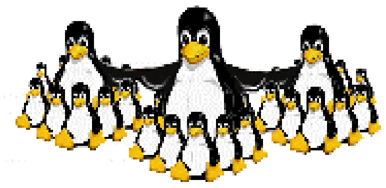
Tools for High Performance Computing (on linux clusters)



Stefano Cozzini (INFM udr Sissa)
(cozzini@sissa.it / 040-2240641)

Agenda

- Short review on compilers
  - A list
  - How to choose a compiler
  - A short comparison among them
  - A few information for the pgi suite
  - Some information on the Intel one
- Libraries for HPC
  - Overview (just a list)
  - Linear Algebra Libraries
    - Lapack –BLAS (serial)
    - scaLAPACK (parallel)
    - ATLAS project
  - MKL by Intel
  - FFTW library

Compilers for Intel/Linux

- Language
  - **Which is <u>the</u> programming language in HPC ?**
  - **Unfortunately... Fortran why ?**
    - **Tons of libraries written in Fortran**
    - **Tons of lazy users**
    - **Tons of computational codes written in F77:**
- Compilers
  - Free:
    - GNU http://www.gnu.org/ (free = open source)
    - Intel (free = individual license) http://developer.intel.com/software/products/compilers/index.htm
  - NON Free:
    - NAG http://www.nag.co.uk
    - PGI http://www.pgroup.com/
    - Lahey http://www.lahey.com/
    - Absoft http://www.absoft.com/

How to choose a compiler ?

- Efficiency
  - Does it produce efficient code ?
  - Is it able to exploit at best the hardware (Ex SSE2) ?
- Parallelism
  - Is it able to compile OpenMp directives ? Is it able to automatic parallelise on SMP machines ?
- Prices
  - FREE/COST
- Interoperability
  - Does it operate with other tools/compiler/languages ?
- Utilities / tools
  - Does it have a Debugger/ Profiler / other utilities ?
- Diagnostic Capabilities
  - Is it able to detected errors/bugs in programs ?
- Documentation/ support /training ...

A comparison on Fortran compilers (by www.polyhedron.com/)

- Supported language features..
  - Which constructs are allowed or not ?
- Diagnostic Capabilities:
  - 33 tests for f77 and 48 tests for F90
    - Argument mismatches
    - Bounds Checks
    - Uninitialised variables...
    - other....
  - Results:
    - g77 41%
    - Intel 63%
    - PGI 23%
    - Absoft 31%
- Performances:
  - Different tests for f77 and f90 (available on the lab-server: polyhedron.tgz)

Performances...

- ...it is clearly folly to expect that any single benchmark or aggregate will give an accurate indication of how another *untested program will perform on another untested computer*.
- However, if the results really matter, *there is no substitute* for testing with your *own* software and hardware.
- Proposed exercise: test the compilers against the polyhedron programs and against your own codes...

The PGI suite

- Quite diffused (even ICTP has its own server license...)
- AutoParallelization for multi-CPU
- Native Open-MP for multi-CPU
- support for Pentium III SSE
- Gnu Interoperability: can cross-link gcc/g77 libraries...
- PGDBG graphical debugger
- PGProf: graphical profiler
- Precompiled libraries (Blas/Lapack) come together
  - Hint: recompile them !!!

The Intel compiler

- Got the license ? (C++ or Fortran does not matter...)
- Features (from Intel site) :
  - Compatible with widely used Linux* software development utilities
  - Compatibility with Compaq Visual Fortran*
  - Excellent floating-point instruction throughput
  - Data prefetching
  - Interprocedural Optimization (IPO)
  - Profile-Guided Optimization (PGO)
  - Your choice of debuggers
- Further informations: read the “getting started with Intel Linux Compiler” (provided here)

IMHO

- Intel
  - diagnostic capability is too much (not a problem unless your code is something like 50k lines)
  - Performances: at the moment quite good…
- PGI:
  - diagnostic: not so good (it does not detect too many errors…)
  - Performances:still good but Intel is now performing better [at least on my codes]) than PGI compiler
- NAG:
  - Diagnostic: excellent !!
  - Performances: poor (at least on my code)

IMHO

- My software development cycle procedure:
  - Develop/debug/test code using NAG compiler
  - Cross check compilation using the Intel one after some large changes..
  - Compile production code with Intel/PGI/Absoft on the set of machines I want to use
  - Perform some short benchmarks on the simulation project (cross checking the results)
  - Start production with the best one…

