MPLAPACK version 2.0.1

An Extension of BLAS and LAPACK for Multiple Precision Arithmetic

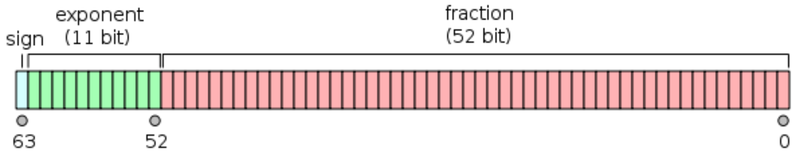
中田真秀 (Nakata Maho) RIKEN, ICIAM 2023, Waseda Univ, E803, 2023-08-21 16:40-17:10 JST

What is LAPACK?

- **Definition: LAPACK stands for "Linear Algebra PACKage."**
- **Purpose: While utilizing BLAS as its building block, LAPACK can solve more advanced problems such as systems of linear equations, least squares solutions, eigenvalue problems, and singular value problems.**
- **Subroutine Offerings: LAPACK provides a suite of subroutines for matrix decompositions, including LU decomposition, Cholesky decomposition, QR decomposition, singular value decomposition, Schur decomposition, and generalized Schur decomposition. It also offers routines for condition number estimation and inverse matrix calculations.**
- **Quality Assurance: LAPACK boasts meticulous and systematic quality assurance, ensuring it's a reliable tool for computations.**
- **Compatibility: LAPACK operates on a wide range of systems, from personal computers to supercomputers, and is compatible with various CPUs and operating systems.**
- **Development: Written in Fortran 90, version 3.11.0 comprises over 1900 routines.**
- **Popularity: The official website has garnered an impressive 200 million hits and counting.**
- **Ongoing Development: Development continues actively on GitHub. https://github.com/Reference-LAPACK/lapack/**
- **Official Website: https://www.netlib.org/lapack/**

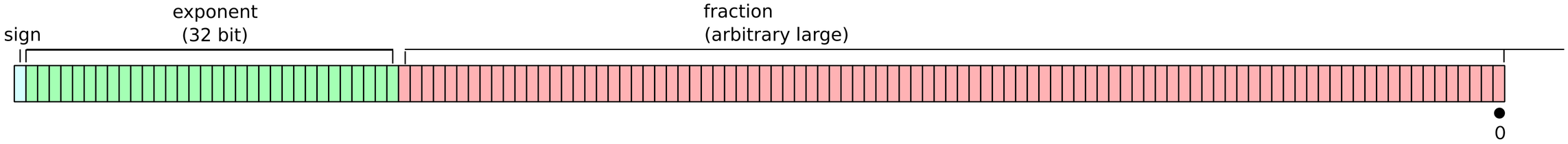
What is Multiple Precision?

- On computers, we can't represent real values with perfect accuracy, so we rely on floating-point numbers, which are approximations.
  - Typically, we use formats like binary64 and binary32, as defined in the IEEE 754-2019 Standard for Floating-Point Arithmetic or bfloat16 etc.



$$\pm\left(1+\sum_{n=1}^{52} a_n \overbrace{\left(\frac{1}{2}\right)^n}^{fraction} \times \overbrace{2^{-1022\sim1023}}^{exponent}\right)$$

  - The binary64 format provides approximately 16 decimal digits of precision.
  - However, there are times when we need **greater precision than 16 decimal digits**! In such cases, we simply **add more to the fraction**. This approach is called multiple precision. Various extensions exist for practical use.



In which situations is multiple precision required?

- Semidefinite programming
  - The condition number of linear equations diverges as it approaches optimality.
  - many applications: mathematics, vast topics in physics, optimization

$$
\begin{aligned}
\text{Primal SDP:} \quad & \text{minimize} && \sum_{i=1}^{m} c_i x_i \\
& \text{subject to} && \boldsymbol{X} = \sum_{i=1}^{m} \boldsymbol{F}_i x_i - \boldsymbol{F}_0, \\
& && \boldsymbol{X} \succeq \boldsymbol{O} \\
\text{Dual SDP:} \quad & \text{maximize} && \boldsymbol{F}_0 \bullet \boldsymbol{Y} \\
& \text{subject to} && \boldsymbol{F}_i \bullet \boldsymbol{Y} = c_i \ (i = 1, 2, \ldots, m), \\
& && \boldsymbol{Y} \succeq \boldsymbol{O}.
\end{aligned}
$$

  - cf. https://scholar.google.co.jp/scholar?q=SDPA-GMP
- ill conditioned linear systems, large summentions, Long-time simulations, Large scale simulations, Resolving small scale-phenomena, Experimental mathematics,
  - cf. https://doi.org/10.1016/j.amc.2012.03.087 and https://www.mdpi.com/2227-7390/3/2/337

Precision Support in MPLAPACK

- MPLAPACK supports an array of formats, including GMP, MPFR, binary128, double-double, quad-double, binary64, and _Float64x (an 80-bit extended double).
- GMP: Provides arbitrary precision without specific rounding modes and is the fastest, especially for fractions larger than 256 bits.
- MPFR: Offers arbitrary precision with features resembling IEEE 754 standards; however, it operates slower than GMP.
- binary128: defined in IEEE 754-2019 as genuine quadruple precision.
- double-double: Extremely fast pseudo quadruple precision.
- binary64: Can be interpreted as the C++ counterpart of LAPACK.
- _Float64x: Represents an 80-bit extended double, as outlined in IEEE 754-1985

What is the MPLAPACK 2.0.1?

