo
    enddo
 enddo
```

- Using Fortran MatMul function

```
c=matmul(a,b)
```

- Using Lapack DGEMM function (see Lapack reference for meaning of parameters)

```
call dgemm('N','N',N,N,N,1.0,a,lda,b,ldb,0.0,c,ldc)
```

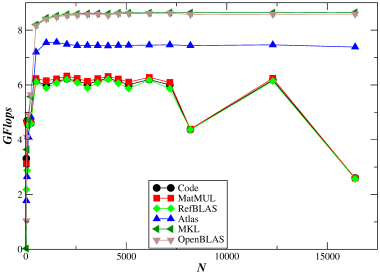
- Why to use the DGEMM call with all these parameters ? matmul is short, simple code is very clear.

Numerical Libraries : Linear Algebra

Matrix-Matrix Multiplication

Single core Matrix-Matrix multiplication results on Euclid.

Note : In compilations the maximum compiler optimization was used to compile the code.



Reasons to use Atlas, OpenBLAS (ex-GotoBLAS2) or MKL.

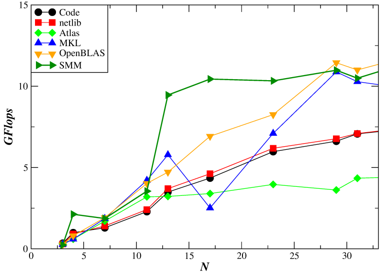
- Relative Performance
- Stable performance in almost all sizes

Numerical Libraries : Linear Algebra :
Matrix-Matrix Multiplication : What happens at small sizes ?
Intro of libSMM.

- In (mainly) Quantum Mechanics codes it is common to perform matrix-matrix multiplication of small matrices with unusual sizes like 5x13.
- LibSMM (Small Matrices Multiplication) : Libsmm use an autotuning approach to produce specialised DGEMM routines for a specified set of small matrix sizes.
- It is the prefered by CP2K.
- Autotuning takes long time, 1-3 days on a desktop. It compiles $24^3 + 14^3$ source files, and runs each produced executable for ~ 1 GFlop.
- A wrapper routine is also included such that SMM can be called for any sizes of M,N,K and if these are not supported directly in the library, external BLAS dgemm is called.
- Note that when one compiles autotuned software, CPU throttling (frequency scaling) SHOULD be off in order to get reliable tuning.
- Recent versions of ATLAS refuse to build when it is ON
- On Euclid, CPU throttling is ON.

Numerical Libraries : Linear Algebra : Matrix-Matrix Multiplication : Small Sizes

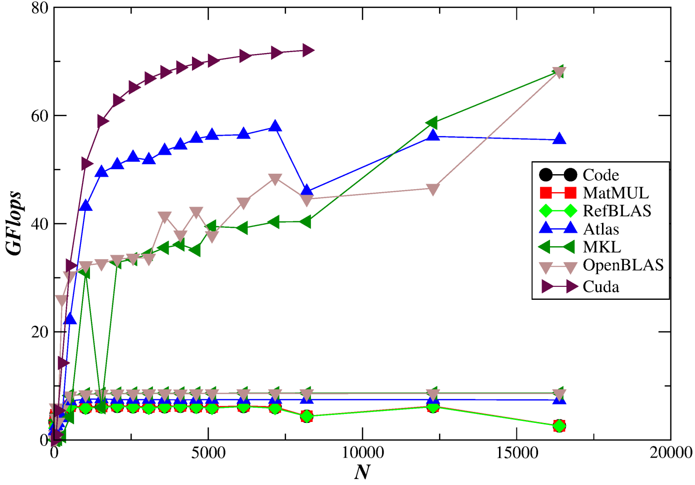
- What is the performance of BLAS libraries at small sizes
- Results on a SandyBridge



- libSMM is faster than all BLAS variants for sizes in range ∼ 12-25. At unusuall sizes it may be much faster (∼ 15x, depending on architecture and throttling during compilation).

Numerical Libraries : Linear Algebra
Matrix-Matrix Multiplication

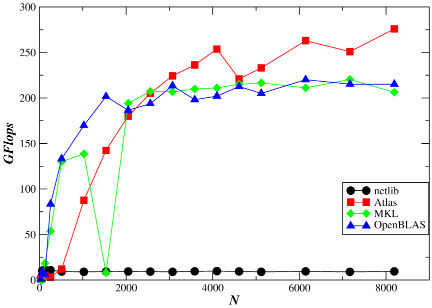
- SMP (8 cores)
  - All SMP versions use OpenMP except Atlas that may use OpenMP or pthreads.
  - Default Atlas build use pthreads.
- Matrix-Matrix multiplication results on Euclid.
- Graph also includes
  - Serial version performance
  - Single core/single Cuda GPU performance.
  - Cuda matrix size is limited by GPU device memory.


GFlops
N
0
20
40
60
80
0
5000
10000
15000
20000
Code
MatMUL
RefBLAS
Atlas
MKL
OpenBLAS
Cuda

Numerical Libraries : Linear Algebra

Matrix-Matrix Multiplication

SMP (24 cores) Matrix-Matrix multiplication results on a 24-cores/node Sandybridge.



Some notes for CUDA or any other interface

- Usually, functions for accelerators have different name. For example, cublas_dgemm instead of dgemm.
- You can avoid to substitute all dgemm with cublas_dgemm by creation of stubs routines (not only for cuda, lammps people should be familiar with mpi_stubs).
- For example : In your code you have a library call like : call dgemm(listofargs)
- Instead of substitution of all dgemm, you may create a pseudo-library that does name translation : i.e. (in Fortran terms)

