Note Taker Checklist Form -*MSRI*

Name : Rob Stapleton

E-mail Address/ Phone #: jrstaple@ncsu.edu

Talk Title and Workshop assigned to:
Interactive Parallel Computing in Support of Research in Algebra, Geometry and Number Theory

Lecturer (Full name): Yuzo Ishida

Date & Time of Event: 10:30 a.m. Jan 30, 2007

Check List:

(✓) Introduce yourself to the lecturer prior to lecture. Tell them that you will be the note taker, and that you will need to make copies of their own notes, if any.

(✓) Obtain all presentation materials from lecturer (i.e. Power Point files, etc). This can be done either before the lecture is to begin or after the lecture; please make arrangements with the lecturer as to when you can do this.

(✓) Take down all notes from media provided (blackboard, overhead, etc.)

(✓) Gather all other lecture materials (i.e. Handouts, etc.)

(✓) Scan all materials on PDF scanner in 2nd floor lab (assistance can be provided by Computing Staff) – Scan this sheet first, then materials. In the subject heading, enter the name of the speaker and date of their talk.

Please do **NOT** use **pencil** or colored pens other than black when taking notes as the scanner has a difficult time scanning pencil and other colors.

Please fill in the following after the lecture is done:

1. List 6-12 lecture keywords: parallel, interactive, ScaLAPACK, Lapack, arbitrary precision, GMP, MPFR, ARPREC

2. Please summarize the lecture in 5 or less sentences.
Lapack and ScaLapack are widely-used packages in Scientific computation. There is a small but significant group who want very high precision. In a parallel environment, implementing arbitrary precision is not always obvious.

Once the materials on check list above are gathered, please scan ALL materials and send to the Computing Department. Return this form to Larry Patague, Head of Computing (rm 214)

10:30 a.m.
Moving Sca/Lapack to Higher Precision (without too much work)
Yozo Hida

Lapack 3.1 has been released for about 2 months now.
- -Improved MRRR algorithm for symmetric eigenproblems
- -Faster Hessenberg QR
- -Faster Hessenberg, tridiagonal, bidiagonal reductions.
- -Mixed precision iterative refinement.
- -Thread safety

Current Activities:
- Faster Algorithms
  - -O(n^2) solver for polynomial roots via companion matrix
- More Accurate Algorithms
  - -Higher precisions
- Expanded contents (more Lapack in ScaLapack)
- Improve ease of use

What could go into Lapack?

```
for all linear algebra problems do
      for all matrix types do
            for all data types do
                  for all architectures and networks do
                        for all programming interfaces do
                              Procude fastest algorithm providing
acceptable accuracy;
                              Produce most accurate algorithm running at
acceptable speed;
                        od;
                  od;
            od;
      od;
od;
```

This talk will focus on what we can do as far as data types goes, especially multiple precision.

Motivation
- Lapack and ScaLapack are widely used.
- Significant minority want multiple precision
- Want to support codes like:

```
      n_bits <- 32
      repeat
            n_bits <- n_bits x 2
            Solve with n_bits
      until error < tol
```

Fixed Precision:
- Compiler supported quad
- Double-double, quad-double

Arbitrary Precision
- GMP / MPFR
- ARPREC

Double-double / Quad-double Package:
- -These represent a higher precision numers as an unevaluated sum of 2 or 4 double precision numbers
- -Ex: 2^60 + 1 is represented as (2^60, 1)
- -Can represent about 32/64 digits of precision
- -Highly efficient algorithms due to fixed, small size.

-Simple representation. Easy for allocation or to use with MPI
-Somewhat fuzzy definition of "machine precision" for double-double
-Exponent range limited by that of double precision.
-C/C++/Fortran 95 interafaces
-Support for ocmplex data types recently added

GMP/MPFR:
-One of most widely used multiple-precision floating point computations with correct rounding.
-Speed
-Uses a somewhat complicated data structure, mixing various types in a C structure.

ARPREC:
-Uses simple floating point array to represent data.
-Message passing is made easier
-Slower than GMP

Considerations:
-Code will be easier to maintain if only a single source code is used for varying precision.
-Need to let the user know how much workspace to allocate
-Allocation of temporary variables
-Multiple precisions in one instance: allowable?
-How to adjust precision.
-Co-existence of multiple versions?