- Overview: MPLAPACK is a multiple precision version of BLAS and LAPACK.
- Precision Support: Compatible with GMP, MPFR, binary128, quad-double, double-double, double, and binary80.
- API & Language: Provides a C++ API based on LAPACK 3.9.1. Users can utilize MP floating point numbers similarly to 'double'. Entirely re-written from Fortran90 to C++.
- Implementation: All routines are fully implemented, excluding mixed precision routines.
- Functionality: Supports both real and complex numbers, eigenvalue computations, singular value decompositions, least square fits and more!
- Compatibility: Works seamlessly on Linux, Mac, and Windows platforms.
- Performance: The double-double version of Rgemm achieves a remarkable 600 GFlops on A100
- Licensing: Distributed under the 2-clause BSD license.

https://github.com/nakatamaho/mplapack

MPLAPACK's C++ Advantage

- Flexibility in Precision: It's simpler to introduce new precision types using class.
- Simplified Conversion: For programs written in C or C++, merely replacing 'double' with '_Float128' or 'mpf_class' is often sufficient to create a multiple precision version.
- Flexibility: C++ can be used for both low-level (like systems programming) and high-level applications, making it versatile; e.g., memory management
- Community and Resources: C++ has a vast community, which means more libraries, tools, and resources are available. This can lead to faster problem-solving and more third-party support.
- Modern Features: Modern C++ (C++11, C++14, C++17, etc.) has introduced a range of advanced features that can simplify and optimize code, such as lambda expressions, smart pointers, and concurrency features.

Proof of concept

Porting SDPA-GMP, -MPFR, -DD, -QD, -binary128

- SDPA: https://sdpa.sourceforge.net/
  - A very fast semidefinite programming solver
  - developed by Fujisawa, Nakata (not myself), Kojima, Yamashita and others.
- A simple replacement of 'double' with 'mpf_class', 'mpreal', 'dd_real', 'qd_real', and '_Float128' was all that was needed.
- They work seamlessly!
- https://scholar.google.co.jp/scholar?q=SDPA-GMP : 167 citations

Fortran to C++ Conversion of LAPACK Routines

- LAPACK, originally developed in Fortran90, and is a huge dense linear algebra library.
  - approximately 80 BLAS routines and **1000** LAPACK routines in version 3.9.1.
  - Some automation is necessary for converting from Fortran90 to C++.
- Convert Fortran90 code to human-readable C++ code
  - f2c produces C codes that are compilable and runnable
    - It's extremely hard to read and only supports up to FORTRAN 77.
  - FABLE converts Fortran90 code to C++ and produces human-readable code.
    - dsyev (eigenvalue solver for sym. mat.) can be converted without modification!
    - some manual adjustments are occasionally needed.
    - https://cci.lbl.gov/fable/
    - The Fortran EMulation library is a header-only library that manages I/O routines.

Fortran to C++ Conversion of LAPACK Routines

- BLAS and LAPACK are written with clarity.
  - Fable can almost automatically convert BLAS and LAPACK.
- Neither BLAS nor LAPACK utilize I/O.
  - Resulting MPLAPACK doesn't rely on the Fortran EMulation library.
  - We've made specific patches to Fable for BLAS and LAPACK.
    - especially for leading dimensions and handling matrices.
- The TESTING routines involve a lot of I/O; using very complex format.
  - Extremely hard to port by hand
  - The Fortran EMulation library is only used for TESTING.

F2C translated codes are hard to read...

```
        if (nota) {

/*           Form  C := alpha*A*B + beta*C. */

            i__1 = *n;
            for (j = 1; j <= i__1; ++j) {
                if (*beta == 0.) {
                    i__2 = *m;
                    for (i__ = 1; i__ <= i__2; ++i__) {
                        c__[i__ + j * c_dim1] = 0.;
/* L50: */
                    }
                } else if (*beta != 1.) {
                    i__2 = *m;
                    for (i__ = 1; i__ <= i__2; ++i__) {
                        c__[i__ + j * c_dim1] = *beta * c__[i__ + j * c_dim1];
/* L60: */
                    }
                }
                i__2 = *k;
                for (l = 1; l <= i__2; ++l) {
                    if (b[l + j * b_dim1] != 0.) {
                        temp = *alpha * b[l + j * b_dim1];
                        i__3 = *m;
                        for (i__ = 1; i__ <= i__3; ++i__) {
                            c__[i__ + j * c_dim1] += temp * a[i__ + l *
                                    a_dim1];
/* L70: */
                        }
                    }
/* L80: */
```

```
/* Subroutine */ int aladhd_(integer *iounit, char *path)
{
    /* Format strings */
    static char fmt_9999[] = "(/1x,a3,\002 drivers:  General dense matrice"
            "s\002)";
    static char fmt_9989[] = "(4x,\0021. Diagonal\002,24x,\0027. Last n/2 co"
            "lumns zero\002,/4x,\0022. Upper triangular\002,16x,\0028. Random"
            ", CNDNUM = sqrt(0.1/EPS)\002,/4x,\0023. Lower triangular\002,16x,"
            "\0029. Random, CNDNUM = 0.1/EPS\002,/4x,\0024. Random, CNDNUM = 2"
            "\002,13x,\00210. Scaled near underflow\002,/4x,\0025. First colu"
            "mn zero\002,14x,\00211. Scaled near overflow\002,/4x,\0026. Last"
            " column zero\002)";
    static char fmt_9981[] = "(3x,i2,\002: norm( L * U - A )  / ( N * norm(A"
            ") * EPS )\002)";
    static char fmt_9980[] = "(3x,i2,\002: norm( B - A * X )  / \002,\002( n"
            "orm(A) * norm(X) * EPS )\002)";
    static char fmt_9979[] = "(3x,i2,\002: norm( X - XACT )   / \002,\002( n"
            "orm(XACT) * CNDNUM * EPS )\002)";
    static char fmt_9978[] = "(3x,i2,\002: norm( X - XACT )   / \002,\002( n"
            "orm(XACT) * (error bound) )\002)";
    static char fmt_9977[] = "(3x,i2,\002: (backward error)   / EPS\002)";
    static char fmt_9976[] = "(3x,i2,\002: RCOND * CNDNUM - 1.0\002)";
    static char fmt_9972[] = "(3x,i2,\002: abs( WORK(1) - RPVGRW ) /\002,"
            "\002 ( max( WORK(1), RPVGRW ) * EPS )\002)";
```