```
subroutine dgemm(listofargs)
definitions of listofargs
call cublas_dgemm(listofargs)
return
end
```

- Something like

```
#define F(x,y) (x*x*y)
```

  in C.

Bigger Matrices.

Memory is a bottleneck (Already appeared for Cuda). What else ?

- Split on multiple nodes. Use Blacs/Scalapack. We'll use it later.

Numerical Libraries : BLAS/LAPACK

Linear Systems of Equations

$$Ax = b$$

- Find array x, given the arrays A and b, that satisfies the above equation. Note that in the notation a number of linear equations systems (M) is concurrently solved. When M=1, a single system of equations is solved.

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1N} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2N} \\ .. & .. & \cdots & .... \\ \alpha_{M1} & \alpha_{M2} & \cdots & \alpha_{MN} \end{bmatrix} x \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1M} \\ x_{21} & x_{22} & \cdots & x_{2M} \\ \cdots & \cdots & \cdots & \cdots \\ x_{N1} & x_{N2} & \cdots & x_{NM} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1M} \\ b_{21} & b_{22} & \cdots & b_{2M} \\ \cdots & \cdots & \cdots & \cdots \\ b_{N1} & b_{N2} & \cdots & b_{NM} \end{bmatrix}$$

Numerical Libraries : BLAS/LAPACK

Linear Systems of Equations

We'll explore two variants Numerical Recipes (NR) LU decomposition/backsubstitution, LAPACK (and variants Atlas, OpenBLAS, MKL) DGESV driver routine.

- NR LU decomposition :

```
call ludcmp(a,NMAX,NMAX,indx,d)
call lubksb(a,NMAX,NMAX,indx,b)
```

- Use of DGESV

