Science & Technology Facilities Council

Technical Report
RAL-TR-2016-014

# Investigation into the mixed precision linear solver HSL_MA79

HS Thorne

October 2016

Enquiries concerning this report should be addressed to:

RAL Library
STFC Rutherford Appleton Laboratory
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446403
email: libraryral@stfc.ac.uk

Science and Technology Facilities Council reports are available online at: http://epubs.stfc.ac.uk

**ISSN 1358-6254**

# Investigation into the mixed precision linear solver HSL_MA79

H. Sue Thorne[1,2]

## ABSTRACT

In software development, a large amount of work is done to ensure that the resulting code is *efficient*, where the wall clock execution time is the normal measure for efficiency. In the future, there will be an increasing desire for codes to be *energy efficient*, that is, the amount of energy consumed whilst the code is run is minimised.

The use of mixed precision is one proposed methodology for reducing both wall clock execution times and energy consumption of some codes. In this report, we consider the HSL code HSL_MA79, which is a mixed precision code for solving linear systems of the form $Ax = b$, where $x$ is the unknown. We compare execution times and energy consumption of the code for a number of different matrices $A$ in a bid to reveal whether there are classes of sparse, symmetric matrix problems where the mixed precision approach may be advantageous. In particular, we will consider the application area that the matrix is from.

This work forms part of the EPSRC Service Level Agreement funded Software Outlook project.

**Keywords:** Energy Efficient Computing, Intel Xeon (IvyBridge), Linear Solver, Single Precision, Double Precision, Mixed Precision, CCP

September 8, 2016

[1]Scientific Computing Department, Science and Technology Facilities Council, Rutherford Appleton Laboratory, Harwell Campus, Didcot, OX11 0QX, UK.

[2]Correspondence to: sue.thorne@stfc.ac.uk

# 1 Background and motivation

The desire for scientific codes to be *efficient* is governed both by wanting to minimise the execution time and, increasingly, trying to reduce the amount of energy consumed whilst running the code. With the move from petascale computing to exascale computing on the horizon and the associated requirement to increase the floating-point operation rate by a factor of 100 whilst only increasing the power usage by a factor of 10, *energy efficient computing* is raising its head in importance. Additionally, some HPC service providers are proposing to base their service fees on the amount of energy consumed by the user rather than a metric purely base on the time and number of cores used.

In this report, we consider the use of a mixed precision approach to solving linear systems of the form $Ax = b$, where $A$ is a real, sparse, symmetric matrix with $n$ rows and columns, and $nz$ non-zero entries ($nz \ll n^2$), $b$ is a real vector of length $n$ and $x$ is the vecotr we wish to compute. Such matrix problems arise in a large number of applications and it is not unusual to need to solve a sequence of similar yet slightly different linear systems. Our investigations will centre around the `HSL` [10] code `HSL_MA79` [9], which is a mixed precision solver for problems of the form $AX = B$, where $A$ is as above but $X$ and $B$ can have more than one column.

In Section 2, we outline the mixed precision solver `HSl_MA79` and we outline our benchmark tests in Section 3. The results of our tests can be found in Section 4, with our conclusions in Section 5.

# 2 Mixed precision solver `HSL_MA79`

When solving $Ax = b$, `HSL_MA79`'s default method for measuring the accuracy of the computed $x$ is the scaled norm

$$\beta = \frac{\|Ax - b\|_\infty}{\|A\|_\infty \|x\|_\infty + \|b\|_\infty}. \tag{1}$$

The smaller the value of $\beta$ for fixed $A$ and $b$, the more accurate the computed value of $x$ is with respect to this scaled norm.

An outline of the method used within `HSL_MA79` is given in Algorithm 1. The user sets the precision of the initial factorisation, `prec`, the choice of linear solver, `solver`, and `accuracy`, the value that $\beta$ should not exceed for the method to successfully terminate.

`HSL_MA79` incorporates two different linear solvers for the factorisation of sparse, symmetric matrices: `MA57` and `HSL_MA77`. `MA57` is a serial, in-core code whilst `HSL_MA77` is a multi-threaded and out-of-core code. Given the nature of our test matrices (Section 3) and the fact that we are concerned with the effect of using mixed precision over that of using just single or double precision, we chose to only use `MA57`.

Iterative refinement is a cheap iterative method that takes an approximation to the solution $x$ and aims to incrementally improve the solution. Whilst each iteration is cheap, it may be slow to converge. If iterative refinement is not converging very quickly, `HSL_MA79` switches to using the iterative method `FGMRES`, which, on average, is a lot more expensive than iterative refinement with respect to the cost of each iteration but it normally has better convergence properties. Further details about the methods used within `HSL_MA79` can be found in [9]. By default, `HSL_MA79` will switch to using a double precision factorisation if the required level of accuracy is not reached: in these tests, we have turned off this default setting by setting `fallback_double` = `.FALSE.`. All other default controls were used.

# 3 Benchmark tests

In Table 1, we list the matrices considered in our tests. The matrices were all obtained from the University of Florida Sparse Matrix Collection [5] and have been grouped according to their classification within [5]. For our test set, we have chosen real, symmetric problems were the number of rows/columns, $n$, are fairly modest and the matrices are all sparse, that is, $nz \ll n^2$. We always set the right-hand side, $b$, to be a vector of length $n$ with entries all equal to 1. For each matrix, we compare four different methods:

**Algorithm 1** `HSL_MA79` outline

```
Analyse matrix A using MA57
Choose solver and initialise precision prec
loop
  Factor A using solver with prec precision
  Use computed factors to (approximately solve Ax = b
  Compute scaled residual β
  if β ≤ accuracy then
    return x
  end if
  Apply iterative refinement and recompute β
  if β ≤ accuracy then
    return x
  end if
  Apply FGMRES and recompute β
  if prec = SINGLE and fallback_double then
    prec = DOUBLE
  else
    Return x with warning
  end if
end loop
```

`S` Factor $A$ using single precision and use `accuracy` $= 2 \times 10^5$ (i.e., no iterative refinement or FGMRES);

`D` Factor $A$ using double precision and use `accuracy` $= 2 \times 10^5$ (i.e., no iterative refinement or FGMRES);

`M1` Factor $A$ using single precision and use `accuracy` $= 5 \times 10^{-15}$;

`M2` Factor $A$ using single precision and use `accuracy` $= 5 \times 10^{-12}$.

For each matrix and method, we report

$t_f$, the time to factor $A$ (only reported for Methods `S` and `D`);

$t_{afs}$, the time to analyse $A$, factor $A$ and compute $x$ to desired level of accuracy;

$E_{tot}$, the total energy consumption to read in $A$, analyse and factor $A$, and compute $x$;

$\beta$, the scaled norm (1) for the computed $x$;

$ItRef$, the number of iterative refinement iterations performed;

$ItFG$, the number of `FGMRES` iterations performed.

All of the tests were run on Neale, a system provided by STFC's Hartree Centre [1]. Neale contains Intel Xeon (IvyBridge ES-2650v2) based nodes: each node has 8 cores (16 threads), the clock speed is 2.6GHz (3.6 GHz Max) and the L1 and L2 Cache sizes are 32 kB and 256 kB, respectively. Each node has 64 GB of memory. Energy monitoring on Neale uses the PAPI (Performance API) interface to begin an energy monitoring thread on an individual node while the test runs. Results consist of a trace of total power consumption from the A2 node and memory domains and the energy consumption is approximated from this power trace.