Fable + FEM handle "format" in C++!!

```
write(6, "(a)"), "ab";
write(6, "(2a)"), "cd", "ef";
write(6, "(a3)"), "gh";
//Cierr write(6, '(a3.1)') 'ij'
write(6, "(2a3)"), "kl", "mn";
//C
//Cgerr write(6, '(d)') 2.0
//Cgerr write(6, '(2d)') 3.0, 4.0
//Cierr write(6, '(d8)') 5.0
write(6, "(d9.2)"), 6.0f;
write(6, "(2d9.2)"), 7.0f, 8.0f;
//C
//Cgerr write(6, '(e)') 2.0
//Cgerr write(6, '(2e)') 3.0, 4.0
//Cierr write(6, '(e8)') 5.0
write(6, "(e9.2)"), 6.0f;
write(6, "(2e9.2)"), 7.0f, 8.0f;
```

Surprisingly, this is C++ code, not FORTRAN77 code.

Fable converted Rgemm code

```
if (nota) {
    //
    //          Form  C := alpha*A*B + beta*C.
    //
    for (j = 1; j <= n; j = j + 1) {
        if (beta == zero) {
            for (i = 1; i <= m; i = i + 1) {
                c[(i - 1) + (j - 1) * ldc] = zero;
            }
        } else if (beta != one) {
            for (i = 1; i <= m; i = i + 1) {
                c[(i - 1) + (j - 1) * ldc] = beta * c[(i - 1) + (j - 1) * ldc];
            }
        }
        for (l = 1; l <= k; l = l + 1) {
            temp = alpha * b[(l - 1) + (j - 1) * ldb];
            for (i = 1; i <= m; i = i + 1) {
                c[(i - 1) + (j - 1) * ldc] += temp * a[(i - 1) + (l - 1) * lda];
            }
        }
    }
```

Quality assurance

- How to verify our Fortran90 to C++ conversion is difficult task.
  - Even the original LAPACK's tests are quite good, but cannot be complete.
- We ported original LAPACK's TESTING routines.
  - involves lot of complicated "FORMAT" statement

What LAPACK TESTINGS do?

- LAPACK has a TESTING directory for its quality assurance.
  - LIN/xlintstd: Tester for binary64 real linear equation routines
  - LIN/xlintstz: Tester for binary64 complex linear equation routines
  - EIG/xeigtstd: Tester for real eigenvalue problems and singular value problems
  - EIG/xeigtstz: Tester for complex eigenvalue problems and singular value problems
  - MATGEN/: Matrix generation library for testing
- If these are fed files named xxx.in and they terminate normally, then it's considered okay.

LIN/xlintstd: Tester for binary64 real linear equation routines

- Ax=b (general linear equation solver), lapack-3.xx.y/TESTING/LIN/dchkge.f
- It tests DGETRF, -TRI, -TRS, -RFS, and -CON.
- First, it creates the correct answer.
  - The matrix A is varied under different conditions. The condition number of A is scaled to the vicinity of overflow and underflow, or set to values like 2. The norm of A is also varied similarly.
- For DGETRF, since A is decomposed into LU, if |A_new - A| is sufficiently small, then it's considered okay. Precisely, it's (norm(L*U - A) / (N * norm(A) * EPS)).
- Next, using DGETRI, the inverse matrix of A is determined... and so on.
- Since the solution can be found with only algebraic operations, there's no need to be too rigorous.

LIN/xlintstd: Tester for binary64 real linear equation routines

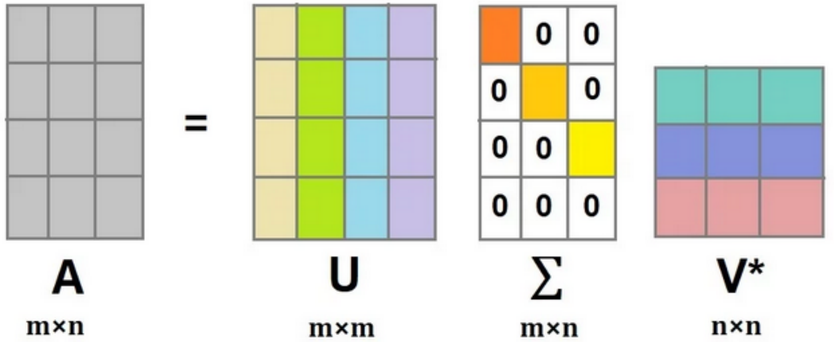
- We conduct tests like → when building LAPACK.
- xlintstd: Quality assurance program for linear equations in the binary64 version.
- dtest.in: Describes which routines to test and how, and it's fed into xlintstd.
- DGE: Checks for LU decomposition, inverse matrices, and systems of linear equations, etc.
- DGB: Band matrix version of DGE.
- DGT: Triangular matrix version of DGE.
- TESTING/LIN/dchkaa.F

```
(fable38) docker@cd4e49cbd3ea:~/mplapack/external/lapack/work/internal/lapack-3.
xlintstd < dtest.in
 Tests of the DOUBLE PRECISION LAPACK routines
 LAPACK VERSION 3.*.1

 The following parameter values will be used:
    M   :       0     1     2     3     5    10    50
    N   :       0     1     2     3     5    10    50
    NRHS:       1     2    15
    NB  :       1     3     3     3    20
    NX  :       1     0     5     9     1
    RANK:      30    50    90

 Routines pass computational tests if test ratio is less than   30.00

 Relative machine underflow is taken to be    0.222507-307
 Relative machine overflow  is taken to be    0.179769+309
 Relative machine precision is taken to be    0.111022D-15


 DGE routines passed the tests of the error exits

 All tests for DGE routines passed the threshold (   3653 tests run)

 DGE drivers passed the tests of the error exits

 All tests for DGE drivers  passed the threshold (   5748 tests run)

 DGB routines passed the tests of the error exits

 All tests for DGB routines passed the threshold (  28938 tests run)

 DGB drivers passed the tests of the error exits

 All tests for DGB drivers  passed the threshold (  36567 tests run)

 DGT routines passed the tests of the error exits

 All tests for DGT routines passed the threshold (   2694 tests run)
```

