MANCHESTER 1824

NLA GROUP MANCHESTER

The University of Manchester

ChopBLAS: Simulating Mixed-Precision and Stochastically Rounded Linear Algebra

Ian McInerney, Nick Higham
Department of Mathematics
The University of Manchester

i.mcinerney17@imperial.ac.uk

SIAM CSE 2023, Amsterdam, NL

Motivation

Enormous interest in mixed precision and stochastic rounding in NLA

THREE-PRECISION GMRES-BASED ITERATIVE REFINEMENT FOR LEAST SQUARES PROBLEMS*

ERIN CARSON†, NICHOLAS J. HIGHAM‡, AND SRIKARA PRANESH‡

Accelerating Restarted GMRES With Mixed Precision Arithmetic

Neil Lindquist, Piotr Luszczek, and Jack Dongarra, *Fellow, IEEE*

Climate Modeling in Low Precision: Effects of Both Deterministic and Stochastic Rounding

E. Adam Paxton,[a] Matthew Chantry,[a] Milan Klöwer,[a] Leo Saffin,[b] and Tim Palmer[a]
[a] *University of Oxford, Oxford, United Kingdom*
[b] *University of Leeds, Leeds, United Kingdom*

STOCHASTIC ROUNDING AND ITS PROBABILISTIC BACKWARD ERROR ANALYSIS*

MICHAEL P. CONNOLLY†, NICHOLAS J. HIGHAM†, AND THEO MARY‡

The Positive Effects of Stochastic Rounding in Numerical Algorithms

El-Mehdi El Arar[1], Devan Sohier[1], Pablo de Oliveira Castro[1], Eric Petit[2]
[1]Université Paris-Saclay, UVSQ, LI-PaRAD
[2]Intel Corp
{el-mehdi.el-arar, devan.sohier, pablo.oliveira}@uvsq.fr
eric.petit@intel.com

FIVE-PRECISION GMRES-BASED ITERATIVE REFINEMENT *

PATRICK AMESTOY†, ALFREDO BUTTARI‡, NICHOLAS J. HIGHAM§, JEAN-YVES L'EXCELLENT†, THEO MARY¶, AND BASTIEN VIEUBLÉ‖

Motivation

But tedious and slow to test

```
% Implement c = alpha*Ax + beta*y
c = zeros(length(x), 1);
for i=1:length(c)
    c(i) = chop( beta*y(i), mulopts);
    for j=1:1:size(A, 2)
        tx = chop( alpha*x(j), mulopts );
        c(i) = chop( c(i) + chop( A(i,j) * tx, mulopts ), addopts );
    end
end
```

ChopBLAS to the Rescue

Requirements

- Provide basic linear algebra functions
- Support mixed precision operations
- Good performance with large matrices

Features

- BLAS-like interface
- Selectable rounding function
- Per-operation rounding options
- Selectable reduction operator

```
function [nrm] = chnrm2( x, varargin )
%CHNRM2 Compute the 2-norm of the vector x with operation-level rounding

%% Setup the argument parsing
p = inputParser;
p.StructExpand = false;
addOptional( p, 'mulopts', struct([]) );
addOptional( p, 'addopts', struct([]) );
addOptional( p, 'sqrtopts', struct([]) );
addParameter( p, 'Rounding', @chop );
addParameter( p, 'Accumulator', @chaccum_recursive );

parse( p, varargin{:} )

accum     = p.Results.Accumulator;
mulopts   = p.Results.mulopts;
addopts   = p.Results.addopts;
sqrtopts  = p.Results.sqrtopts;
roundfunc = p.Results.Rounding;

% Allow only the first to be specified and have it be used for both
if ( isempty(addopts) || isempty(sqrtopts) ) && ~isempty(mulopts)
    addopts  = mulopts;
    sqrtopts = mulopts;
end

pp = roundfunc( x.*x, mulopts );
dot = accum( pp, roundfunc, addopts );
nrm = roundfunc( sqrt( dot ), sqrtopts );

end
```

Implemented Functions

Function	Operation[1]	Description
`chscal`	αx	Scale all entries of the vector x by α
`chaxpy`	$\alpha x + y$	Add the scaled vector x to the vector y
`chdot`	$x' y$	Compute the dot product between x and y
`chnrm2`	$\|\|x\|\|_2$	Compute the 2-norm of the vector x
`chasum`	$\sum_i \|x_i\|$	Compute the sum of the absolute value of the elements of the vector x
`chgemv`	$\alpha\, \mathrm{op}(A)x + \beta y$	Compute the matrix-vector product $Ax + y$
`chtrmv`	$\mathrm{op}(A)x$	Compute the matrix-vector product Ax when A is a triangular matrix
`chtrsv`	Find x in $\mathrm{op}(A)x = b$	Compute the solution to the triangular system of equations given by A and b
`chger`	$\alpha x y^T + A$	Compute the rank-1 update of A using the scaled outer product between x and y