In all of the figures, the test problems have been ordered in the same order as in Table 1 and vertical lines have been used to separate different classes of problems.

| Class | Name | $n$ | $nz$ |
|---|---|---|---|
| Structural | apache1 | 80800 | 542184 |
| | apache2 | 715176 | 4817870 |
| | bcsstk36 | 23052 | 1143140 |
| | bmw3_2 | 227362 | 11288630 |
| | bmw7st_1 | 141347 | 7339667 |
| | crankseg_1 | 52804 | 10614210 |
| | ct20stif | 52329 | 2698463 |
| | dawson5 | 51537 | 1010777 |
| | F2 | 71505 | 5294285 |
| | hood | 220542 | 9895422 |
| | msc23052 | 23052 | 1154814 |
| | nasasrb | 54870 | 2677324 |
| | oilpan | 73752 | 3597188 |
| | pwtk | 217918 | 11634424 |
| | s3dkq4m2 | 90449 | 4820891 |
| | s4dkt3m2 | 90449 | 3753461 |
| | ship_003 | 121728 | 8086034 |
| | shipsec8 | 114919 | 6653399 |
| | thread | 29736 | 4470048 |
| | vanbody | 47072 | 2329056 |
| Network | case39 | 40216 | 1042160 |
| | TSOPF_FS_b39_c30 | 120216 | 3121160 |
| Model order reduction | boneS01 | 127224 | 6715152 |
| | filter3D | 106437 | 2707179 |
| | rail_79841 | 79841 | 553921 |
| | t3dl | 20360 | 509866 |
| Comp. fluid dynamics | cfd1 | 70656 | 1828364 |
| | cfd2 | 123440 | 3087898 |
| | copter2 | 55476 | 759952 |
| | parabolic_fem | 525825 | 3674625 |
| | shallow_water1 | 81920 | 327680 |
| | stokes128 | 49666 | 558594 |
| Optimisation | a0nsdsil | 80016 | 355034 |

| Class | Name | $n$ | $nz$ |
|---|---|---|---|
| Optimisation | a5esindl | 60008 | 255004 |
| | blockqp1 | 60012 | 640033 |
| | boyd2 | 466316 | 1500397 |
| | brainpc2 | 27607 | 179395 |
| | c-56 | 35910 | 380900 |
| | cont-201 | 80595 | 438795 |
| | cont-300 | 180895 | 988195 |
| | dixmaanl | 60000 | 299998 |
| | jnlbrng1 | 40000 | 19920 |
| | k1_san | 67759 | 559774 |
| | ncvxbqp1 | 50000 | 349968 |
| | torsion1 | 40000 | 197608 |
| Circuit sim. | G2_circuit | 150102 | 726674 |
| Economics | finan512 | 74752 | 596992 |
| Comp. graphics | Andrews | 60000 | 760154 |
| Acoustics | qa8fk | 66127 | 1660579 |
| Thermal | thermal1 | 82654 | 574458 |
| | thermomech_dM | 204316 | 1423116 |
| Electromagn. | tmt_sym | 726713 | 5080961 |
| 2D/3D | aug3dcqp | 35543 | 128115 |
| | cant | 62451 | 4007383 |
| | d_pretok | 182730 | 1641672 |
| | darcy003 | 389874 | 2101242 |
| | Dubcova2 | 65025 | 1030225 |
| | Dubcova3 | 146689 | 3636649 |
| | ecology1 | 1000000 | 4996000 |
| | ecology2 | 999999 | 4995991 |
| | helm2d03 | 392257 | 2741935 |
| | helm3d01 | 32226 | 428444 |
| | tuma1 | 22967 | 87760 |
| | turon_m | 189924 | 1690876 |
| | wathen120 | 36441 | 565761 |
| Materials | crystk03 | 24696 | 1751178 |

Table 1: The test matrices considered in our tests along with the number of rows/columns, $n$, and the number of non-zero entries, $nz$. The class of each matrix is also given.

# 4 Results and discussion

In Table 2 (at the end of the report), we list the values of $t_{afs}$, $E_{tot}$, $\beta$, $ItRef$ and $ItFG$ for each test problem and method. In Figure 1, we compare the value of $t_{afs}$ for Methods `S` and `D`. As expected, using single precision is faster than double precision for most test problems. For those classed as structural problems, there is between a 13 and 30% reduction in $t_f$ but, as seen in Figure 2, the value of $\beta$ is roughly $10^8$ times larger. With regards the optimisation problems, for 7 out of the 13 problems there is no difference in the value of $t_{afs}$ but the value of $\beta$ is about $10^8$ times larger when single precision is used. It should also be noted that the values of $\beta$ from Method `D` for 5 of the optimisation problems were also several orders of magnitude larger than that of most other problems in our large test set, which indicates the difficult nature of these problems.

Factorising $A$ using double precision and computing $x$ using these factors resulted in $\beta$ being larger than $5 \times 10^{-15}$ for 11 out of the 65 test problems. For all of these 11 test problems, Method `M1` computed an $x$ for which $\beta$ was smaller than $5 \times 10^{-15}$ and, hence, the mixed precision methodology produced a more accurate value of $x$ for these problems. The most extreme case was the optimisation test problem k1_san, Method `D` resulted in $\beta = 6.5 \times 10^{-7}$ whilst Method `M1` achieved $\beta < 5 \times 10^{-15}$ with 1 iterative refinement iteration and 12

FGMRES iterations. Thus, even without time and energy consumption considerations, a mixed precision approach may be advantageous over a double precision factorisation. This also makes it an attractive approach for when higher than double precision accuracy is required but the factorisation can only be done using double precision arithmetic.

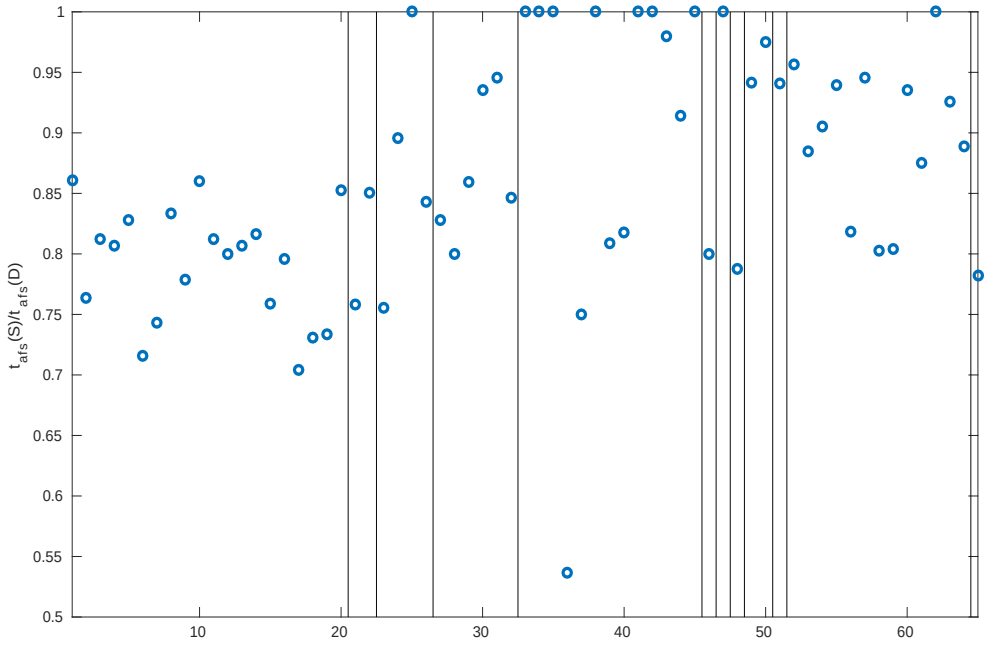


Figure 1: Value of $t_{afs}(\texttt{S})/t_{afs}(\texttt{D})$.