EIG/xeigtstd: Tester for real eigenvalue problems and singular value problems

Test for singular value decomposition

- How is singular value decomposition handled?
  - lapack-3.xx.y/TESTING/EIG/ddrvbd.f
- For the decomposition A=UΣV*, we first construct Σ.
  - This can be 0, I (identity matrix), a random matrix, scaled near overflow, or scaled near underflow.
- Then solve by some routines, and evaluate the following:
  - |A–Udiag(Σ) V*| / (|A|×max(M,N)×ulp)
  - |I–UU'| / (M×ulp)
  - |I–V* V| / (M×ulp)
  - etc
- All of these values should be sufficiently small. There are several routines available, such as DGESVD, DGESDD, DGESVDQ, DGESVDX, and so on.



EIG/xeigtstd: Tester for real eigenvalue problems and singular value problems

Test for singular value decomposition

- /EIG/xeigtstd < svd.in
- xeigstd: Quality assurance program for eigenvalue problems, singular value decomposition, etc., in the binary64 version.
- svd.in: Input file for quality assurance of singular value decomposition.
- We need to check such .in files for 23 (real) + 23 (complex) for a total of 46.

```
Tests of the Singular Value Decomposition routines

LAPACK VERSION 3.*.1

The following parameter values will be used:
   M:         0     0     0     1     1     1     2     2     3     3
              3    10    10    16    16    30    30    40    40
   N:         0     1     3     0     1     2     0     1     0     1
              3    10    16    10    16    30    40    30    40
   NB:        1     3     3     3    20
   NBMIN:     2     2     2     2     2
   NX:        1     0     5     9     1
   NS:        2     0     2     2     2

Relative machine underflow is taken to be    0.222507-307
Relative machine overflow  is taken to be    0.179769+309
Relative machine precision is taken to be    0.111022D-15

Routines pass computational tests if test ratio is less than   50.00


DBD routines passed the tests of the error exits ( 55 tests done)

DGESVD passed the tests of the error exits (  8 tests done)
DGESDD passed the tests of the error exits (  6 tests done)
DGEJSV passed the tests of the error exits ( 11 tests done)
DGESVDX passed the tests of the error exits ( 12 tests done)
DGESVDQ passed the tests of the error exits ( 11 tests done)


SVD:  NB =   1, NBMIN =   2, NX =   1, NRHS =   2

All tests for DBD routines passed the threshold (  10260 tests run)

All tests for DBD drivers  passed the threshold (  14820 tests run)


SVD:  NB =   3, NBMIN =   2, NX =   0, NRHS =   0
```

Problems with LAPACK testing.

- The LAPACK TESTING programs are meticulously made.
- Eigenvalue problems and singular value problems involve the concept of convergence, so in principle, they cannot be checked with algebraic test programs.
- Random inputs are used, but...
- Since LAPACK TESTING programs generate random numbers with a fixed seed, it's uncertain whether all cases are covered.
- When optimizing with compilers or implementing optimizations, the error can increase, or sometimes it doesn't pass the tests.
- Do the tests cover all possible inputs?
- Do they go through all the cases for every routine?
- The test matrix dimensions are relatively small, around 50x50.

MPLAPACK Quality assuance

After two years effort, we were able to build the
QA (test) program in all seven precisions.
Furthermore, we were able to pass the tests.

```
Cbak.in          Ced.in    Csb.in          Rbb.in    Rgd.in  glm.in  se2.in                xeigtstC__Float128  xeigtstC_mpfr       xeigtstR_doubl
Cbal.in          Cgbak.in  Csg.in          Rec.in    Rgg.in  gqr.in  sep.in                xeigtstC__Float64x  xeigtstC_qd         xeigtstR_gmp
Cbal_double.in   Cgbal.in  Rbak.in         Red.in    Rsb.in  gsv.in  svd.in                xeigtstC_dd         xeigtstR__Float128  xeigtstR_mpfr
Cbb.in           Cgd.in    Rbal.in         Rgbak.in  Rsg.in  lse.in  test_eig_all.sh       xeigtstC_double     xeigtstR__Float64x  xeigtstR_qd
Cec.in           Cgg.in    Rbal_double.in  Rgbal.in  csd.in  nep.in  test_eig_all_mingw.sh  xeigtstC_gmp        xeigtstR_dd
```

```
Ctest.in         test_lin_all_mingw.sh  xlintstC_gmp        xlintstR_dd      xlintstrfC__Float128  xlintstrfC_mpfr       xlintstrfR_double
Ctest_rfp.in     xlintstC__Float128     xlintstC_mpfr       xlintstR_double  xlintstrfC__Float64x  xlintstrfC_qd         xlintstrfR_gmp
Rtest.in         xlintstC__Float64x     xlintstC_qd         xlintstR_gmp     xlintstrfC_dd         xlintstrfR__Float128  xlintstrfR_mpfr
Rtest_rfp.in     xlintstC_dd            xlintstR__Float128  xlintstR_mpfr    xlintstrfC_double     xlintstrfR__Float64x  xlintstrfR_qd
test_lin_all.sh  xlintstC_double        xlintstR__Float64x  xlintstR_qd      xlintstrfC_gmp        xlintstrfR_dd
```

MPLAPACK QA scene

Linear routine QA scene with binary128

```
 Tests of the Multiple precision version of LAPACK MPLAPACK VERSION 2.0
 Based on the original LAPACK VERSION 3.10.1

The following parameter values will be used:
       M  :     0     1     2     3     5    10    50
       N  :     0     1     2     3     5    10    50
    NRHS  :     1     2    15
      NB  :     1     3     3     3    20
      NX  :     1     0     5     9     1
    RANK  :    30    50    90

 Routines pass computational tests if test ratio is less than   30.00

 Relative machine underflow is taken to be : +3.3621031431120935e-4932
 Relative machine overflow  is taken to be : +1.1897314953572318e+4932
 Relative machine precision is taken to be : +1.9259299443872359e-34
 RGE routines passed the tests of the error exits

 All tests for RGE routines passed the threshold (   3653 tests run)
 RGE drivers passed the tests of the error exits

 All tests for RGE drivers  passed the threshold (   5748 tests run)
 RGB routines passed the tests of the error exits
```