Some Standard Mathematical libraries

- Vendor's library
  - ESSL Engineering and scientific subroutine library (IBM)
  - DXML/CXML Advanced mathematical library (DEC/Compaq)
  - SCSL Silicon Graphics
- ISV
  - IMSL International Mathematical and Statistical Libraries
  - NAG Numerical Algorithm group (UK labs)
- Free
  - **NETLIB** a WWW metalibrary of free math software
  - **SLATEC** comprehensive Mathematical and statistical Package
  - **LAPACK** Linear algebra package
  - **CERN** European center for nuclear research
  - **Petsc:** ODE/PDE parallel solvers
  - **FFTW:** fft library

Standard Linear Algebra problems

- Lots of computational problems can often be reduced in solving one or a sequence of the following standard problems:
  - Linear system of equations: A**x**=B where A is *n x n* non singular real or complex matrix , b is a column vector and x is a column vector we wish to compute.
  - Least square problems: compute the x which minimize $||Ax-b||^2$.
  - Eigenvalue problems: given a *n x n* matrix A find a n vector x and a scalar λ so that $Ax= \lambda x$

Examples of applications..

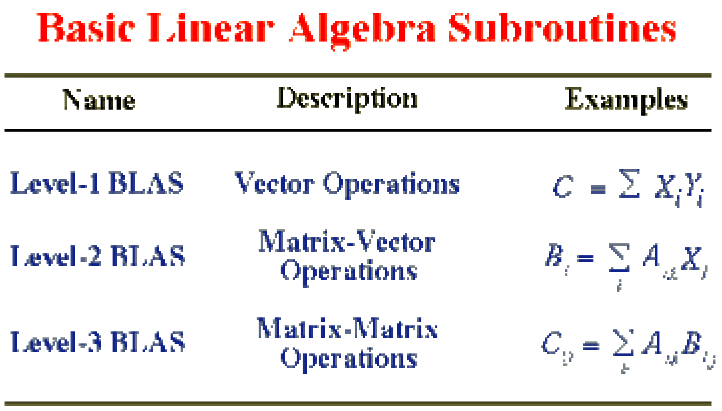
- Linear system often arise in numerical solution of differential equations
- Least squares methods are common in analysis of data
- Eigenvalue problems are always present in quantum mechanical problems..

LAPACK: Linear Algebra Package

- 1992: Dongarra et al.
- Supersedes and extends Eispack and Linpack packages…
  - Eispack Linpack: well-tuned for old machine
- It has been designed to be efficient on a wide range of modern high performance computer:
    - vector processing
    - risc workstations
    - shared memory multiprocessors
- It uses very efficient KERNELS : BLAS library
- Last version 3.0 (may 2000)

BLAS: Basic Linear Algebra Subprograms

- IDEA: create a standard to identify the basic set of operations involved in linear algebra problems;
- Objectives:
  - Accuracy;
  - Efficiency;
  - Portability;
  - Maintainability;
- Dates: 70's; ===> vector programming
  Fortran programming

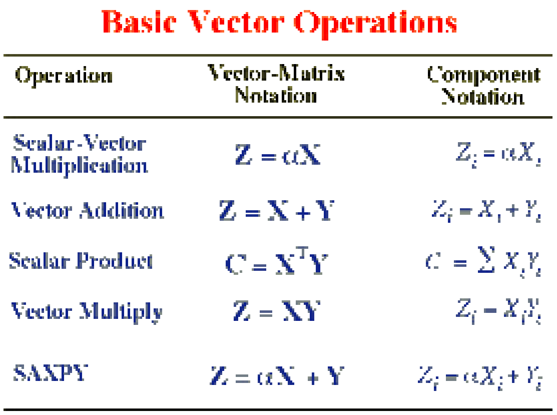


Basic Linear Algebra Subroutines

Name	Description	Examples
Level-1 BLAS	Vector Operations	$C = \sum X_i Y_i$
Level-2 BLAS	Matrix-Vector Operations	$B_i = \sum_j A_{ij} X_j$
Level-3 BLAS	Matrix-Matrix Operations	$C_{ij} = \sum_k A_{ik} B_{kj}$

BLAS: level 1

- 1979
- 58 routines based on vector operations:



Basic Vector Operations

Operation	Vector-Matrix Notation	Component Notation
Scalar-Vector Multiplication	$Z = \alpha X$	$Z_i = \alpha X_i$
Vector Addition	$Z = X + Y$	$Z_i = X_i + Y_i$
Scalar Product	$C = X^T Y$	$C = \sum X_i Y_i$
Vector Multiply	$Z = XY$	$Z_i = X_i Y_i$
SAXPY	$Z = \alpha X + Y$	$Z_i = \alpha X_i + Y_i$

BLAS level 2

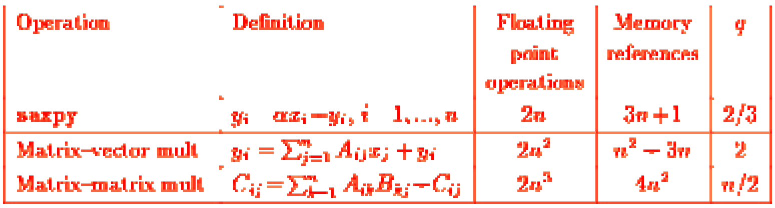
- 1988: Dongarra Ducroz Hammarling Hanson
- 30 routines for matrix- vector operations

BLAS level 3

- 1990: Dongarra Ducroz, Hammarling..
- 28 routines for matrix-matrix operations

Efficiency: q parameter

Table 2: Basic Linear Algebra Subroutines (BLAS)



Operation	Definition	Floating point operations	Memory references	q
saxpy	$y_i \quad \alpha x_i - y_i,\ i \quad 1,\ldots,n$	$2n$	$3n+1$	$2/3$
Matrix-vector mult	$y_i = \sum_{j=1}^{n} A_{ij} x_j + y_i$	$2n^2$	$n^2 - 3n$	2
Matrix-matrix mult	$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} - C_{ij}$	$2n^3$	$4n^2$	$n/2$

The parameter q is the ratio of flops to memory references. Generally:

1. Larger values of q maximize useful work to time spent moving data.
2. The higher the level of the BLAS, the larger q.

It follows…

- BLAS1 are not very efficient:

(for each computation a memory transfer is required)

- BLAS2 can achieve near peak-performance on vector computers. Good performance on super-scalar architecture

- BLAS3 can be very efficient on super-scalar computers !

Obtaining BLAS and Additional Information

- Frequently asked questions concerning BLAS can be found at the following site: **http://www.netlib.org/blas/faq.html**
- The BLAS can be obtained from the following URL: **http://www.netlib.org/blas**
- **FROM FAQ: Is there a C interface to the BLAS?** Yes, a C interface to the BLAS was defined in the BLAS Technical Forum Standard. The source code is also available.

Blas: how to install them

- On the infolab-51 server: /home/school/blas.tgz
- To compile it: use the makefile within it (not provided by netlib !)
- Exercise:
  - compile them with g77/pgf77 (just modify the Makefile)
  - Compare performances of blas1/blas2/blas3 for double precision data (dblat1.f dblat2.f dblat3.f again on server: test_blas.tgz)
  - Compare against precompiled pgi blas

LAPACK

- LAPACK is a library of Fortran 77 routines for solving the most common problems in numerical linear algebra. It can be downloaded for free from the netlib archives.
- It includes routines for:
  - Solving systems of simultaneous linear equations
  - Finding least squares solutions of overdetermined systems of equations
  - Solving eigenvalue problems
  - Solving singular value problems

different types of LAPACK routines

- Simple Driver Routines
  - solve a complete problem, like finding the eigenvalues of a matrix or solving a set of linear equations.
- Expert Driver Routines:
  - provide more options/information than the simple driver routines. Examples are, calculation of error bounds or equilibrating matrices to improve accuracy.
- Computational Routines
  - are called by the driver routines and perform a distinct computational task, like a LU factorization, or the reduction of a real symmetric matrix to tridiagonal form.

Naming scheme of Lapack

- All drivers and computational routines are of the form
  XYYZZZ
- X indicates the data type:
  - S REAL
  - D Double
  - C Complex
  - Z Double complex
- YY indicates the type of matrix:
- ZZZ indicates what the routine does...
- Example:
  - SGEBRD single precision(S) routine that performs a bidiagonal reduction (BRD) of a real general matrix. (GE)

Lapack: some warning

- Read carefully the man pages: some vendors have slightly changed something.
- The best usage of them is just calling drivers routines
- Computational routines can be used but take care of what you are doing..
- Check for naming convention and be sure about the type of your input data