In Figures 3 and 4, we compare the execution times and energy consumption values for Methods D and M1. For the 11 problems where Method D failed to calculate an $x$ satisfying $\beta < 5 \times 10^{-15}$, Method M1 took slightly less time for 3 of the problems, the same amount of time for 2 of the problems and between 15 and 45% longer for the remaining 6 problems. If it is important that $\beta < 5 \times 10^{-15}$, then this increase in time will generally be acceptable. The energy consumption ratios are slightly larger than those of the execution times but, again, the values are almost certainly within an acceptable level. Method M1 failed to compute an $x$ satisfying $\beta < 5 \times 10^{-15}$ for 9 of the problems. For 8 of these problems, the method had longer execution times than when Method D was used: this is due to the large number of FGMRES iterations. For our two network problems, Method M1 failed to reach the desired level of accuracy with the execution times and energy consumption values increased by at least 45% and, hence, if the method had been allowed to fallback to using double precision, then the mixed precision method would take roughly 2.5 times the time and energy to calculate an acceptable $x$. This highlights that, if very high accuracy is required, then there may be some classes of problems where it is best to stay with using double precision factorisations. For the computational fluid dynamics problems, there was always an advantage to using Method M1 over Method D with respect to execution time; there was little difference when energy consumption is considered. For the 12 optimisation problems, Method M1 failed to reach the desired accuracy for one problem but, for the 6 problems where Method D performed well, there was little difference in time and energy consumption for 4 of the problems. Certainly, when high accuracy is required for linear systems from optimisation, mixed precision should be seriously considered. For the 13 2D/3D problems, the execution times differ by up to 20% (increase and decrease) but Method M1 achieved the required accuracy for the 3 problems where Method D failed; there is normally a slight increase in energy consumption. In general, the results show

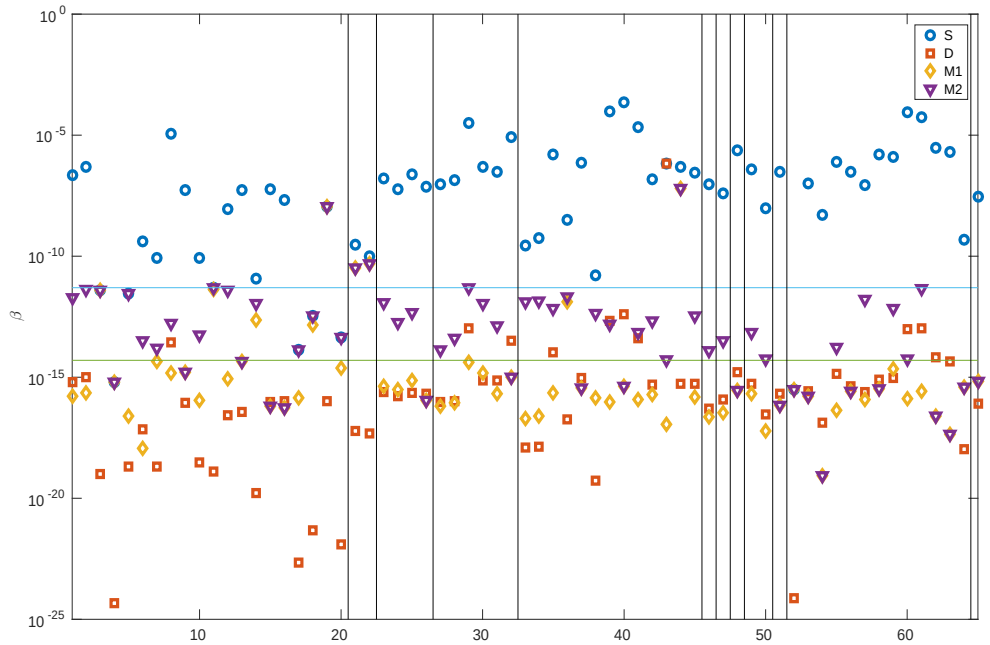


Figure 2: Value of $\beta$ for Methods `S`, `D`, `M1` and `M2`.

that, when high accuracy is required, the mixed precision approach may be of use and it is worth investigating whether it should be used for your particular application.

By the nature of some applications and the methods used to formulate the linear system, finding an $x$ that satisfies $\beta < 5 \times 10^{-15}$ is often overkill because the data involved has much lower levels of accuracy. If, instead, we require that $\beta < 5 \times 10^{-12}$, then we will expect the number of iterations to decrease and, hence, for execution times and energy consumption values to drop. In Figures 5 and 6, we observe that the execution times and energy consumption values are similar for the vast majority of problems but Method `M2` only fails for four of the problems: again, there are failures for the two network problems. For the four structural problems where Method `M1` fails but Method `M2` is successful, the execution times for Method `M2` are between 43 and 62% lower than Method `M2` because the number of `FGMRES` iterations have dramatically dropped; the energy consumption values have dropped by between 23 and 46%.

In Figures 7 and 8, we compare Methods `D` and `M2`. Method `D` on failed to the reach the desired accuracy of $\beta < 5 \times 10^{-12}$ for one problem (k1_san). Of the 65 test problems, Method `M2` is successful for 61 of the problems and, of these problems, it has a negative impact on the execution times for just 16 of the problems (we consider there to be no negative impact for the problem that Method `D` failed on); the ratios for energy consumption are slightly larger. Hence, when lower levels of accuracy for are required for the solution of linear, mixed precision methods are even more attractive, particularly for structural, model order reduction and computational fluid dynamics problems.