SVD QA scene with MPFR

```
Tests of the Singular Value Decomposition routines
Tests of the Multiple precision version of LAPACK MPLAPACK VERSION 2.0.1
Based on original LAPACK VERSION 3.10.1

The following parameter values will be used:
   M:         0     0     0     1     1     1     2     2     3     3
              3    10    10    16    16    30    30    40    40
   N:         0     1     3     0     1     2     0     1     0     1
              3    10    16    10    16    30    40    30    40
   NB:        1     3     3     3    20
   NBMIN:     2     2     2     2     2
   NX:        1     0     5     9     1
   NS:        2     0     2     2     2

 Relative machine underflow is taken to be+9.5302596195518043e-323228497
 Relative machine overflow  is taken to be+2.0985787164673877e+323228496
 Relative machine precision is taken to be+7.4583407312002067e-155

 Routines pass computational tests if test ratio is less than+5.0000000000000000e+01

 ZBD routines passed the tests of the error exits ( 35 tests done)
 Cgesvd passed the tests of the error exits (  8 tests done)
 Cgesdd passed the tests of the error exits (  6 tests done)
 Cgejsv passed the tests of the error exits ( 11 tests done)
 Cgesvdx passed the tests of the error exits ( 12 tests done)
 Cgesvdq passed the tests of the error exits ( 11 tests done)


 SVD:  NB =   1, NBMIN =   2, NX =   1, NRHS =   2

 All tests for CBD routines passed the threshold (   4085 tests run)
```

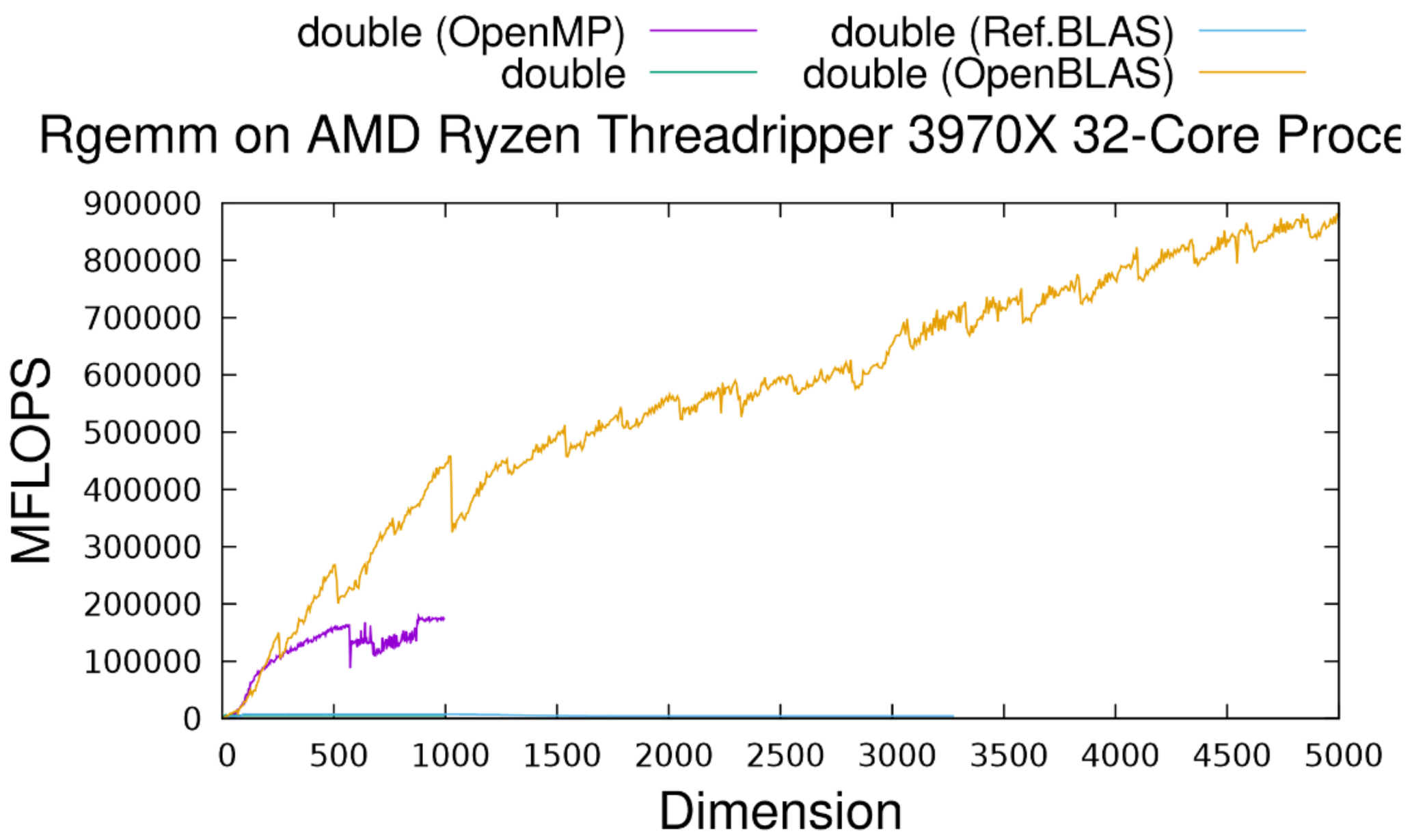
- Most of the tests pass.
- Not all consistently pass the tests
  - We're seriously generating random numbers.
- It takes an incredible amount of time!!
  - MP calculations are 100x slower and we have to pass the QA for all seven precision types!