Lapack: compilation tips…

- Tar zxvf lapack.tgz
- cd LAPACK
- Cp INSTALL/make.inc.Linux make.inc
- Edit make.inc changing what needed (compiler/ directory where to store library….)
- Then
- Make ; make install
- Compilation takes some time…
- Download the math_gnu.tar.bz2 package from the infolab-51 to get them precompiled (using g77)

LAPACK Related Projects

- Alternative language interfaces to LAPACK (or translations/conversions of LAPACK) are available in Fortran95, C, and Java.
  - http://www.netlib.org/lapack95/
  - http://www.netlib.org/clapack/
  - http://www.netlib.org/java/f2j/

scaLAPACK

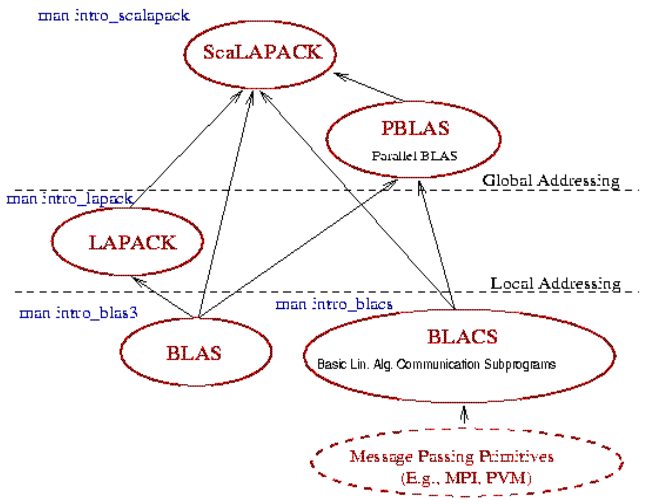
- 1995: Dongarra et. al.version 1.0 of
- Scalable Linear Algebra PACK AGE
  - (now version 1.7):
- parallel MP-implementation of LAPACK:

- Form FAQ:
- The **ScaLAPACK** (or Scalable LAPACK) library includes a subset of **LAPACK** routines redesigned for distributed memory MIMD parallel computers. It is currently written in a Single-Program-Multiple-Data style using explicit message passing for interprocessor communication. It assumes matrices are laid out in a two-dimensional block cyclic decomposition.

LAPACK and ScaLAPACK

	LAPACK	ScaLAPACK
Machines	Workstations, Vector, SMP	Distributed Memory, DSM
Based on	BLAS	BLAS, BLACS
Functionality	Linear Systems Least Squares Eigenproblems	Linear Systems Least Squares Eigenproblems (less than LAPACK)
Matrix types	Dense, band	Dense, band, out-of-core
Error Bounds	Complete	A few
Languages	F77 or C	F77 and C
Interfaces to	C++, F90	HPF
Manual?	Yes	Yes
Where?	www.netlib.org/ lapack	www.netlib.org/ scalapack

scaLAPACK: components

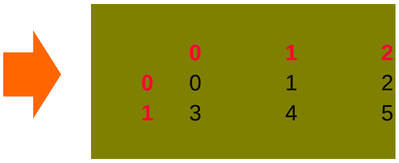


BLACS: Basic Linear Algebra Communication Subprogram

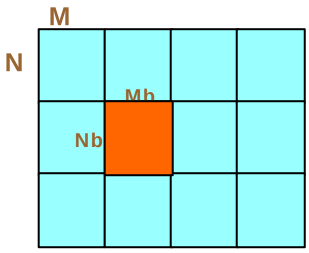
- 1995: Dongarra et al.
- Message Passing library for Linear Algebra
  - communication operation for matrix and vectors
    - communication point to point;
    - global communication;
- Defines a a 2D processors grid;
- A processor can belong to more that one grid (context) (cfr MPI communicator)
- Built on a standard MP library:
  - MPI/PVM (or specific libraries for specific machines)
  - We will use MPIBLACS

2D block-cyclic layout

- Grid of processors :
  - P x Q
  - enumeration is "row - major order"
- How to distribute the matrix ?
  - Matrix M xN is splitted in blocks of size Mb x Nb
  - blocks are cyclic distribute on the grid
- Context:
  - belongs to the matrix and to the processors