# 5 Conclusions

We have seen that if a linear system is being solved by factoring the matrix using double precision accuracy, then switching to a mixed precision approach may be beneficial, particularly for execution times. For problems where the solution needs calculating to a very high accuracy but the double precision factorisation does not compute

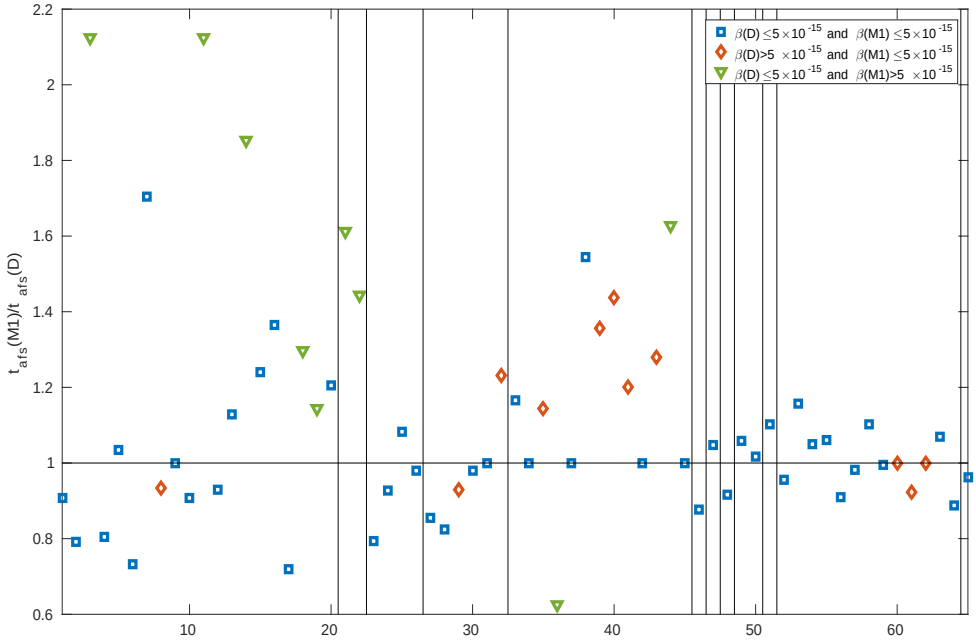


Figure 3: Value of $t_{afs}(\texttt{M1})/t_{afs}(\texttt{D})$.

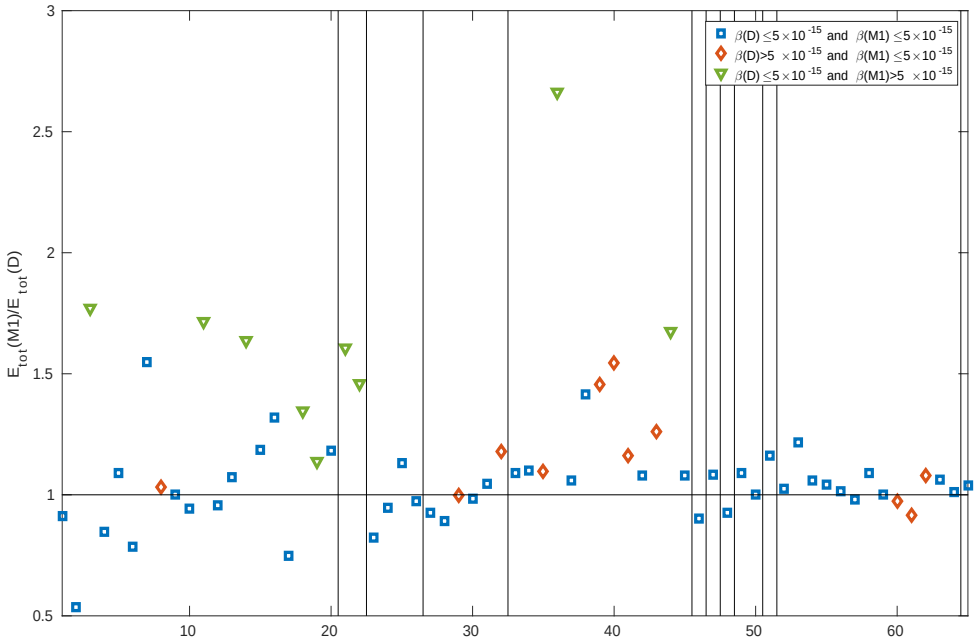


Figure 4: Value of $E_{tot}(\texttt{M1})/E_{tot}(\texttt{D})$.

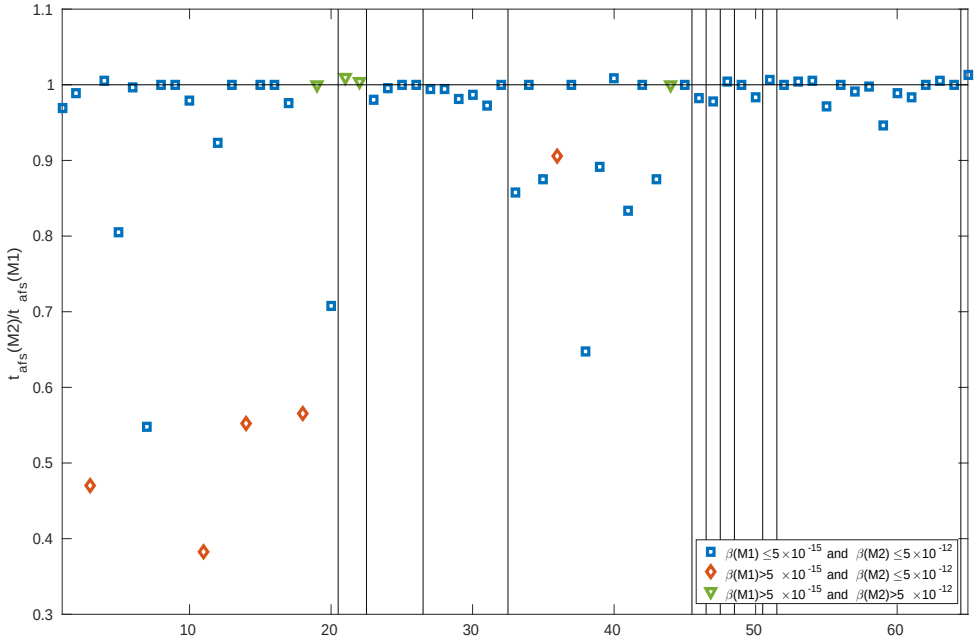


Figure 5: Value of $t_{afs}(\texttt{M2})/t_{afs}(\texttt{M1})$.

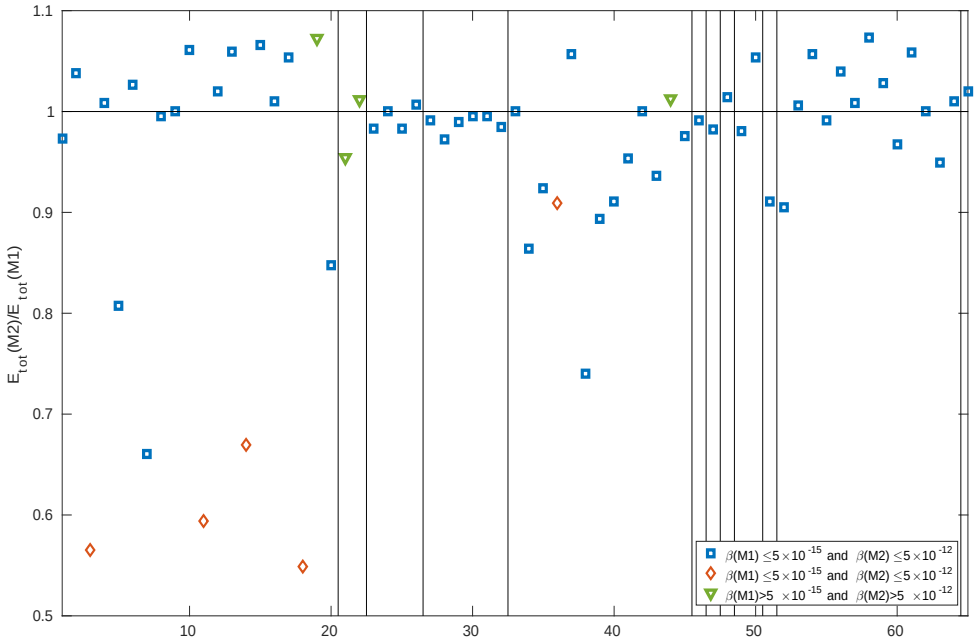


Figure 6: Value of $E_{tot}(\texttt{M2})/E_{tot}(\texttt{M1})$.