```
call dgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
```

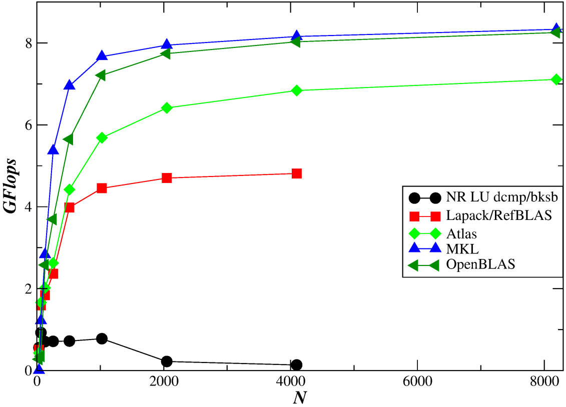
Numerical Libraries : BLAS/LAPACK

Linear Systems of Equations

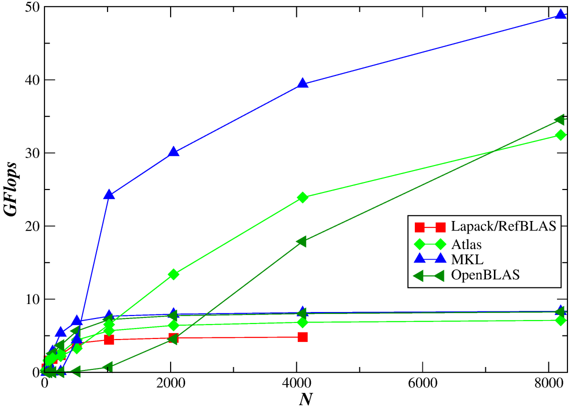
Performance Units

- DGESV GFLOPS$=10^{-9}\left(\frac{\frac{2}{3}N^3+2N^2}{Time[s]}\right)$
- Let's see some Performance Results. NR LU, Netlib Lapack, Atlas, MKL, OpenBLAS results.

Numerical Libraries : BLAS/LAPACK
Linear Systems of Equations
Single Core Performance Results



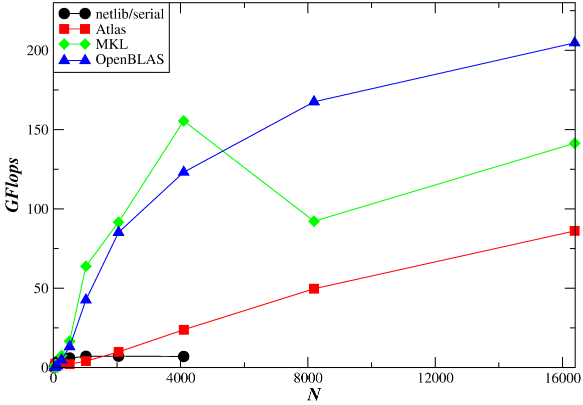
Numerical Libraries : BLAS/LAPACK
Linear Systems of Equations
SMP Performance Results on Euclid



Numerical Libraries : BLAS/LAPACK

Linear Systems of Equations

SMP Performance Results on a 24-cores/node SandyBridge



- Use of Blacs/Scalapack together with Blas/Lapack libraries.
- Blacs/Scalapack setup and distribute arrays across processes.
- Performance/Scalability depends on some parameters: MB, NB. Optimum usually around 100.
- For Function Calls/Parameters meaning in next slide see Scalapack manual.

```
....
call MPI_Dims_Create( nprocs, ndims, dims, ierr)
nprow = dims(1)
npcol = dims(2)
call blacs_get( -1, 0, context )
call blacs_gridinit( context, 'Row-major', nprow, npcol)
call blacs_gridinfo( context, nprow, npcol, myrow, mycol )
Mloc = numroc( M, MB, myrow, 0, nprow )
Nloc = numroc( N, NB, mycol, 0, npcol )
allocate( A( Mloc, Nloc ) )
allocate( B( Mloc ) )
allocate( ipiv( Mloc+MB ) )
allocate( work( M ) )
call descinit(descA, M, N, MB, NB, 0, 0, context, max(1,Mloc), info)
call descinit(descB, M, 1, MB,  1, 0, 0, context, max(1,Mloc), info)
do j=1,N
  do i=1,M
      call pdelset( A, i, j, descA, myrandom(iseed) )
  enddo
enddo
do i=1,M
  call pdelset( B, i, 1, descB, myrandom(iseed) )
enddo
call pdgesv( N, 1, A, 1, 1, descA, ipiv, B, 1, 1, descB, info )
.....
```

What is Fast Fourier Transform ?

- We have a function (or data points) as function of one variable, say $f(x)$.
- We need (for various reasons) to project this function (or data points) in x conjugate variable ξ space.

$$\textit{Forward} \quad \textit{FFT} \quad \hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx$$

$$\textit{Inverse} \quad \textit{FFT} \quad f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i x \xi} d\xi$$

Why this transformation is so frequently referred ? (most of you, already know at least one reason)

1. Signal processing : Digital Filtering (transform, drop say high frequencies, transform back without high frequencies)
2. Ewald Summation for Electrostatics for Molecular Dynamics/Monte Carlo etc. : Convert a (very) slowly converging integral into two fast converging integrals
3. Quantum Mechanics DFT calculations

Library FFTW3 (the successor of FFTW2)

Usage (in C) :

- Serial