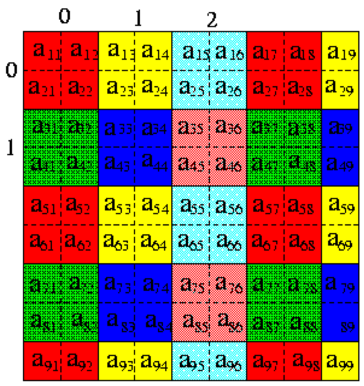


	0	1	2
0	0	1	2
1	3	4	5



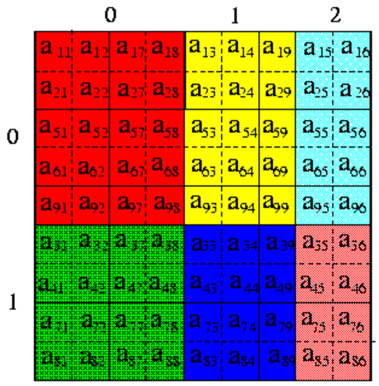
2D-cyclic distribution: example

- Matrix 9x9 blocks 2x2 :processors-grid 2x3



	0	0	1	1	2	2	0	0	1
0	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}
0	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}	a_{29}
1	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}	a_{38}	a_{39}
1	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}	a_{48}	a_{49}
0	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}	a_{57}	a_{58}	a_{59}
0	a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}	a_{67}	a_{68}	a_{69}
1	a_{71}	a_{72}	a_{73}	a_{74}	a_{75}	a_{76}	a_{77}	a_{78}	a_{79}
1	a_{81}	a_{82}	a_{83}	a_{84}	a_{85}	a_{86}	a_{87}	a_{88}	a_{89}
0	a_{91}	a_{92}	a_{93}	a_{94}	a_{95}	a_{96}	a_{97}	a_{98}	a_{99}

Global view



	0	0	0	0	1	1	1	2	2
0	a_{11}	a_{12}	a_{17}	a_{18}	a_{13}	a_{14}	a_{19}	a_{15}	a_{16}
0	a_{21}	a_{22}	a_{27}	a_{28}	a_{23}	a_{24}	a_{29}	a_{25}	a_{26}
0	a_{51}	a_{52}	a_{57}	a_{58}	a_{53}	a_{54}	a_{59}	a_{55}	a_{56}
0	a_{61}	a_{62}	a_{67}	a_{68}	a_{63}	a_{64}	a_{69}	a_{65}	a_{66}
0	a_{91}	a_{92}	a_{97}	a_{98}	a_{93}	a_{94}	a_{99}	a_{95}	a_{96}
1	a_{31}	a_{32}	a_{37}	a_{38}	a_{33}	a_{34}	a_{39}	a_{35}	a_{36}
1	a_{41}	a_{42}	a_{47}	a_{48}	a_{43}	a_{44}	a_{49}	a_{45}	a_{46}
1	a_{71}	a_{72}	a_{77}	a_{78}	a_{73}	a_{74}	a_{79}	a_{75}	a_{76}
1	a_{81}	a_{82}	a_{87}	a_{88}	a_{83}	a_{84}	a_{89}	a_{85}	a_{86}

Local (distributed) view

Array Descriptor

- Object that contains all the information on data distribution..
- Integer array:
  - 1.DESC(M_): row-number
  - 2. DESC(N_): column-number
  - 3. DESC(MB_): block-size (row)
  - 4. DESC(NB_): block-size (column) etc.. etc..
- array descriptor must be passed to the parallel routines

How to call a scalapack routine ?

- Initialize processors -grid
  - BLACS_PINFO: ask number of processors
  - BLACS_GET: obtain the context
  - BLACS_GRIDINIT: assign processors to grid
  - BLACS_GRIDINFO: identifies coordinates of each processors
- Distribute the global matrix
  - DESCINIT- assign the array descriptor to the global matrix
- call SCALAPACK routine
- release the grid

Example: diagonalizer

```
* Initialize the BLACS
      CALL blacs_pinfo(iam,nprocs)

IF(iam.NE.0)OPEN(6,FILE='/dev/null',STATUS='unknown')
      WRITE(6,'('' running on nprocs = '',i4)')nprocs
* Initialize a single BLACS context
      CALL blacs_get(-1,0,context)
      CALL blacs_gridinit(context,'r',nprow,npcol)
•allocate minimum work space
•CALL  worksize(n,nb,nprow,npcol,ilwork,iliwork)
      ALLOCATE(work(ilwork),STAT=ier)
      ALLOCATE(iwork(iliwork),STAT=ier)
*
```

```
* These are basic array descriptors
      CALL descinit(desca,n,n,nb,nb,0,0,context,maxy,info)

* same for eigenvectors
      CALL descinit(descz,n,n,nb,nb,0,0,context,maxy,info)

* distribute the global matrix in submatrixes
      CALL pdlamodhilb(n,a,a_global,1,1,desca,info)

*  Ask PDSYEVX to compute the entire eigendecomposition
      CALL pssyevx('v','a','u',n,a,1,1,desca,zero,zero,13,
     &              -13,mone,m,nz,w,mone,z,1,1,descz,work,
     &
ilwork,iwork,iliwork,ifail,iclustr,gap,info)

* do other stuff with the result
   ....
* exit
      CALL blacs_gridexit(context)
      CALL blacs_exit(0)
```