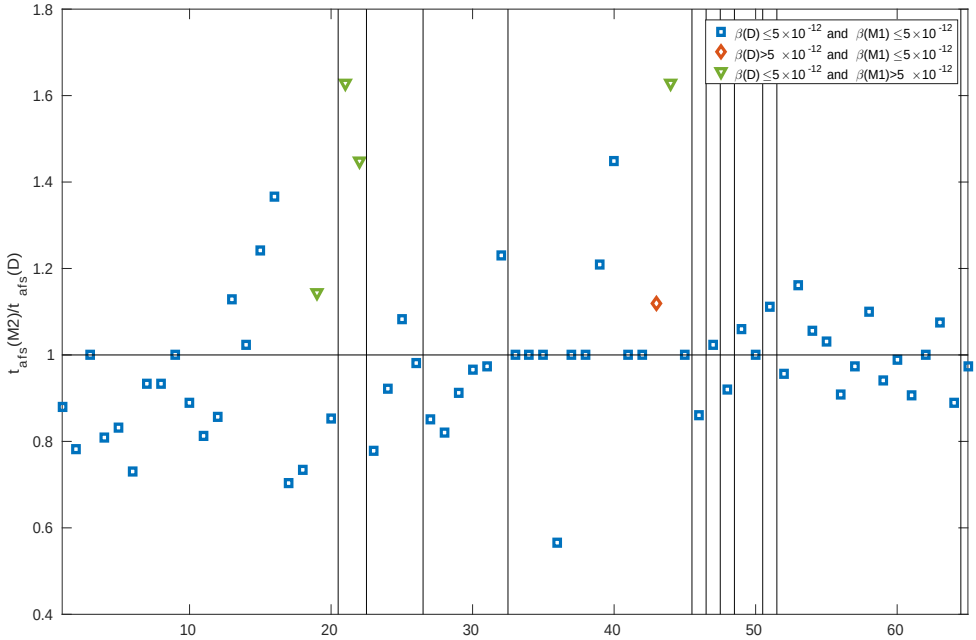


Figure 7: Value of $t_{afs}(\texttt{M2})/t_{afs}(\texttt{D})$.

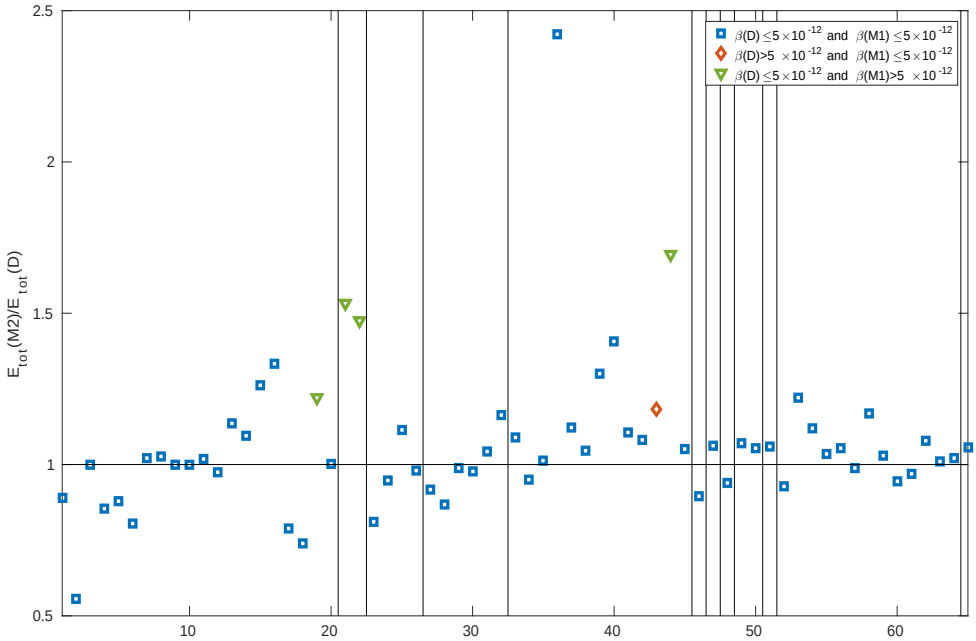


Figure 8: Value of $E_{tot}(\texttt{M2})/E_{tot}(\texttt{D})$.

a good enough solution, the mixed precision approach is very attractive. When a lower level of accuracy is required, the mixed precision approach becomes even more attractive for some problems. For some problems, a double precision factorisation may not be possible due to memory restrictions but a single precision factorisation is possible and the mixed precision approach could then be used. We should also note that iterative refinement did not converge quickly enough and `FGMRES` had to be used for many of the problems. `FGMRES` can be a very expensive method and cheaper Krylov-based methods may be available for certain classes of matrix, for example, the preconditioned conjugate gradient method for symmetric, sparse, positive definite matrices, which has well understood convergence properties [12]. For problems where a factorisation of $A$ is not possible in either double or single precision due to memory or time constraints, there are a number of iterative method possibilities along with ways to improve the rate of convergence, for example, the use of preconditioners [6, 7, 12].

Below we list some of the available sparse linear solvers that contain methods for refining the solution $x$.

**HSL_MA79** Either in-core and serial, or out-of-core and multi-threaded. Single and double precision versions available with iterative refinement and `FGMRES`

**MUMPS** Parallel code (MPI and OpenMP) for symmetric and unsymmetric problems. Single and double precision versions available with iterative refinement [2]

**PaStiX** Parallel code (MPI and OpenMP) for symmetric and unsymmetric problems. Single precision factorisation NOT available but has a number of iterative refinement methods available to improve accuracy of the solution [8, 3]

**SuperLU** Parallel code (MPI and OpenMP) for unsymmetric with single and double precision versions available with iterative refinement [11, 4].

The ideal linear solver would be an in-core parallel code that has both single and double precision versions and methods for refining $x$ that include Krylov-subspace iterative methods. Such a code does not seem to be publicly available at the moment.

# References


[1] See http://www.hartree.stfc.ac.uk/hartree/.

[2] See `http://mumps.enseeiht.fr/index.php?page=home`.

[3] See `http://pastix.gforge.inria.fr/`.

[4] See `http://http://crd-legacy.lbl.gov/~xiaoye/SuperLU/#superlu_mt`.

[5] T. Davis, *The University of Florida Sparse Matrix Collection*, 2007. http://www.cise.ufl.edu/davis/techreports/matrices.pdf.

[6] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*, Oxford University Press, Oxford, 2005.

[7] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, third ed., 1996.

[8] P. Hénon, P. Ramet, and J. Roman, *PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems*, Parallel Computing, 28 (2002), pp. 301–321.