Workspace allocation:
Specifying syze by bytes doesn't work well in Fortran. Specifying size by bignum slots works okay if every bignum in the workspace has the same size.
Allocate ourself? Would be nice for cache friendliness and ease of message passing.

Temporary Variable Allocation: Memory for temp variables needs to be allocated somewhere.
Could use external malloc routine or explicitly allocate. With fixed precisions, have compiler automaticall allocate
Could be solved through the use of macros.

Memory allocation: how much? when? Where?
Fixed Precision, the easiest approach.
Many compilers provide support for quad precision
OMF77 supports multi-precision variables (precision is compile-time selected)
Maximium precision:
User specifies maximum precision at compile time. At run time, program specifies precision to be used. Memory allocation issues are handled automatically, but memory can be wasted needlessly.
Variable precision:
Memory allocation becomes tricky. Memory must be allocated dynamically.

A simple replacement of variable types is not enough when working in multiple precision. The compiler must see each piece of a variable as a multiple precision number so that multiple precision. Also, when dealing with constants not expressable as a double, extra care is needed.

There are naming issues that need to be resolved as more packages are

added.

Implemented currently as a Perl script to convert Lapack sources to perform what is needed.

Iterative Refinement:
Use Newton's iteration on r(x) = b-Ax but compute the residual in double the working precision
You can get a more accurate answer up to a point. Error does not grow with condition number up to a point.

Once the high-precision library is set up appropriately, most of the existing Lapack code can be transformed with little trouble.

Future work:
More comprehensive error handling.
How much performance can we buy from using multi-core BLAS
Apply this to ScaLapack.

Recent Results in Fast Stable matrix Computations:
-Coppersmith and Winograd, 1990, fastest
-Strassen et al. showed O(n^a) matrix multiplication implies O (n^a) other things.
-Cohn, Kleinberg, Szegedy, Umans 2005 algorithm equals or beats Coppersmith and Winograd, depending on finding the right groups. IT uses Wedderburn's Theorem to reduce matrix multiplication to FFT.
New results (Demmel, Dumitriu, Holtz, Kleinberg 2006):
-CKSU(2005) algorithm is stable
-Any O(n^a) algothm can be converted to a stable one (based on Rax, 2003)
-There are stable algorithms for QR, LU, solve, and determinant costing O(n^(a+e)) for e>=0

Moving SCA/LAPACK to Higher Precision

without *too much* work

Yozo Hida

yozo@cs.berkeley.edu

Computer Science Division
EECS Department
U.C. Berkeley

January 30, 2007

Outline

)utline

Current Status

- LAPACK 3.1 has been released.
  - Improved MRRR algorithm for symmetric eigenproblem.

    2006 SIAM SIAG LA Prize winning algorithm of Dhillon, Parlett
  - Faster Hessenberg QR (up to 10×).

    2003 SIAM SIAG LA Prize winning algorithm of

    Braman, Byers and Mathias.
  - Faster Hessenberg, tridiagonal, bidiagonal reductions.
  - Mixed precision iterative refinement.
  - Thread safety (`save` and `data` statements removed).
  - Many bug fixes.
- Feedback welcome: `http://www.netlib.org/lapack-dev/`

Current Activities

- Faster algorithms.
  - Recursive blocked layouts.
  - Better QZ.
  - $O(n^2)$ companion matrix solver (polynomial roots).
- More accurate algorithms.
  - Extra-precise iterative refinement.
  - Jacobi SVD.
  - Higher precisions.
- Expanded contents: More LAPACK in SCALAPACK.
- Automated performance tuning.
- Improve ease of use.
- Community involvement (this means you!)

See `www.cs.berkeley.edu/~demmel` for details.

ome Participants – more are welcome!