[1] $\mathrm{op}(A)$ can either be $\mathrm{op}(A) = A$ or $\mathrm{op}(A) = A'$

Reduction Operators

- Recursive summation
- Pairwise summation
- Sorted summation
  - Ascending sort
  - Descending sort
  - Insertion sort
- Compensated summation
- Doubly compensated summation

Changing the reduction operator

```
z = chnrm2( x, halfopts, 'Accumulator', @chaccum_pairwise );
```

Rounding Operator

Chop-based rounding (Default) [Higham and Pranesh(2019)]

```
z = chnrm2( x, halfopts );
z = chnrm2( x, halfopts, 'Rounding', @chop );
z = chnrm2( x, halfopts, singleopts, doubleopts, 'Rounding', @chop );
```

Cpfloat-based rounding [Fasi and Mikaitis(2023)]

```
z = chnrm2( x, halfopts, 'Rounding', @cpfloat );
```

Custom rounding

```
addopts.digits = 1;
multopts.digits = 2;
sqrtopts.digits = 3;
z = chnrm2( x, multopts, addopts, sqrtopts, 'Rounding', @(x, s) round( x, s.digits ) );
```

Implementation

Key Idea

- Vectorize as many rounding function calls as possible
- Use MATLAB's elementwise ops + Implicit expansion

Input: Matrix $A \in \mathbb{R}^{m \times n}$, vectors $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, and scalars α and β

Output: Result vector $\hat{y} \in \mathbb{R}^m$

$\hat{x} \leftarrow \circ_\times(\alpha x)$

$\hat{y} \leftarrow \circ_\times(\beta y)$

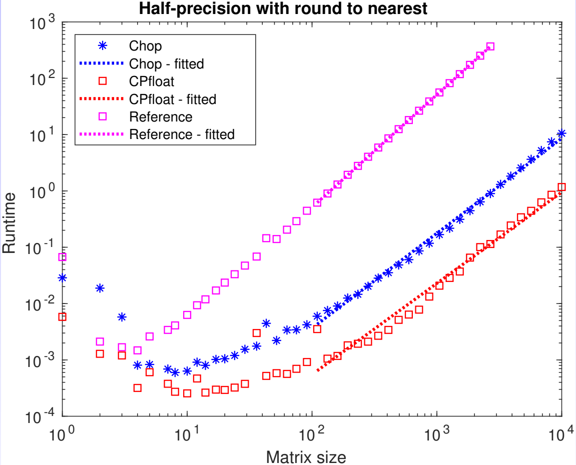
$$V \leftarrow \circ_\times\left(\begin{bmatrix} A_{1,:} \odot \hat{x}' \\ \vdots \\ A_{m,:} \odot \hat{x}' \end{bmatrix}\right)$$

$\hat{y} \leftarrow \text{sum}\left(\begin{bmatrix}\hat{y} & V\end{bmatrix}, \circ_+\right)$

return $\hat{y}$

```
% Initialize output using scaled y vector
xout = roundfunc( beta.*y, mulopts );

% Apply the scaling on the matrix-vector product
x = roundfunc( alpha.*x, mulopts );

if trans
    % Matrix indexing needed to compute the transposed product
    matind = @(i) A(:,i)';
else
    % Matrix indexing needed to compute the non-transposed product
    matind = @(i) A(i,:);
end

lx = length(x);
for i=1:blocksize:lx
    inds = i:1:min(i+blocksize-1, lx);

    t = [xout(inds), roundfunc( matind(inds).*x', mulopts )];

    xout(inds) = accum( t, roundfunc, addopts );
end
```

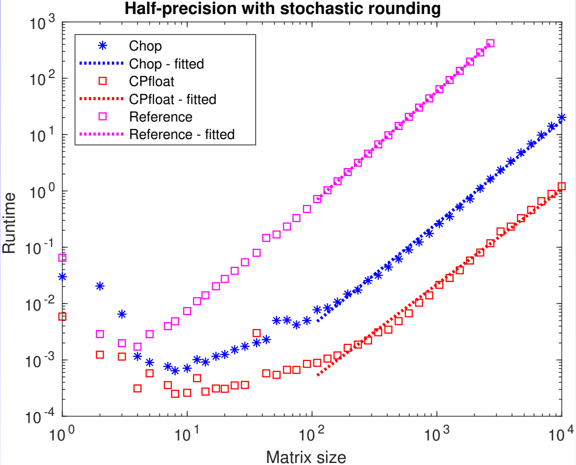
Performance (i)



