A C and Matlab interface to LAPACK

by Rémi Delmas
supervised by Julien Langou

Innovative Computing Laboratory
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE
A C Interface to LAPACK

by Rémi Delmas
supervised by Julien Langou
Summary

1. Why a wrapper for C to LAPACK?

2. Design of the package

3. Testing!!!
/* Example on how to use LAPACK from C */
/* This code solve the symmetric positive definite system Ax=b */
/* using Cholesky factorization. One RHS*/

```c
int lda, ldb, n, nrhs;
double *A, *b;
/* Allocate and Affect values in A and b (column-major) */
```

/* access to LAPACK FORTRAN library */

```c
nrhs=1;
dposv_( "U", &n, &nrhs, A, &lda, b, &ldb, &info );
```

```c
#include “lapack.h”
/* access to LAPACK FORTRAN library through a wrapper */
lapack_dposv( lapack_uplo_U, n, 1, A, lda, b, ldb, &info );
```

Problem with direct access:
- Name mangling: dposv_, dposv, DPOSV, dposv__ ??
- Need to declare nrhs
- Assume types ok between Fortran and C
- Enums enables clearer code and useful debugging information
- Proto provided in the .h file
Conversion of types, example.

```c
#define C2F_CHAR(a) ( _cptofcd( (a), 1 ) )
#define C2F_STR(a, i) ( _cptofcd( (a), (i) ) )
#define F77_STRLEN(a) (_fcdlen)
```

```c
#define F77_INT long
```

```c
#ifdef CRAY

#include <fortran.h>
#define F77_CHAR _fcd
```

```c
#ifdef WeirdNEC

#define F77_INT long
```

```c
#endif
```

```c
#endif
```

1. **Implement a “nice” user interface, for C this means:**
   a) Avoid address passing where it is not mandatory.
   b) Enums
   c) Prototype
   d) Transparent conversion from C type to Fortran (and vice-versa) for logical/integer/character
   e) Takes care of the name mangling
   f) Double-layer for functions

2. **Be portable**
   a) It has to work on all machines where LAPACK runs with any combination C/Fortran compiler.
   b) Compatible with any LAPACK distribution (netlib, vendors,…).

3. **Be as efficient as possible**

4. **Have a test suite**

5. **Be automatically generated (as much as possible, at least)**
   a) To avoid maintenance
   b) Be usable by other Fortran packages

⇒ Portability of the user application code

⇒ The C user should not know that LAPACK is written in Fortran*
<table>
<thead>
<tr>
<th>Package</th>
<th>Name mangling</th>
<th>Conversion integer character logical</th>
<th>Enum</th>
<th>Avoid address passing if possible</th>
<th>Prototype</th>
<th>Double layer for function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK Fortran</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>LAPACK netlib great for Fortran</td>
</tr>
<tr>
<td>CBLAS wrapper</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>For BLAS only</td>
</tr>
<tr>
<td>CLAPACK</td>
<td>OK</td>
<td>OK</td>
<td></td>
<td></td>
<td>OK</td>
<td></td>
<td>F2C version of LAPACK (slow?, what about other dist.)</td>
</tr>
<tr>
<td>CLAPACK + clapackwrap</td>
<td>OK</td>
<td>OK</td>
<td></td>
<td>OK</td>
<td>OK</td>
<td></td>
<td>'Only drivers</td>
</tr>
<tr>
<td>Vendors, ISV LAPACK</td>
<td>OK</td>
<td>OK</td>
<td>??</td>
<td>OK</td>
<td>OK</td>
<td></td>
<td>$\dagger$</td>
</tr>
<tr>
<td>LAPACK ATLAS</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td></td>
<td>- Only DGESV/DPOSV + Row major/Column major</td>
</tr>
<tr>
<td>LPP (LAPACK PLUS PLUS)</td>
<td>OK</td>
<td>OK</td>
<td></td>
<td>OK</td>
<td>OK</td>
<td></td>
<td>For C++, automatic allocation in option</td>
</tr>
<tr>
<td>Babel/SIDL</td>
<td>OK</td>
<td>OK</td>
<td></td>
<td>OK</td>
<td>OK</td>
<td></td>
<td>Ask Piotr why bad – Ask Victor why great : discussion on the LAPACK developer forum</td>
</tr>
<tr>
<td>LAPACK C wrapper</td>
<td>OK</td>
<td>OK</td>
<td></td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>What I am speaking of today (+ test)</td>
</tr>
</tbody>
</table>
3. Design

- A wrapper looks like:

```
calling program (C)
main()

.h file

LAPACK wrapper (C)
lapack_dgesv()
lapack_dlange()

this

LAPACK (fortran)
dgesv()
dlange()
```
3. Design of clapack_C_wrapper

Makefile/Make.inc

src

lib

include

testing

examples
3. Design

liblapack.a

include

liblapack_C_wrap.a
3. Design

LAPACK_SRC_DIR

liblapack.a

include

liblapack_C_wrap.a

src → Makefile

Make.inc
3. Design

LAPACK_SRC_DIR

src → Makefile

Configure script (Julie) → Make.inc

liblapack.a

include

liblapack_C_wrap.a
3. Design

- LAPACK_SRC_DIR
- liblapack.a
- include
- liblapack_C_wrap.a

- generator-script
- src
- Makefile
- Configure script (Julie)
- Make.inc
3. Design