- UC Berkeley

  Jim Demmel, Ming Gu, W. Kahan, Beresford Parlett, Xiaoye Li, Osni Marques, Christof Vömel, David Bindel, Yozo Hida, Jason Riedy, Jianlin Xia, Jiang Zhu
- U. Tennessee, Knoxville

  Jack Dongarra, Victor Eijkhout, Julien Langou, Julie Langou, Piotr Luszczek, Stan Tomov
- Other Academic Institutions

  UT Austin, UC Davis, Florida IT, U Kansas, U Maryland, North Carolina SU, San Jose SU, UC Santa Barbara TU Berlin, FU Hagen, U Madrid, U Manchester, U Umeå, U Wuppertal, U Zagreb
- Research Institutions

  CERFACS, LBNL
- Industrial Partners

  Cray, HP, Intel, MathWorks, NAG, SGI

What could go in LAPACK?

for all linear algebra problems **do**
for all matrix types **do**
for all data types **do**
for all architectures and networks **do**
for all programming interfaces **do**
Produce fastest algorithm providing acceptable accuracy.
Produce most accurate algorithm running at acceptable speed.

What could go in LAPACK?

for all linear algebra problems **do**
for all matrix types **do**
for all data types **do**
for all architectures and networks **do**
for all programming interfaces **do**
Produce fastest algorithm providing acceptable accuracy.
Produce most accurate algorithm running at acceptable speed.

Some prioritization and automation is obviously needed.

Rest of the talk will concentrate on data types, specifically precisions higher than double precision.

Motivation for higher precision

- LAPACK and SCALAPACK are widely used.
- User survey revealed small but significant portion of users wanted precision higher than double precision.
  `http://icl.cs.utk.edu/lapack-forum/survey/`
- We want to do least amount of work to support multiple precision.
- We want to support codes like

  n_bits $\leftarrow$ 32
  repeat
    n_bits $\leftarrow$ n_bits $\times$ 2
    Solve with n_bits
  until error $<$ tol

Outline

ome Implementations of Higher Precision

- Fixed precision
  - Compiler supported quad
  - Double-double, Quad-double
- Arbitrary precision
  - GMP / MPFR
  - ARPREC

)ouble-double / Quad-double Package

- `http://crd.lbl.gov/~dhbailey/mpdist/`.
- These represents a higher precision numbers as an unevaluated sum of 2 or 4 double precision numbers.
- Example: $2^{60} + 1$ is represented as the pair $(2^{60}, 1)$.
- Can represent about 32 / 64 digits of precision,
- Highly efficient algorithms due to their fixed, small size.
- Simple representation: fixed size array of doubles.
  Makes it easy to allocate arrays and use MPI.
- Somewhat fuzzy definition of "machine precision" for these numbers:
  $1 + 2^{-1000}$ can be represented exactly in double-double, but not $1 + 2^{-500} + 2^{-1000}$.
- Exponent range limited to that of double precision.

Double-double / Quad-double Package

- C, C++ and Fortran 95 interfaces.

```
subroutine f_main
use qdmodule
type (qd_real) a, b
a = 1.d0
b = cos(a)**2 + sin(a)**2 - 1.d0
call qdwrite(6, b)
end subroutine
```

- Support for complex data types recently added.

GMP/MPFR

- Multiple-precision floating-point computations with correct rounding.
- `http://www.mpfr.org/`
- Uses integer arithmetic instructions.
- Highly optimized for variety of platforms.
- Somewhat complicated data structure, mixing various types in a C struct.
  This makes inter-language operation, porting, and message passing somewhat harder.
- C, C++ interfaces. No Fortran 95 interface.

ARPREC

- http://crd.lbl.gov/~dhbailey/mpdist/.
- Uses simple floating point array to represent data.
  This makes inter-language operation and message passing easier.
- Slower than GMP. (sometimes by factor of 10).
- C, C++, and Fortran 95 interfaces.

```
subroutine f_main
use mpmodule
type (mp_real) a, b
call mpinit (500)
a = 1.d0
b = cos(a)**2 + sin(a)**2 - 1.d0
call mpwrite(6, b)
end subroutine
```

)utline

Things to Consider

- Single source code for varying precision. This makes it much easier to maintain code.
- Workspace allocation. We need to tell the user how much workspace to allocate.
- Temporary variable allocation.
- Should we allow multiple precisions in one routine? Within one matrix?
- How to adjust precision.
- Can multiple versions co-exist? Naming issues.