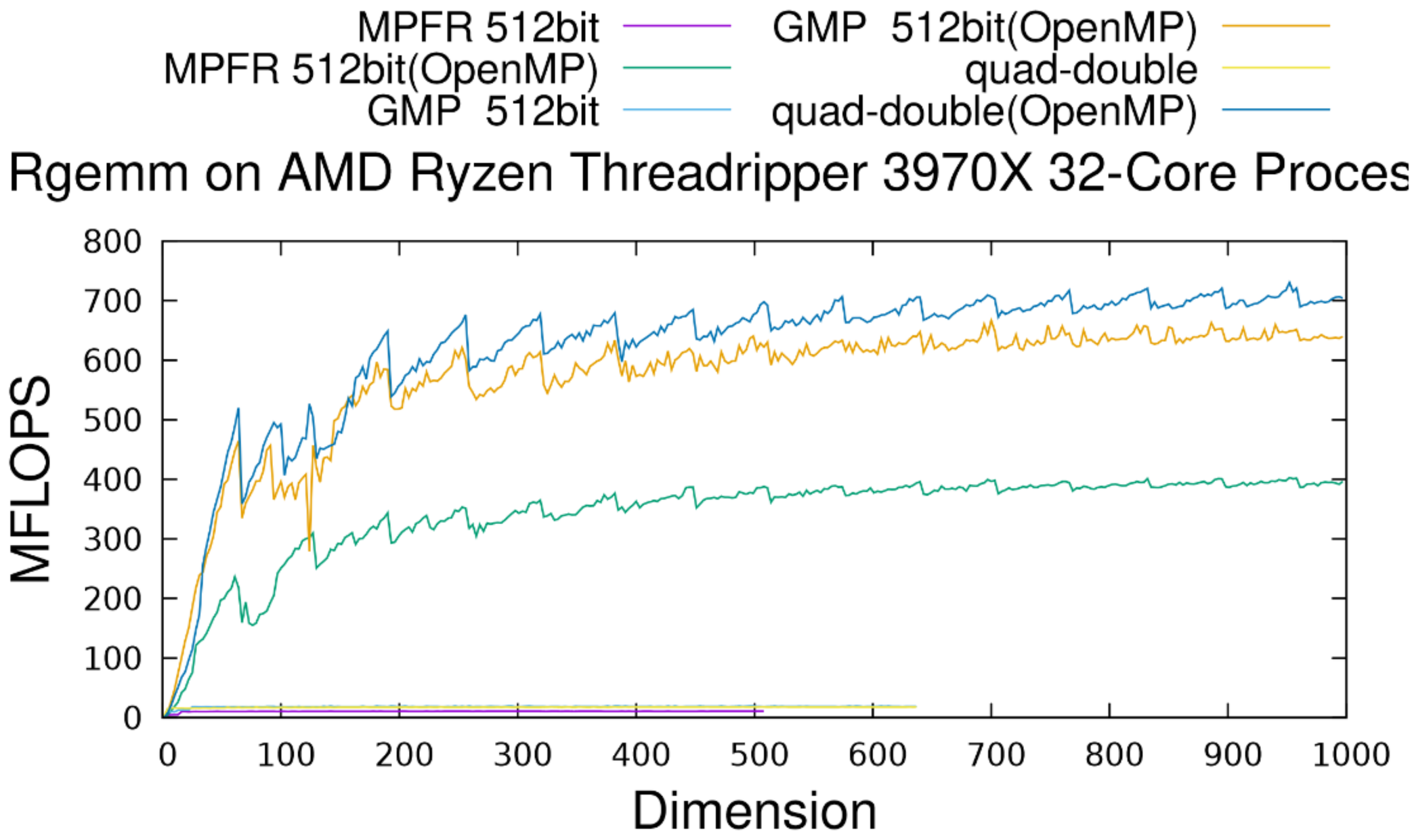
https://github.com/nakatamaho/mplapack/tree/master/mplapack/test/lin/results