LAPACK_SRC_DIR

generator-script

testing

examples

src

Makefile

liblapack.a

include

liblapack_C_wrap.a

Configure script (Julie)

Make.inc
3. Design

- LAPACK_SRC_DIR
- lapack_enum_list
- lapack_pregen (1%)
- testing
- generator-script
- examples
- src
- Makefile
- Configure script (Julie)
- Make.inc
- liblapack.a
- include
- liblapack_C_wrap.a
void lapack_cgesv(const int n, const int nrhs, void * a, const int lda, int * ipiv, void * b, const int ldb, int * info )
{
    #ifdef F77_INT
        int i_local;
        F77_INT F77_ipiv[n*1];
        for(i_local=0;i_local<n*1;i_local++) {
            F77_ipiv[i_local]=(F77_INT) ipiv[i_local];
        }
    #else
        #define F77_n n
        ...
    #endif
    f77_cgesv(&F77_n, &F77_nrhs, a, &F77 lda, F77_ipiv, b, &F77 ldb, F77_info);
    ...
}
/** lapack_enum.h **/
enum lapack_uplo_type { 
  lapack_upper = 121,
  lapack_lower = 122,
  lapack_upper_lower = 123
};

/ * lapack_f77.h */
#if defined(ADD_)
  #define f77_cgesv cgesv_
#endif
#if defined(UPCASE)
  #define f77_cgesv CGESV
...
3. Design:
3.b: SCRIPT

Automatic generation

a) Based on the LAPACK comments section

c) Two steps

i. Used David Bindel's awk script to parse the LAPACK comments to extract useful information on the functions and their parameters. This information is written to a file as pseudo-code.

   ii. Then a perl-script parses this file to generate the actual wrapper source.

d) Need correct and standardize LAPACK comment. If comments are wrong or not in the standard format, the parser fails.

   => Many many LAPACK comments were changed mostly for bugs (580 lines in 359 routines changed over 1293 routines).

e) The 2-level parsing allows for easily writing wrappers in other languages.
3. SCRIPT = standard comments

Enforcing LAPACK standardization in comments:

» If the norm is not respected, the script will fail, or worse, give false results.
» Order of appearance of the variable in the comments must be the same than in the prototype
» Lines begin with the variable name
» (input) (output) (input/output) (input or output) mandatory
» Type in upper case
» If array, put the word “array” somewhere on the line, along with the dimension between ().
» Everything must be on one line
» Character of size 1 are noted character*1
3. SCRIPT = standard comments

Enforcing LAPACK standardization in comments:

» If the norm is not respected, the script will fail, or worse, give false results.
» order of appearance of the variable in the comments must be the same than in the prototype
» lines begin with the variable name
» (input) (output) (input/output) (input or output) mandatory
» type in upper case
» if array, put the word “array” somewhere on the line, along with the dimension between ().
» everything must be on one line
» character of size 1 are noted character*1

But SCRIPT = correct comments !!!!
4. Testing

» Two ways to test.

» The easiest one: unwrap the wrapper ;) to allow for the testing suite provided in the LAPACK package to call the wrapper.

» Problem: when a LAPACK function calls a LAPACK function, that call does not go through the wrapper (eg, dgesv will be tested but not dgetrf or dgetrs)

» => Only the drivers are tested.

» The “hard” one:

» Change all LAPACK functions so that the internal calls are done through the wrapper.
4. Testing

- Problem with first testing: only drivers are tested.
4. Testing

» Solution: hack lapack so that call to aux routines go through the wrapper.
4. Testing

» Some functions were awful. Best of:
  » Size of array that depend on other parameters (like a char)...
  » Size of array that depends on the result of some other function (how I am supposed to know that size?)...
  » Some inconsistensies in the CHARACTER*1 parameters:
    » In *ggsvd, JOBU = 'U' or 'N'
    » JOBV = 'V' or 'N'
    » JOBQ = 'Q' or 'N'
    » .... but they all have the same role!
  » Function pointers...
  » Differenciation of enums is based only on the parameter name. But JOB may have many meanings... Lots of hacks in the script...
  » Places were a character would be needed, but an integer is used.
4. Testing

? Wrapper = Performance issue ?

» Timing was not done, but for default settings the execution time should be more or less the same (function call)

» On the testing:
  » Our testing calls two wrappers (instead of one)
  » Small matrices are used (sometime no computation)
  » Recursive levels of wrapping (instead of only the driver in the general case)

we are 1% more expensive than pure LAPACK

» The copy is a problem (cpu- and memory-wise), but we found no way to correct that.
Conclusion

➢ Complete automatic generation is an utopia, but we came close enough (thanks to some hacks...). Only 18 functions had to be done “by hand’.

➢ We now have a working, portable, complete and tested C wrapper.

➢ Webpage: http://icl.cs.utk.edu/~delmas