[9] J. D. Hogg and J. A. Scott, *A fast and robust mixed-precision solver for the solution of sparse symmetric linear systems*, ACM Trans. Math. Softw., 37 (2010), pp. 17:1–17:24.

[10] HSL, *A collection of Fortran codes for large-scale scientific computation.* `http://www.hsl.rl.ac.uk/`, 2013.

[11] X. S. Li, *An overview of SuperLU: Algorithms, implementation, and user interface*, ACM Trans. Math. Softw., 31 (2005), pp. 302–325.

[12] Y. Saad, *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second ed., 2003.

Table 2: Time to factor $A$, $t_f$; time to analyse $A$, factor $A$ and compute $x$ to desired level of accuracy, $t_{afs}$; total energy consumption to read in $A$, analyse and factor $A$, and compute $x$, $E_{tot}$; the scaled norm (1) for the computed $x$, $\beta$; number of iterative refinement iterations and FGMRES iterations performed, $ItRef$ and $ItFG$, respectively.

| | $t_f$ (s) | | $t_{afs}$ (s) | | | | $E_{tot}$ (kJ) | | | | $\beta$ | | | | $ItRef$ | | $ItFG$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | S | D | S | D | M1 | M2 | S | D | M1 | M2 | S | D | M1 | M2 | M1 | M2 | M1 | M2 |
| apache1 | 0.47 | 0.61 | 0.93 | 1.08 | 0.98 | 0.95 | 0.0502 | 0.0577 | 0.0527 | 0.0513 | 2.3043e-07 | 6.2802e-16 | 1.6923e-16 | 1.9011e-12 | 4 | 2 | 0 | 0 |
| apache2 | 10.89 | 13.36 | 15.60 | 20.43 | 16.17 | 15.99 | 0.814 | 1.580 | 0.846 | 0.878 | 4.7828e-07 | 9.8703e-16 | 2.2223e-16 | 4.3381e-12 | 4 | 2 | 0 | 0 |
| bcsstk36 | 0.09 | 0.11 | 0.13 | 0.16 | 0.34 | 0.16 | 0.0144 | 0.0148 | 0.0262 | 0.0148 | 3.8931e-12 | 9.9955e-20 | 3.8931e-12 | 3.8931e-12 | 1 | 0 | 28 | 0 |
| bmw3_2 | 2.37 | 3.21 | 3.63 | 4.50 | 3.62 | 3.64 | 0.244 | 0.288 | 0.244 | 0.246 | 6.1978e-16 | 4.7241e-25 | 6.1978e-16 | 6.1978e-16 | 0 | 0 | 0 | 0 |
| bmw7st_1 | 1.03 | 1.37 | 1.73 | 2.09 | 2.16 | 1.74 | 0.159 | 0.167 | 0.182 | 0.147 | 2.8843e-12 | 2.0451e-19 | 2.4077e-17 | 2.8843e-12 | 1 | 0 | 4 | 0 |
| crankseg_1 | 2.84 | 4.13 | 3.30 | 4.61 | 3.38 | 3.37 | 0.223 | 0.289 | 0.227 | 0.233 | 4.0596e-10 | 7.2813e-18 | 1.1989e-18 | 3.1028e-14 | 2 | 1 | 0 | 0 |
| ct20stif | 0.66 | 0.93 | 0.78 | 1.05 | 1.79 | 0.98 | 0.0597 | 0.0698 | 0.108 | 0.0713 | 8.5320e-11 | 2.0592e-19 | 4.3354e-15 | 1.5450e-14 | 1 | 1 | 32 | 4 |
| dawson5 | 0.15 | 0.20 | 0.25 | 0.30 | 0.28 | 0.28 | 0.0188 | 0.0216 | 0.0223 | 0.0222 | 1.1317e-05 | 2.8573e-14 | 1.5044e-15 | 1.6487e-13 | 5 | 4 | 0 | 0 |
| F2 | 0.97 | 1.31 | 1.23 | 1.58 | 1.58 | 1.58 | 0.0973 | 0.115 | 0.115 | 0.115 | 5.2479e-08 | 9.0260e-17 | 1.6575e-15 | 1.6575e-15 | 1 | 1 | 4 | 4 |
| hood | 0.95 | 1.21 | 1.78 | 2.07 | 1.88 | 1.84 | 0.159 | 0.174 | 0.164 | 0.174 | 8.2467e-11 | 3.0020e-19 | 1.0720e-16 | 5.7692e-14 | 2 | 1 | 0 | 0 |
| msc23052 | 0.08 | 0.10 | 0.13 | 0.16 | 0.34 | 0.13 | 0.0149 | 0.0155 | 0.0266 | 0.0158 | 4.9256e-12 | 1.3191e-19 | 4.3782e-12 | 4.9256e-12 | 1 | 0 | 28 | 0 |
| nasasrb | 0.43 | 0.57 | 0.56 | 0.70 | 0.65 | 0.60 | 0.0458 | 0.0532 | 0.0509 | 0.0519 | 8.9084e-09 | 2.7733e-17 | 8.3262e-16 | 3.7799e-12 | 7 | 3 | 0 | 0 |
| oilpan | 0.39 | 0.52 | 0.50 | 0.62 | 0.70 | 0.70 | 0.0473 | 0.0537 | 0.0576 | 0.0610 | 5.2672e-08 | 3.6102e-17 | 4.5208e-15 | 4.5208e-15 | 2 | 2 | 4 | 4 |
| pwtk | 1.90 | 2.52 | 2.89 | 3.54 | 6.56 | 3.62 | 0.232 | 0.251 | 0.411 | 0.275 | 1.1980e-11 | 1.6871e-20 | 2.2834e-13 | 1.1112e-12 | 1 | 1 | 33 | 4 |
| s3dkq4m2 | 0.86 | 1.18 | 1.04 | 1.37 | 1.70 | 1.70 | 0.0877 | 0.103 | 0.122 | 0.130 | 5.6467e-08 | 9.7050e-17 | 6.5989e-17 | 6.5989e-17 | 1 | 1 | 12 | 12 |
| s3dkt3m2 | 0.59 | 0.77 | 0.74 | 0.93 | 1.27 | 1.27 | 0.0640 | 0.0742 | 0.0979 | 0.0989 | 2.1728e-08 | 1.0306e-16 | 5.4217e-17 | 5.4217e-17 | 1 | 1 | 12 | 12 |
| ship_003 | 5.40 | 7.96 | 6.16 | 8.75 | 6.30 | 6.15 | 0.368 | 0.501 | 0.375 | 0.395 | 1.3291e-14 | 2.1702e-23 | 1.3699e-16 | 1.3291e-14 | 1 | 0 | 0 | 0 |