Examples at
www.netlib.org/scalapack

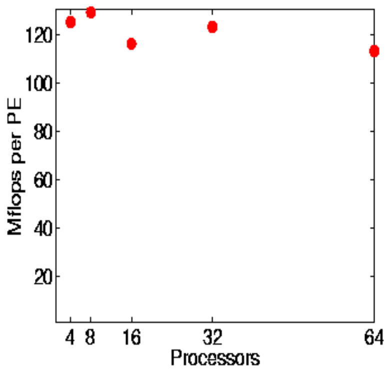
SCALAPACK: efficiency

- User is allowed to modify parameters to enhance performances..
  - Scalar/superscalar case : block size
  - Parallel case: block size: processors grid

High flexibility and “tuning” on real cases

Scalability

- PSGETRF:
  - triangular factorization
- N^2/nprocs=constant
- perfect scalability !

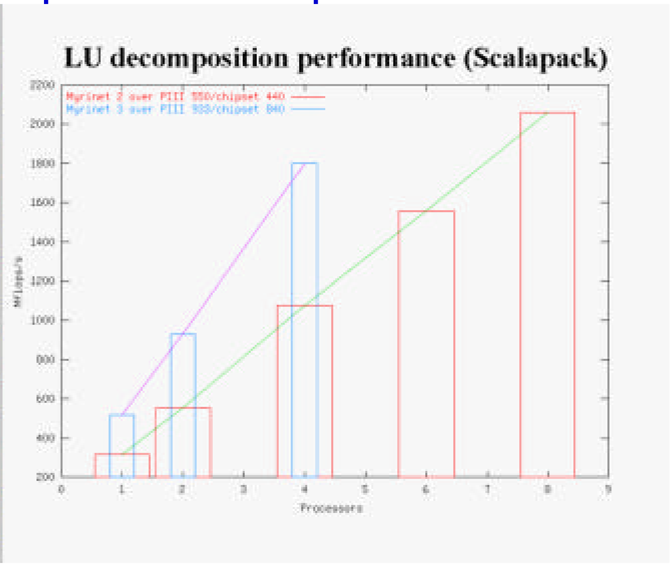


Scalapack:

- Portable and efficient
- At first glance look complex to use (especially data distribution)
- Warning : slight differences between vendor implementations and public domain version ;
- To learn more on them :

http://www.netlib.org/scalapack

scaLapack: which performances ?



- What about our oscar cluster ? And the cineca ones ?

Atlas project

- ATLAS & PhiPAC Projects (Automatically Tuned Linear Algebra Software)
  - Automatic generation of computational kernels for RISC architectures
  - Code generator takes about 1-2 hours to run.
  - Done once for a new architecture.
  - Written in ANSI C
  - Extension of BLAS to Sparse, Parallel and Mixed Precision Operations.
  - Extension to higher level operations.
  - SMPs
    Pentium III/IV

Atlas: which routines?

- From FAQ:

  The current version provides a complete BLAS API (for both C and Fortran77), and a very small subset of the LAPACK API. For all supported operations, ATLAS achieves performance on par with machine-specific tuned libraries.

Obtaining and Installing ATLAS

- Download ATLAS from the following URL: **http://www.netlib.org/atlas/atlas.tgz**
- Unpack the distribution and change into the ATLAS directory:
  - **tar -zxvf atlas.tgz**
    cd ATLAS

  (**NOTES:** read the README contained in this direcory. Also note that the following installation instructions can be found in the INSTALL file also contained within this directory.)
- configuration step: **make config CC=< ANSI C Compiler>**

  Note: If CC is not supplied, gcc is used by default. If your system has gcc, and this is the compiler you wish to use, simply type **make**.
  Config prompts for input and provides the remaining instructions for creating the include makefile.
- Installation step: **make install arch = < ARCH>** where ARCH is the architecture you choose during configuration.