Workspace Allocation

Many LAPACK routines requires workspaces.
How should we specify the size?

- by number of bytes? Doesn't work well in Fortran.
- by number of bignum slots? Works OK if every bignum in the workspace has the same size.

We also like the workspace to have some contiguity for cache friendliness and ease of message passing.

Temporary Variable Allocation

Memory for temporary variables need to be allocated somewhere:

$$x = a + b + c$$

The temporary result $(a + b)$ must be stored somewhere.

- Have the compiler automatically allocate (for fixed precisions),
- Use external malloc routine,
- Explicitly allocate.

ixed vs. Variable Precision

- Memory allocation: how much? when? where?
- Variable precision allows us to support codes like

```
n_bits ← 32
repeat
    n_bits ← n_bits × 2
    Solve with n_bits
until error < tol
```

ixed Precision

This is the easiest approach.

- Many compilers (not all) provide support for quad precision.
- One compiler (OMF77) even supports multi-precision variables (precision is compile-time selected).
- Fortran 90 modules and interfaces can be used to provide operator overloading to custom types (double-double, quad-double).
- Memory allocation issues can be handled automatically.

Maximum Precision

- User specifies maximum precision at compile time.
- At run time, the program specifies the precision to be used (less than the maximum specified at compile time).
- Memory allocation issues can be handled automatically.
- Wastes memory if precision used is much smaller than the maximum precision.

Fortran 95 interface of ARPREC currently works in this mode.

/ariable Precision

- User can specify precision to use at run-time.
- Memory allocation issues can get tricky:
  - Memory must be allocated dynamically.
  - For cache (and communication) efficiency, we want to allocate a single block for each matrix.
  - Do we allow differing precisions within one algorithm? within one matrix?
- Only allocates necessary memory.

C++ interface of ARPREC currently work in this mode. Work is currently being done to make Fortran 95 interface works in this mode as well.

Outline

Modules

The high precision library provides a module that declares what operators and functions are overloaded:

```
module mpmodule
  type mp_real
    sequence
    real*8 mpr(mpwds+5)
  end type

  interface operator (+)
    module procedure mpadd
  end interface

...

end module mpmodule
```

Modules

LAPACK source is then modified slightly:

```
subroutine some_lapack_routine
  !  Use module for arbitrary precision
  use mpmodule

  !  Declare variables in the desired format
  type (mp_real) a, b, c

  !  LAMCH must be provided by the
  !  arbitrary precision package provider
  smlnum = lamch(a, 'Safe minimum')

  !  ...the rest of the code ...
end subroutine
```

Assessment of Constants

Simple replacement of variable type is not enough:

```
x = 0.1d0 / 0.3d0
```

This line is interpreted by the compiler as double precision constants and computation. Needs to be replaced with something like,

```
x = mp_real(0.1d0) / mp_real(0.3d0)
```

or if dealing with constants not representable in double precision:

```
x = mp_real("0.1") / mp_real("0.3")
```

This is the most common "bugs" reported to us when people convert their code to use our higher precision packages.

Naming Issues

How should a routine be named when using a new precision?

- Add a new prefix. DGESVX becomes QGESVX for quad, QD_GESVX for quad-double etc.
  Leads to name explosion, but easier for inter-language operations.
- Use generic names like "GESVX" and dispatch to correct routines using Fortran 95 module and interface facilities.
  LAPACK 95 does this.

.imitations with Fortran 95

- Fortran 95 does not allow I/O routines to be overridden. LAPACK uses I/O only in the test code, so often we can just print out the double-precision approximation instead.
- Fortran 95 does not have a general destructor, making it hard to micromanage memory allocation / destruction. (Fortran 2003 fixes this).
- Annoying Fortran 77 formatting (e.g., line length).

mplementation

Currently implemented as a Perl script to convert LAPACK sources to perform

- use appropriate module file (provided by the dd / qd / arprec packages),
- constant literal substitutions,
- handle Fortran data declarations,
- use a new prefix for each LAPACK routines,
- substitute something appropriate for read / write statements, and
- declare variables in appropriate types.