| shipsec8 | 2.59 | 3.75 | 3.18 | 4.35 | 5.64 | 3.19 | 0.222 | 0.268 | 0.361 | 0.198 | 3.4866e-13 | 4.9033e-22 | 1.3850e-13 | 3.4866e-13 | 1 | 0 | 28 | 0 |
| thread | 2.77 | 3.93 | 3.25 | 4.43 | 5.07 | 5.07 | 0.194 | 0.255 | 0.290 | 0.311 | 1.1477e-08 | 1.0015e-16 | 1.1477e-08 | 1.1477e-08 | 1 | 1 | 28 | 28 |
| vanbody | 0.18 | 0.23 | 0.29 | 0.34 | 0.41 | 0.29 | 0.0340 | 0.0365 | 0.0432 | 0.0366 | 4.3586e-14 | 1.2249e-22 | 2.4848e-15 | 4.3586e-14 | 2 | 0 | 4 | 0 |
| case39 | 0.38 | 0.53 | 0.47 | 0.62 | 1.00 | 1.01 | 0.0339 | 0.0397 | 0.0638 | 0.0609 | 2.9628e-10 | 6.2324e-18 | 3.3708e-11 | 3.3708e-11 | 1 | 1 | 33 | 33 |
| TSOPF_FS_b39_c30 | 0.64 | 0.90 | 1.59 | 1.87 | 2.70 | 2.71 | 0.104 | 0.118 | 0.172 | 0.174 | 9.8540e-11 | 4.9384e-18 | 5.0510e-11 | 5.0510e-11 | 1 | 1 | 28 | 28 |
| boneS01 | 2.93 | 4.13 | 3.77 | 4.99 | 3.96 | 3.88 | 0.228 | 0.289 | 0.238 | 0.234 | 1.5710e-07 | 2.4605e-16 | 4.2061e-16 | 1.1844e-12 | 5 | 3 | 0 | 0 |
| filter3D | 0.72 | 0.94 | 1.97 | 2.20 | 2.04 | 2.03 | 0.120 | 0.131 | 0.124 | 0.124 | 5.9545e-08 | 1.7062e-16 | 3.2143e-16 | 1.8888e-13 | 3 | 2 | 0 | 0 |
| rail_79841 | 0.07 | 0.07 | 0.12 | 0.12 | 0.13 | 0.13 | 0.0111 | 0.0106 | 0.0120 | 0.0118 | 2.4373e-07 | 2.2440e-16 | 7.6461e-16 | 4.5294e-13 | 3 | 2 | 0 | 0 |
| t3dl | 0.21 | 0.28 | 0.43 | 0.51 | 0.50 | 0.50 | 0.0252 | 0.0300 | 0.0292 | 0.0294 | 7.2351e-08 | 2.0181e-16 | 1.0876e-16 | 1.0876e-16 | 1 | 1 | 4 | 4 |
| cfd1 | 0.95 | 1.31 | 1.83 | 2.21 | 1.89 | 1.88 | 0.109 | 0.121 | 0.112 | 0.111 | 9.7009e-08 | 9.4014e-17 | 6.5401e-17 | 1.3781e-14 | 3 | 2 | 0 | 0 |
| cfd2 | 2.19 | 3.06 | 3.56 | 4.45 | 3.67 | 3.65 | 0.209 | 0.241 | 0.215 | 0.209 | 1.3805e-07 | 1.0586e-16 | 8.7176e-17 | 4.1914e-14 | 3 | 2 | 0 | 0 |
| copter2 | 0.48 | 0.63 | 0.98 | 1.14 | 1.06 | 1.04 | 0.0495 | 0.0571 | 0.0570 | 0.0564 | 3.1027e-05 | 1.0599e-13 | 4.2681e-15 | 4.9110e-12 | 7 | 5 | 0 | 0 |
| parabolic_fem | 0.97 | 1.21 | 3.62 | 3.87 | 3.79 | 3.74 | 0.214 | 0.227 | 0.223 | 0.222 | 5.0959e-07 | 7.5833e-16 | 1.5511e-15 | 1.1547e-12 | 3 | 2 | 0 | 0 |
| shallow_water1 | 0.07 | 0.09 | 0.35 | 0.37 | 0.37 | 0.36 | 0.0220 | 0.0215 | 0.0225 | 0.0224 | 3.0114e-07 | 7.1967e-16 | 2.0391e-16 | 1.2796e-13 | 2 | 1 | 0 | 0 |
| stokes128 | 0.07 | 0.09 | 0.11 | 0.13 | 0.16 | 0.16 | 0.0107 | 0.0111 | 0.0131 | 0.0129 | 8.3321e-06 | 3.2026e-14 | 1.0054e-15 | 1.0054e-15 | 1 | 1 | 4 | 4 |
| a0nsdsil | 0.02 | 0.02 | 0.06 | 0.06 | 0.07 | 0.06 | 0.0057 | 0.0056 | 0.0061 | 0.0061 | 2.8434e-10 | 1.2970e-18 | 2.0195e-17 | 1.3332e-12 | 3 | 1 | 0 | 0 |
| a5esindl | 0.01 | 0.01 | 0.04 | 0.04 | 0.04 | 0.04 | 0.0039 | 0.0040 | 0.0044 | 0.0038 | 5.4069e-10 | 1.3010e-18 | 2.4379e-17 | 1.3745e-12 | 3 | 1 | 0 | 0 |
| blockqp1 | 0.03 | 0.03 | 0.07 | 0.07 | 0.08 | 0.07 | 0.0081 | 0.0084 | 0.0092 | 0.0085 | 1.6571e-06 | 1.0505e-14 | 2.2717e-16 | 7.0114e-13 | 3 | 2 | 0 | 0 |
| boyd2 | 84.01 | 156.53 | 84.22 | 156.93 | 98.01 | 88.73 | 4.572 | 1.987 | 5.291 | 4.811 | 3.1674e-09 | 1.7893e-17 | 1.3139e-12 | 2.0722e-12 | 2 | 2 | 33 | 4 |
| brainpc2 | 0.02 | 0.02 | 0.03 | 0.04 | 0.04 | 0.04 | 0.0034 | 0.0033 | 0.0035 | 0.0037 | 7.3372e-07 | 9.3950e-16 | 3.5660e-16 | 3.5660e-16 | 2 | 2 | 0 | 0 |
| c-56 | 0.04 | 0.05 | 0.11 | 0.11 | 0.17 | 0.11 | 0.0089 | 0.0087 | 0.0123 | 0.0091 | 1.6866e-11 | 5.4522e-20 | 1.3617e-16 | 4.4217e-13 | 4 | 1 | 12 | 0 |
| cont-201 | 0.56 | 0.76 | 0.93 | 1.15 | 1.56 | 1.39 | 0.0516 | 0.0598 | 0.0871 | 0.0778 | 9.4863e-05 | 2.2078e-13 | 9.6909e-17 | 1.5005e-13 | 1 | 1 | 24 | 16 |
| cont-300 | 1.16 | 1.58 | 1.93 | 2.36 | 3.39 | 3.42 | 0.102 | 0.123 | 0.190 | 0.173 | 2.2748e-04 | 4.0072e-13 | 4.3510e-16 | 4.3510e-16 | 1 | 1 | 28 | 28 |
| dixmaanl | 0.02 | 0.02 | 0.05 | 0.05 | 0.06 | 0.05 | 0.0056 | 0.0056 | 0.0065 | 0.0062 | 2.0930e-05 | 4.2226e-14 | 1.1610e-16 | 6.9768e-14 | 3 | 2 | 0 | 0 |