  Note: The installation is self-sufficient, requiring no interaction from the user. Installation times may vary from 15 minutes to 4 hours. (on our PentiumIII almost 3 hours...)

MKL: Math Kernel Library (intel)

- Subset of LAPACK,BLAS, the extended BLAS (a set of sparse level 1 functions), FFTs, vector math functions, and the cblas interface to the BLAS.
- Specific version of the library optimized for the Pentium® III, Pentium 4, and Itanium processors as well as a default version for all other versions of Intel processors.
- LAPACK routines have been optimized at the blocking level / threaded for SMP processors.
- The FFTs have also been optimized and multithreaded for additional performance on multiprocessor computers.
- VML, a set of vectorized functions, provides substantial performance advantages on vectors of operations such as trigonometric functions, exponentials, logarithms, and so on.

MKL

- **Linear Algebra:**
  - Full set of LAPACK computational routines
  - Extended BLAS (sparse vector routines)
  - Level 1 BLAS (Vector operations)
  - Level 2 BLAS (Vector-matrix operations)
  - Level 3 BLAS (Matrix-matrix operations)
  - cblas interface--interface to BLAS for the C programmer
- **Signal Processing:**
  - Single and double precision FFTs--Fortran /C
  - 2D version
- **Vector Math:**
  - Vectorized transcendental functions.
- **Auxiliary:**
  - Testing routines with makefiles and run scripts
  - Example code for LAPACK routines

ATLAS vs INTEL…

- Exercise on Thursday…

FFT algorithm…

- Fourier Transform: frequency analysis of time series data.
- DFT: Discrete Fourier Transform (N time/freq points) NxN
- FFT: Fast Fourier Transform: efficient implementation ~O(Nlog$_2$N)

$$H(f)=\int_{-\infty}^{\infty} h(t)\,e^{2p\,i\,f\,t}dt \qquad H_n=\sum_{k=0}^{N-1} h_k e^{2p\,ikn/N}$$

$$h(t)=\frac{1}{2p}\int_{-\infty}^{\infty} H(f)\,e^{-2p\,i\,f\,t}dw \qquad h_k=\frac{1}{N}\sum_{n=0}^{N-1} H_n e^{-2p\,ikn/N}$$

$$h_k \equiv h(t_k),\ t_k \equiv k\Delta t,\ k=\{0,1,\text{☺},N-1\}$$

$$f_n \equiv \frac{n}{N\Delta t},\ n=\{-N/2,\text{☺},N/2-1\}$$

FFTW

- Fastest Fourier Transform of the West !
- Developed at MIT by M.Frigo and S.G.Johnson
- Public domain library (version 2.1.3)
  http://www.fftw.org

- Pros:
  - efficient
  - portable
  - written in C
  - parallel !
- Cons:
  - written in C

Characteristics of FFTW

- C routines generated by Caml-Light ML
- 1D/nD, real/complex data
- Arbitrary input size, not necessary 2^n
- Serial/Parallel, Share/Distributed Memory
- Faster than all others, high performance
- Portable, automatically adapt to machine

FFTW idea..

- FFTW does not use fixed algorithm but it can adapt the DFT algorithms to details of the underlying hardware in order to achieve best performance
- Computation is split in two phases
  - First FFTW Planner is called: it “learns” the fastest way to compute DFT on a particular machine and produce a plan (a data structure)
  - the plan along with array of data are then passed to the FFTW “executor”
- Planner phase can be expensive but if the produced plan is used several time this is acceptable
- Apply to all FFTw operation modes
  - 1D/nD, complex/real, serial/parallel

FTTW: operation modes

- One dimensional complex transform
- multidimensional complex transform
- One dimensional real transform
- multidimensional complex transform

- Each mode come with its own planner and executor !

FFTW: basic usage

```
#include <fftw.h>
....
{
      fftw_complex in[N] ,out[N] ;
      fftw_plan p;
      ....
      p= fftw_create_plan(N,FFTW_FORWARD,FFW_ESTIMATE);
      ...
      fftw_one(p,in,out);
      ....
      fftw_destroy_plan(p);
}
```

Can I call FFTW from FORTRAN ?