```
plan = fftw_plan_dft_3d(N0,N1,N2,in,out,FFTW_FORWARD,FFTW_ESTIMATE);
fftw_execute(plan);
fftw_destroy_plan(plan);
```

- OpenMP

```
fftw_init_threads();
nthreads=omp_get_max_threads();
tzero=csecond_();
fftw_plan_with_nthreads(nthreads);
plan = fftw_plan_dft_3d(N0,N1,N2,in,out,FFTW_FORWARD,FFTW_ESTIMATE);
fftw_execute(plan);
fftw_destroy_plan(plan);
fftw_cleanup_threads();
```

- MPI version of library is also available

- Please login on training Machine
- Please load the following modules

```
module load iccifort
module load imkl
module load ATLAS
module load GCC
module load OpenMPI
module load CUDA
```

- Download the training material tarball :

```
git clone https://github.com/hpc-grnet-gr/prace-autumn-school-2014/
```

- Have a look in directory structure....
- untar training material.

Exercise 1 :

1. Compile all programs in Sources/BLAS directory :

   ```
   make
   make -f Makefile.cuda
   ```

2. Run some of these. slurm scripts are in the source directory : `runserial`, `runsmp`, `runcuda`

Exercise 2 :

Find the optimum NB for the Scalapack PDGESV for two (2) system sizes. In order to get results within this school timeframe plz. use sizes up to 2k.

1. See what solvesystemMPI.f.template does
2. Select few system sizes and a NB series in runsizes.sh
3. Prepare a SLURM Script that after all SLURM related stuff contains :

   ```
   bash runsizes.sh > outputfile
   ```
4. Submit the job(s) using 8 and 16 cores (i.e. 1 or 2 nodes), wait to finish.
5. Collect the results to discuss them if we have time. If time is not enough, we could discuss them in break or during dinner.

Summary

- Libraries should be used whenever possible
  - Use Optimized libraries, if you have a choice
  - Use common, standardized interfaces
- Libraries often depend on other libraries, especially BLAS.
- No single library or API covers everything
- Libraries have not the same behaviour on all architectures.
- MPI scaling behaviour on the same machine is not the same for different MPI flavours (OpenMPI, MPICH1, MPICH2, Intel MPI, IBM MPI etc.) and versions (Not explored here, because differences, when they are present, usually appear on more than 2048 cores)

Questions ?

Feel free to discuss here or later at
ntell at grnet dot gr

Self Training / HomeWork

Exercise 1 :

Write verification code to check the correctness of the solution.
Hint : Instead of filling b with random numbers, fill x with random numbers, use *DGEMM* or *DGEMV* to prepare b, keep backup of original x to x_{orig}, check that the solution x is (within say 1% accuracy) the same as the original x_{orig})

Exercise 2 :
Change the supplied code for the solution of a general linear system of equations for M=1 to concurrently solve a system with M=3 (3 is common for cartesian coordinates), i.e from the form :

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1N} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2N} \\ \cdot\cdot & \cdot\cdot & \cdots & \cdot\cdot\cdot\cdot \\ \alpha_{M1} & \alpha_{M2} & \cdots & \alpha_{MN} \end{bmatrix} x \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_N \end{bmatrix}$$

to

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1N} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2N} \\ \cdot\cdot & \cdot\cdot & \cdots & \cdot\cdot\cdot\cdot \\ \alpha_{M1} & \alpha_{M2} & \cdots & \alpha_{MN} \end{bmatrix} x \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{33} \\ \cdots & \cdots & \cdots \\ x_{N1} & x_{N2} & x_{N3} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ \cdots & \cdots & \cdots \\ b_{N1} & b_{N2} & b_{N3} \end{bmatrix}$$

Do not forget to check accuracy of solution (ex.1)

Exercise 3 :

Try to implement Hybrid Parallelization with no code changes.
Hint : Use SMP versions of BLAS/Lapack variants, playing with Makefile options, for each MPI Task. Of Course use the verification code of Exercise 1 to verify results correctness.

Exercise 4 :

The installed NVIDIA libcublas, currently does not support *DGESV* (it may happen with any driver routine or with any package that supports accelerators). Try to find a way to use GPU to solve a linear system of equations, although apparently this is not supported.

<u>Hint :</u> DGESV is using routines that are supplied by libcublas. If you find a solution, check its performance and of course accuracy.