| jnlbrng1 | 0.03 | 0.04 | 0.16 | 0.16 | 0.16 | 0.16 | 0.0104 | 0.0099 | 0.0107 | 0.0107 | 1.5541e-07 | 4.9092e-16 | 1.8882e-16 | 2.0713e-13 | 2 | 1 | 0 | 0 |
| k1_san | 0.10 | 0.11 | 0.49 | 0.50 | 0.64 | 0.56 | 0.0278 | 0.0286 | 0.0361 | 0.0338 | 6.5304e-07 | 6.5304e-07 | 1.0906e-17 | 5.1150e-15 | 1 | 1 | 12 | 4 |
| ncvxbqp1 | 0.08 | 0.10 | 0.32 | 0.35 | 0.57 | 0.57 | 0.0190 | 0.0193 | 0.0323 | 0.0327 | 5.0848e-07 | 5.5830e-16 | 6.3422e-08 | 6.3422e-08 | 2 | 2 | 28 | 28 |
| torsion1 | 0.03 | 0.03 | 0.05 | 0.05 | 0.05 | 0.05 | 0.0041 | 0.0038 | 0.0041 | 0.0040 | 2.7399e-07 | 5.4982e-16 | 1.4803e-16 | 3.4396e-13 | 2 | 1 | 0 | 0 |
| G2_circuit | 0.42 | 0.55 | 0.52 | 0.65 | 0.57 | 0.56 | 0.0316 | 0.0382 | 0.0345 | 0.0342 | 9.3088e-08 | 5.1094e-17 | 2.3525e-17 | 1.2924e-14 | 3 | 2 | 0 | 0 |
| finan512 | 0.07 | 0.07 | 0.43 | 0.43 | 0.45 | 0.44 | 0.0267 | 0.0256 | 0.0277 | 0.0272 | 3.9712e-08 | 1.1616e-16 | 3.5138e-17 | 3.1115e-14 | 2 | 1 | 0 | 0 |
| qa8fk | 1.38 | 1.94 | 2.15 | 2.73 | 2.50 | 2.51 | 0.120 | 0.149 | 0.138 | 0.140 | 2.3260e-06 | 1.5754e-15 | 2.8583e-16 | 2.8583e-16 | 1 | 1 | 4 | 4 |
| thermal1 | 0.09 | 0.10 | 0.16 | 0.17 | 0.18 | 0.18 | 0.0143 | 0.0144 | 0.0157 | 0.0154 | 3.9185e-07 | 5.3882e-16 | 2.0527e-16 | 7.2083e-14 | 3 | 2 | 0 | 0 |
| thermomech_dM | 0.23 | 0.26 | 1.17 | 1.20 | 1.22 | 1.20 | 0.0701 | 0.0726 | 0.0727 | 0.0766 | 9.8505e-09 | 2.9112e-17 | 6.1288e-18 | 5.4852e-15 | 2 | 1 | 0 | 0 |
| tmt_sym | 1.21 | 1.50 | 4.92 | 5.23 | 5.77 | 5.81 | 0.292 | 0.308 | 0.358 | 0.326 | 3.0980e-07 | 2.0406e-16 | 6.6903e-17 | 6.6903e-17 | 2 | 2 | 4 | 4 |
| aug3dcqp | 0.04 | 0.05 | 0.22 | 0.23 | 0.22 | 0.22 | 0.0126 | 0.0123 | 0.0126 | 0.0114 | 3.1344e-16 | 7.5887e-25 | 3.1344e-16 | 3.1344e-16 | 0 | 0 | 0 | 0 |
| cant | 0.70 | 0.93 | 1.92 | 2.17 | 2.51 | 2.52 | 0.131 | 0.135 | 0.164 | 0.165 | 9.7857e-08 | 2.7082e-16 | 1.6021e-16 | 1.6021e-16 | 1 | 1 | 12 | 12 |
| d_pretok | 0.55 | 0.70 | 1.63 | 1.80 | 1.89 | 1.90 | 0.0985 | 0.100 | 0.106 | 0.112 | 5.0041e-09 | 1.3495e-17 | 8.3858e-20 | 8.3858e-20 | 1 | 1 | 4 | 4 |
| darcy003 | 0.29 | 0.33 | 0.62 | 0.66 | 0.70 | 0.68 | 0.0502 | 0.0527 | 0.0550 | 0.0545 | 8.0935e-07 | 1.4268e-15 | 4.4588e-17 | 1.7657e-14 | 3 | 2 | 0 | 0 |
| Dubcova2 | 0.13 | 0.16 | 0.18 | 0.22 | 0.20 | 0.20 | 0.0194 | 0.0199 | 0.0202 | 0.0210 | 2.9705e-07 | 4.3318e-16 | 2.6094e-16 | 2.6094e-16 | 2 | 2 | 0 | 0 |
| Dubcova3 | 0.24 | 0.29 | 1.04 | 1.10 | 1.08 | 1.07 | 0.0829 | 0.0862 | 0.0845 | 0.0852 | 8.8322e-08 | 2.3664e-16 | 1.1648e-16 | 1.6047e-12 | 2 | 1 | 0 | 0 |
| ecology1 | 2.12 | 2.75 | 2.73 | 3.40 | 3.75 | 3.74 | 0.192 | 0.213 | 0.232 | 0.249 | 1.6257e-06 | 7.7664e-16 | 3.2838e-16 | 3.2838e-16 | 1 | 1 | 4 | 4 |
| ecology2 | 2.10 | 2.72 | 2.71 | 3.37 | 3.35 | 3.17 | 0.190 | 0.211 | 0.211 | 0.217 | 1.2482e-06 | 9.5026e-16 | 2.2080e-15 | 6.8880e-13 | 7 | 5 | 0 | 0 |
| helm2d03 | 0.68 | 0.83 | 2.47 | 2.64 | 2.64 | 2.61 | 0.157 | 0.159 | 0.155 | 0.150 | 8.5695e-05 | 9.7370e-14 | 1.3157e-16 | 5.5712e-15 | 4 | 3 | 0 | 0 |
| helm3d01 | 0.27 | 0.35 | 0.56 | 0.64 | 0.59 | 0.58 | 0.0309 | 0.0355 | 0.0325 | 0.0344 | 5.5628e-05 | 1.0883e-13 | 2.6408e-16 | 4.6545e-12 | 5 | 3 | 0 | 0 |
| tuma1 | 0.02 | 0.02 | 0.09 | 0.09 | 0.09 | 0.09 | 0.0054 | 0.0051 | 0.0055 | 0.0055 | 3.0377e-06 | 6.5697e-15 | 2.4636e-17 | 2.4636e-17 | 2 | 2 | 0 | 0 |
| turon_m | 0.50 | 0.63 | 1.62 | 1.75 | 1.87 | 1.88 | 0.0916 | 0.0987 | 0.105 | 0.0997 | 1.9880e-06 | 4.4638e-15 | 4.4807e-18 | 4.4807e-18 | 1 | 1 | 4 | 4 |
| wathen120 | 0.05 | 0.06 | 0.08 | 0.09 | 0.08 | 0.08 | 0.0096 | 0.0096 | 0.0097 | 0.0098 | 4.7024e-10 | 1.1000e-18 | 3.8770e-16 | 3.8770e-16 | 1 | 1 | 0 | 0 |
| crystk03 | 0.43 | 0.59 | 0.61 | 0.78 | 0.75 | 0.76 | 0.0480 | 0.0539 | 0.0559 | 0.0570 | 2.8861e-08 | 8.1166e-17 | 6.7088e-16 | 6.7088e-16 | 1 | 1 | 4 | 4 |