	Reference	ChopBLAS (chop)	ChopBLAS (cpfloat)
Scaling	1.9988	1.6852	1.6185

[1] Test system: Intel Xeon W-2255, 128GB RAM

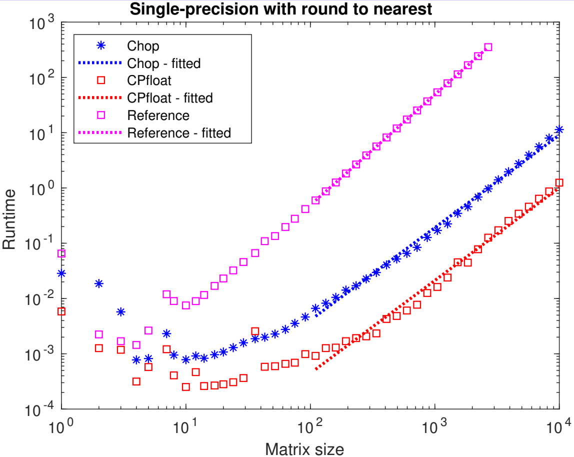
Performance (ii)



	Reference	ChopBLAS (chop)	ChopBLAS (cpfloat)
Scaling	1.9991	1.8162	1.6858

[1] Test system: Intel Xeon W-2255, 128GB RAM

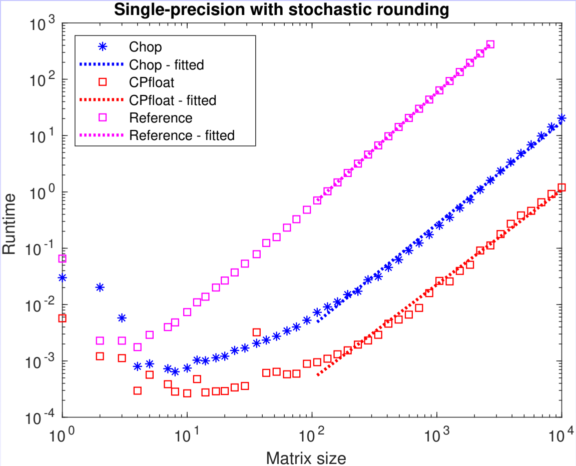
Performance (iii)



	Reference	ChopBLAS (chop)	ChopBLAS (cpfloat)
Scaling	1.9985	1.6734	1.6778

[1] Test system: Intel Xeon W-2255, 128GB RAM

Performance (iv)



	Reference	ChopBLAS (chop)	ChopBLAS (cpfloat)
Scaling	1.9982	1.8126	1.6803

[1] Test system: Intel Xeon W-2255, 128GB RAM

Example: Stochastically Rounded CG

```
% Configure chop rounding options (single precision, stochastic rounding)
roundopts.format = 's';
roundopts.round = 5;

% Initial values
x(:,1) = x0;
r(:,1) = chgemv( -1.0, A, x0, 1.0, b, roundopts );                        % r = b - a*x
p(:,1) = r(:,1);                                                          % p = r
s(:,1) = chgemv( 1.0, A, r(:,1), 0.0, [], roundopts );                    % s = A*r
v(1) = chdot( r(:,1), r(:,1), roundopts );                                % v = r'r
alpha(1) = chop( v(1) / chdot( p(:,1), s(:,1), roundopts ), roundopts); % a = v / p's

for i = 2:1:max_iter
    x(:,i) = chaxpy( alpha(i-1), p(:,i-1), x(:,i-1), roundopts );      % x(i) = alpha*p(i-1) + x(i-1)
    r(:,i) = chaxpy( -alpha(i-1), s(:,i-1), r(:,i-1), roundopts );     % r(i) = -alpha*s(i-1) + r(i-1)

    v(i) = chdot( r(:,i), r(:,i), roundopts);                           % v(i) = r(i)'r(i)
    beta(i) = chop( v(i) / v(i-1), roundopts );                         % beta(i) = v(i) / v(i-1)

    p(:,i) = chaxpy( beta(i), p(:,i-1), r(:,i), roundopts );            % p(i) = beta*p(i-1) + r(i)
    s(:,i) = chgemv( 1.0, A, p(:,i), 0.0, [], roundopts );              % s(i) = A*p(i)

    u(i) = chdot( p(:,i), s(:,i), roundopts );                          % u(i) = p(i)'s(i)
    alpha(i) = chop( v(i) / u(i) );                                     % a(i) = v(i) / u(i)
end
```

Conclusions

- Provides an easier way to simulate mixed-precision and stochastically rounded NLA
- Exploits built-in vectorization of MATLAB to scale better
- More BLAS functions to implement
  - `chtrsm`
  - `chgemm`
  - Maybe more...

Get it now

https://github.com/imciner2/ChopBLAS

References

- Massimiliano Fasi and Mantas Mikaitis.
  CPFloat: A C library for simulating low-precision arithmetic.
  ACM Transactions on Mathematical Software, In press, February 2023.
  doi: 10.1145/3585515.
- Nicholas J. Higham and Srikara Pranesh.
  Simulating Low Precision Floating-Point Arithmetic.
  SIAM Journal on Scientific Computing, 41(5): C585–C602, 2019.
  doi: 10.1137/19M1251308.