https://github.com/nakatamaho/mplapack/tree/master/mplapack/test/eig/results

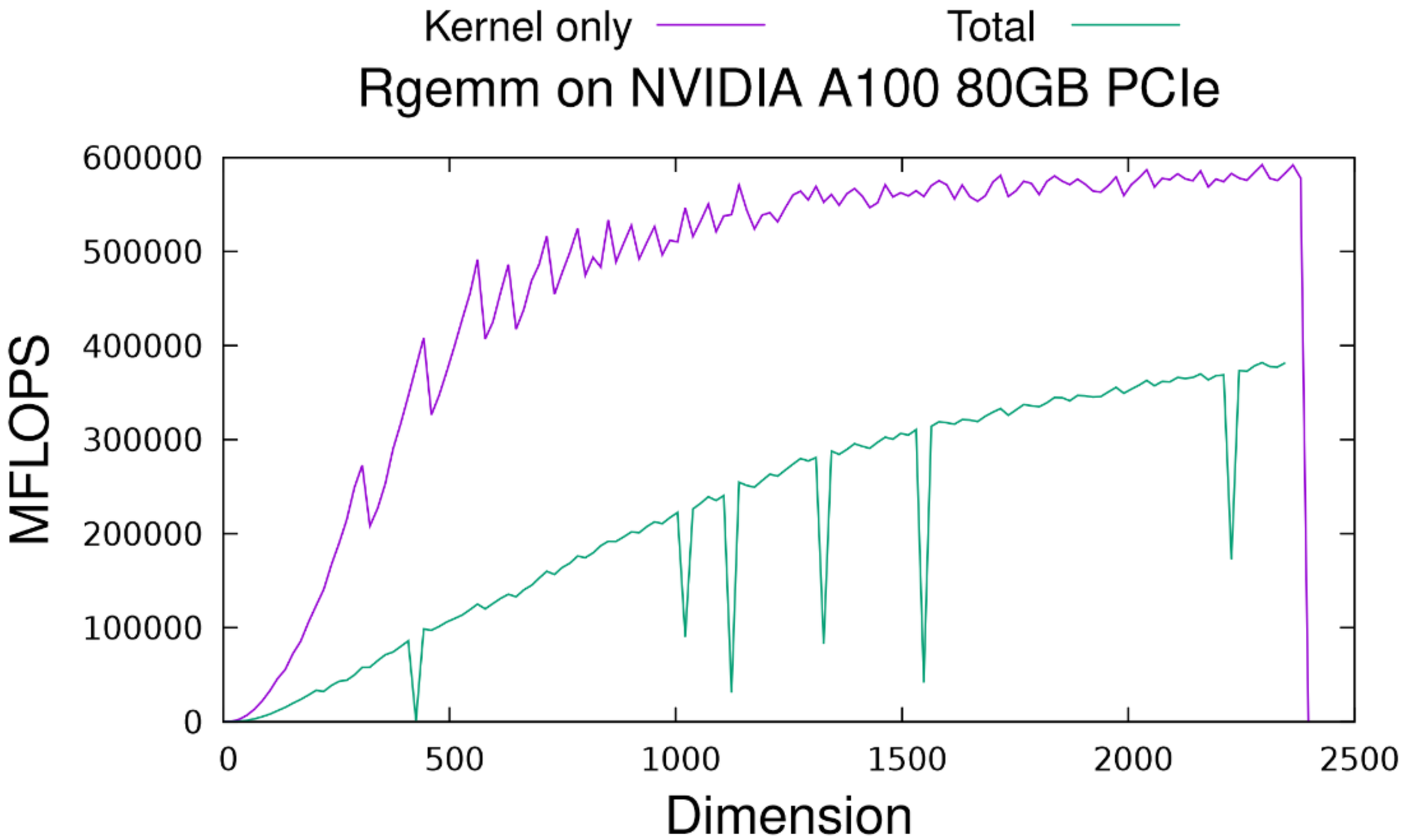
Benchmarks

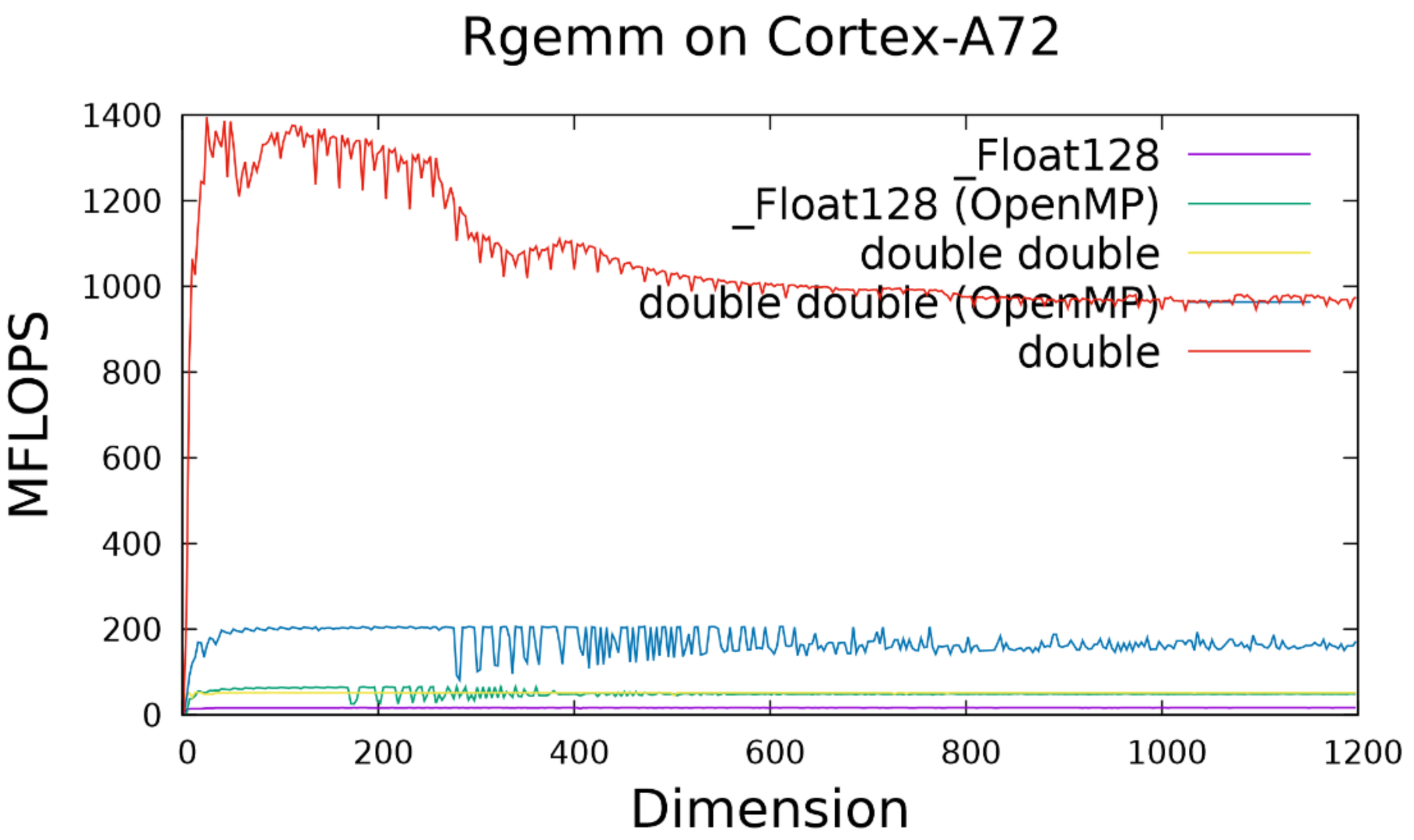
https://github.com/nakatamaho/mplapack/tree/master/benchmark/results