Passes many LAPACK tests for Quad and QD precisions, including linear systems.
Work being done on ARPREC implementations.

Outline

terative Refinement

Use Newton's iteration on $r(x) = b - Ax$ but compute the residual in double the working precision:

$\hat{x} \leftarrow A^{-1}b$ (using basic solution method)

repeat

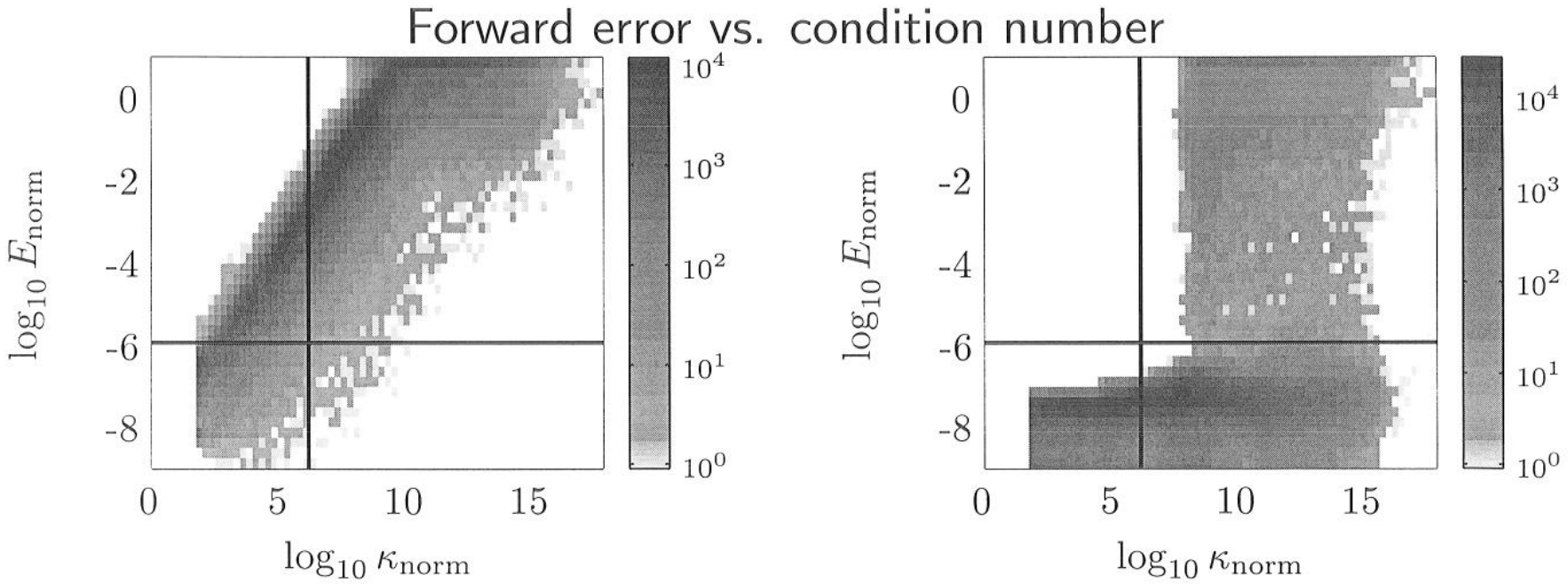
$r \leftarrow A\hat{x} - b$ (compute residual in doubled precision)

$dx \leftarrow A^{-1}r$ (solve for correction)

$x \leftarrow \hat{x} - dx$ (update solution)

until convergence **or** no progress

terative Refinement



We can obtain small errors until condition number gets too bad; at which point we just need more precision.

Accurate Summation

Theorem

Suppose we have n f-digit, base-b floating point numbers, and we sum them in decreasing order of their exponent using a F-digit accumulator. Then if $n \leq 1 + \frac{b^{F-f}}{1-b^{-f}} \equiv N$ then the computed sum $\hat{s}$ has a relative error bounded by

$$\left| \frac{s - \hat{s}}{\hat{s}} \right| < \frac{1}{2} \left[b + 1 + \frac{3b^{-f}}{1 - b^{1-f}} \right] b^{-f}$$

For b^f moderately large, relative error is bounded by just over $\frac{1}{2}(b+1)b^{-f}$. If $n \geq N + 2$, then relative error can be arbitrary.