- From FFTW FAQ:
- Not directly. The main problem is that Fortran cannot pass parameters by value. However, FFTW can be called indirectly from Fortran through the use of special C "wrapper" routines. A package of appropriate wrapper code is included with FFTW

FFTw Fortran-Callable Wrappers

- Routine names, append `_f77` in C routine names
  - `fftw/fftwnd/rfftw/rfttwnd ->`
  - `fftw_f77/fftwnd_f77/rfftw_f77/rfttwnd_f77`
  - `fftw_mpi/fftwnd_mpi ->`
  - `fftw_f77_mpi/fftwnd_f77_mpi`
  - e.g. `fftwnd_create_plan(3, n_dim, FFTW_FORWARD, FFTW_ESTIMATE | FFTW_IN_PLACE)`
  - `-> fftwnd_f77_create_plan(`**`plan`**`, 3, n_dim, FFTW_FORWARD, FFTW_ESTIMATE` **`+`** `FFTW_IN_PLACE)`

FFTw Fortran-Callable Wrappers

- Notes
  - Any function that returns a value is converted into a subroutines with an additional (first) parameter.
  - No `null` in Fortran, must allocate and pass an array for `out`.
  - nD arrays, column-major, Fortran order
  - `plan` variables: be declared as `integer`
- Constants
  - `FFTW_FORWARD, FFTW_BACKWARD, FFTW_IN_PLACE`, ...
  - separated by '+' instead of '|'
  - In file `fortran/fftw_f77.i, fftw_f90.i, fftw_f90_mpi.i`

C/ Fortran Example 1

```
fftw_complex in[N], *out[N];
        fftw_plan plan;

        plan=fftw_create_plan(N,FFTW_FORWARD,FFTW_ESTIMATE);
        fftw_one(plan,in,out);
        fftw_destroy_plan(plan);
```

```
double complex in, out
    dimension in(N), out(N)
    integer plan
call fftw_f77_create_plan(plan,N,FFTW_FORWARD,FFTW_ESTIMATE)
call fftw_f77_one(plan,in,out)
call fftw_f77_destroy_plan(plan)
```

C/Fortran example 2

To transform a three-dimensional array in-place with C, you might do:

```
fftw_complex arr[L][M][N];
        fftwnd_plan plan;
        int n[3] = {L,M,N};

        plan = fftwnd_create_plan(3,n,FFTW_FORWARD,
                                   FFTW_ESTIMATE | FFTW_IN_PLACE);
        fftwnd_one(plan, arr, 0);
        fftwnd_destroy_plan(plan);
```

In Fortran, you would use this instead:

```
double complex arr
 dimension arr(L,M,N)
 integer n
 dimension n(3)
 integer plan

 n(1) = L
 n(2) = M
 n(3) = N
 call fftwnd_f77_create_plan(plan,3,n,FFTW_FORWARD,
+                           FFTW_ESTIMATE + FFTW_IN_PLACE)
 call fftwnd_f77_one(plan, arr, 0)
 call fftwnd_f77_destroy_plan(plan)
```

Note: Note that we pass the array dimensions in the "natural" order

Parallel FFTw

- Multi-thread
  - Skipped
- MPI
  - nD complex
    - Routines
    - Notes
    - Data Layout
  - 1D complex
  - nD real

1D Complex MPI FFTw

- Routines, similar to nD case, except no nd...
  - fftw_mpi_plan fftw_create_plan(mpi_comm comm,
    int n, fftw_direction dir, int flags);
  - void fftw_mpi_local_size(fftw_mpi_plan p,
    int *local_n, int *local_n_start,
    int *local_n_after_transpose,
    int *local_start_after_transpose,
    int *total_local_size);
    - See manual for more details..

1D Complex MPI FFTw (cont.)

- Routines (cont.)
  - void fftw_mpi(fftw_mpi_plan p, int n_fields, fftw_complex *local_data, fftw_complex *work, fftw_mpi_output_order output_order);

    void fftw_mpi_destroy_plan(fftw_mpi_plan p);

Tips on MPI version (from Manual)

- experiment with the best number of processors to use for your problem.
- The fftw_mpi_test program can output helpful performance benchmarks.
- It accepts the same parameters as the uniprocessor test programs (c.f. tests/README)
- Example:
  - mpirun -np 4 fftw_mpi_test -s 128x128x128 will benchmark a 128x128x128 transform on four processors, reporting timings and parallel speedups for all variants of fftwnd_mpi (transposed, with workspace, etcetera).

FFTw compilation tips

- ./configure –help : too see some option

- ./configure –prefix=/where/to/put/the/lib
- ./configure –-enable-mpi (for parallel version)
- ./configure –disable-fortran : to exclude the creation of the f77 wrappers...