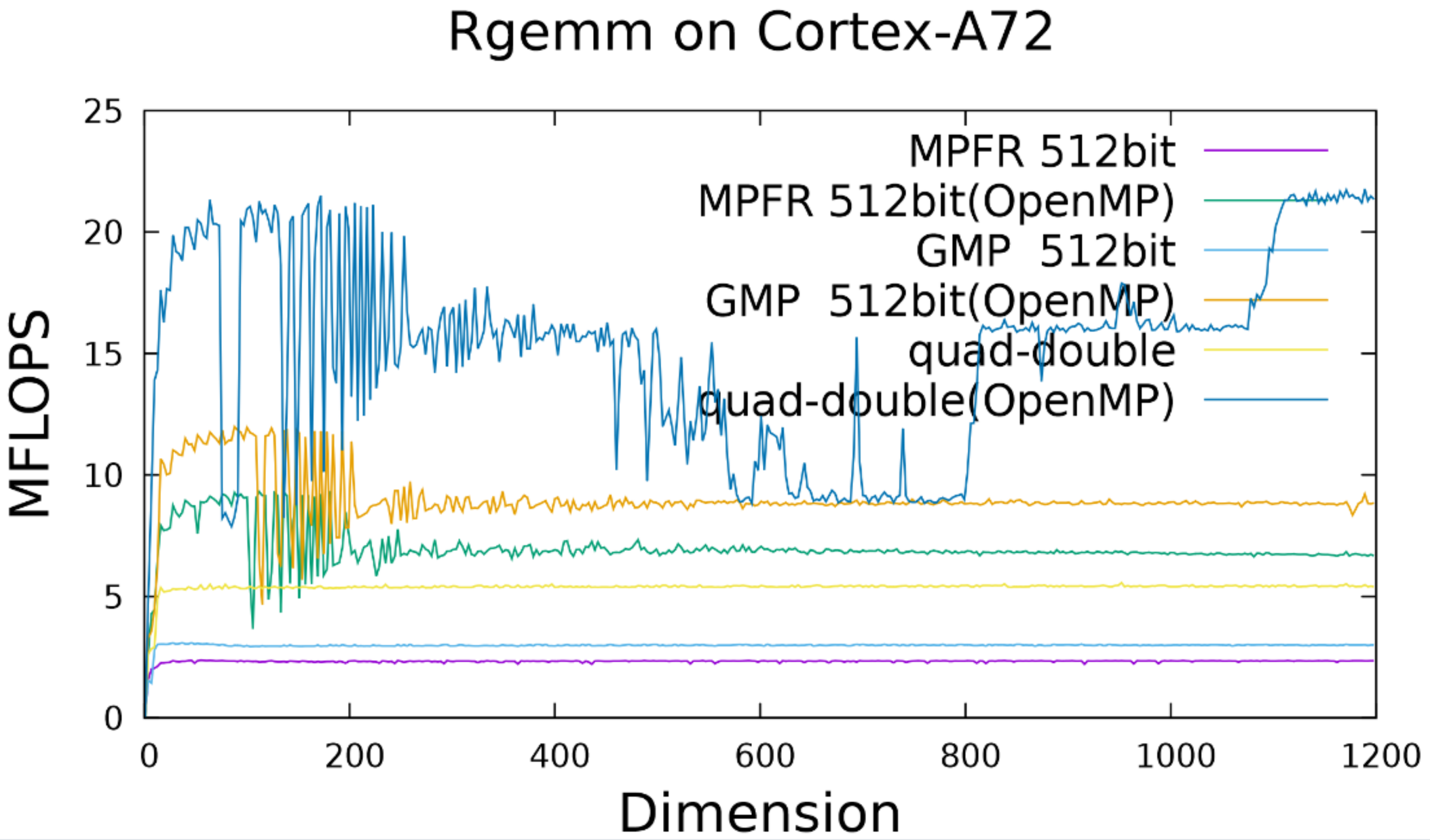

double (OpenMP)
double
double (Ref.BLAS)
double (OpenBLAS)
Rgemm on AMD Ryzen Threadripper 3970X 32-Core Proce
MFLOPS
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Dimension


MPFR 512bit
MPFR 512bit(OpenMP)
GMP 512bit
GMP 512bit(OpenMP)
quad-double
quad-double(OpenMP)
Rgemm on AMD Ryzen Threadripper 3970X 32-Core Proces
MFLOPS
Dimension
0
100
200
300
400
500
600
700
800
900
1000

Rgemm double-double




Rgemm on Cortex-A72
MFLOPS
Dimension
0
200
400
600
800
1000
1200
1400
_Float128
_Float128 (OpenMP)
double double
double double (OpenMP)
double


Rgemm on Cortex-A72
MFLOPS
Dimension
0
5
10
15
20
25
0
200
400
600
800
1000
1200
MPFR 512bit
MPFR 512bit(OpenMP)
GMP 512bit
GMP 512bit(OpenMP)
quad-double
quad-double(OpenMP)

Conclusion & Future Directions

- Introduction of MPLAPACK 2.0.1
  - API definition in C++
  - Full implementation
  - Simplified migration process from LAPACK
- Future Plans
  - Fortran90 interface (?)
  - Further optimizations and speed enhancements
  - Implementation of mixed-precision routines
  - Utilization of templates
  - Integration with Python and Octave

What is the MPLAPACK 2.0.1?

- Overview: MPLAPACK is a multiple precision version of BLAS and LAPACK.
- Precision Support: Compatible with GMP, MPFR, binary128, quad-double, double-double, double, and binary80.
- API & Language: Provides a C++ API based on LAPACK 3.9.1. Users can utilize MP floating point numbers similarly to 'double'. Entirely re-written from Fortran90 to C++.
- Implementation: All routines are fully implemented, excluding mixed precision routines.
- Functionality: Supports both real and complex numbers, eigenvalue computations, singular value decompositions, least square fits and more!
- Compatibility: Works seamlessly on Linux, Mac, and Windows platforms.
- Performance: The double-double version of Rgemm achieves a remarkable 600 GFlops on A100
- Licensing: Distributed under the 2-clause BSD license.

https://github.com/nakatamaho/mplapack