Accurate Summation

If we use x86 80-bit floating point format as accumulator, we can add $2^{11} + 1 = 2049$ doubles accurately.

Previous theorem is especially useful when dealing with multi-precision variables, since

- sorting by exponent is often much faster, and
- we can often pick the size of the accumulator (F) to achieve a desired accuracy, based on the number of terms to be added.

ARPREC uses $b = 2^{48}$ as the base, so we just need to keep an extra word to compute the sum accurately.

Outline

Conclusion

Once the high-precision library provides the appropriate module file declaring the overridden functions, most of LAPACK code can be transformed to use them without too much effort.

Outline

uture Work

- Exception handling.
- How much performance can we buy from using multi-core BLAS?
- Apply same technique to SCALAPACK.
- Provide F95 interface to MPFR. Anyone?

Recent Results in Fast, Stable Matrix Computations

- Current record for fast matrix multiplication: $O(n^{2.38})$. (Coppersmith & Winograd, 1990).
- Strassen et al. showed $O(n^a)$ matmul implies $O(n^a)$ matrix inversion, determinant etc.
- New algorithm by Cohn, Kleinberg, Szegedy, Umans (2005)
  - Uses Wedderburn's Theorem to reduce matmul to FFT.
  - Equals Coppersmith & Winograd, beating it depends on finding right groups.
- New results (Demmel, Dumitriu, Holtz, Kleinberg, 2006)
  - Above algorithm is stable (normwise).
  - Any $O(n^a)$ algorithm can converted to a stable one (based on Raz, 2003)
  - There are stable algorithms for QR, LU, solve, determinant costing $O(n^a)$ or $O(n^{a+\epsilon})$ for any $\epsilon > 0$.

Parallel sparsening and simplification of systems of equations

Part II

Thomas Wolf, Winfried Neun

Zuse Institute Berlin

MSRI Workshop Feb2007

History of Parallel REDUCE
(machines used)

Cray Vector Machines 1986..

Cray MPP (using PVM) 1995..

RISC workstation networks 1998..

Brock Beowulf cluster 2002..

Sharcnet's Opteron clusters 2005 …

Ideas

- Using more than one (all?) processor for very hard computer algebra calculations on whatever hardware the computing centre people just recently bought. Nobody buys a huge machine for CA today.
- Coarse grain parallelism / Cray vector assembler c.
- Dedicated to some challenging applications e.g. Gröbner bases, QE (RedLog, U Passau) Crack (TW, Brock U)

Ideas II

- Adapting to available machine models SPMD /MPMD, clusters etc. Playstation??
- We use **PVM** (Oak Ridge Nat.Lab.) and **LISP**
- Using more than one CA system. We added interfaces to Singular (U. Kaiserslautern) and Gb (J.-C. Faugère), others ad libitum.

  'Auf Wiedersehen' etc..
- Use standards like OpenMath or MathML!

Ideas III

- Make parallel operation available at user level ‘algebraic mode’.
- Interactive use, e.g. creating processes on user demand: “create_process”, send/receive,kill_process
- Dynamic behaviour, i.e. the number of used processors/cores is not determined at startup

Remarks

- Interactive parallel computation is a night-mare for the administrators in the computer centres
- Ability to quickly adapt to local needs, e.g. PVM, MPI, qsub (batch steps) is important
- Kathy Yelick's remark on the different nature of symbolic applications.

Remarks II

PVM does the dirty job of spawning tasks, finding a free node, transport layer. Binary transport of LISP data.

We have the job to serialize LISP structures and forward them to the nodes. This requires a system ‘in hand’

Conclusions and future work

We are trying to solve problems which have not been soluble so far. In some cases we have been successul.

To do:

Threads, Mult Cores,Cell processors, ..

Reentrant LISP code must be fun!

Thank you for your attention !