Fastest Fourier Transform in the West

Devendra Ghate and Nachi Gupta

devg@comlab.ox.ac.uk, nachi@comlab.ox.ac.uk

Oxford University Computing Laboratory

Cooley-Tukey Algorithm for FFT

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}nk}, \qquad k = 0, \ldots, N-1$$

If we solve for X_k directly then $O(N^2)$ operations required for the solution.

If $N = p \times q$, then this transformation can be split into two separate transformations...

...recursively, we get the FFT algorithm.

If N is split into $\frac{N}{r}$ intervals of r each then the complexity is $r \times N \times \log_r N$.

This can be generalised to mixed-radix algorithms.

Overview of FFT algorithms

- Cooley-Tukey FFT algorithm (popularized in 1965), known by Gauss in 1805.
- Prime-factor FFT algorithm – a.k.a. Good-Thomas (1958-1963), $N = N_1 N_2$ becomes 2D N_1 by N_2 DFT for only relatively prime N_1, N_2.
- Bruun's FFT algorithm (1978, generalized to arbitrary even composite sizes by H. Murakami in 1996), recursive polynomial factorization approach.
- Rader's FFT algorithm (1968), for prime size by expressing DFT as a convolution.
- Bluestein's FFT algorithm (1968) – a.k.a. chirp z-transform algorithm (1969), for prime sizes by expressing DFT as a convolution.
- Rader-Brenner (1976) – Cooley-Tukey like, but purely imag twiddle factors

What is FFTW?

- FFTW is a package for computing a one or multidimensional complex discrete Fourier transform (DFT) of arbitrary size.
- At the heart of FFTW lies a philosophy geared towards speed, portability, and elegance – achieved via an adaptive software architecture.
- Code, documentation, and some benchmarking results at `www.fftw.org`.

Features

- Speed.
- Both one-dimensional and multi-dimensional transforms.
- Arbitrary size (small prime factors are best, but FFTW uses O(N log N) algorithms even for prime sizes).
- Fast transforms of purely real input or output data.
- Discrete Cosine Transform and Discrete Sine Transform.
- Parallel Transforms (an MPI version of distributed memory transforms).
- Portable to any platform with a C compiler.
- Both C and Fortran interfaces.
- GPL

Wilkinson Prize

FFTW received the **1999 J. H. Wilkinson Prize for Numerical Software**, which is awarded every four years to the software that “best addresses all phases of the preparation of high quality numerical software.”

Wilkinson, a seminal figure in modern numerical analysis, was a key proponent of the notion of reusable common libraries for scientific computing.

Why FFTW is *fast!*

The transform is computed by an *executor, composed of highly optimized composable blocks of C code called* codelets.

- At runtime, a *planner finds an efficient way to mix these codelets: it* measures the speeds of different plans and chooses the best using a dynamic programming algorithm.

- The executor then interprets this plan with negligible overhead.

- Codelets are generated automatically and fast.

FFTW is easy to use

FFTW’s internal complexity is not visible to the user.

```
int n = 1024;

// allocate memory
in  = fftw_malloc( sizeof( fftw_complex ) * n );
out = fftw_malloc( sizeof( fftw_complex ) * n );

// Create plan
fftw_plan plan = fftw_plan_dft_1d( n, in, out,
                                FFTW_FORWARD, FFTW_ESTIMATE );

// ... fill in ...

// Execute
fftw_execute( plan );

// Destroy plan
fftw_destroy_plan( plan );
```

The Executor

The executor implements the recursive divide and conquer Cooley-Tukey FFT algorithm.

- The executor is composed of many optimized code sequences called codelets.
- Codelets come in two flavors:
  - Nontwiddle codelets compute transforms of small sizes.
  - Twiddle codelets combine small transforms to compute bigger transforms.
- The executor invokes the codelets as dictated by the plan.

The Planner

A planner tries out various combinations of codelets to decide on the best algorithm.

- The planner can produce many plans, measure the speed of them, and then pick the best.
- This is accomplished via dynamic programming techniques.
- The planner can produce a “reasonable” plan quickly if desired.
- The planner collects information about the machines, which can be stored on disk and reused at a later time.

What a plan looks like

```
FFT( 128 ) =
     DIVIDE-AND-CONQUER( 128, 4 )
     DIVIDE-AND-CONQUER( 32, 8 )
     SOLVE( 4 )
```

The plan is a sequence of instructions, either

(SOLVE**)**(n): compute the FFT of size n using a nontwiddle codelet.

(DIVIDE-AND-CONQUER**)**(n, p): solve p problems of size n/p using the rest of the plan, and then combine the results using a twiddle codelet.

What a real plan looks like

Plan for an array of size 247

```
(dft-ct-dit/13
 (dftw-generic-dit-13-19
   (dft-direct-13-x19 "n1\_13"))
     (dft-vrank>=1-x13/1
      (dft-generic-19)))
```

FFTW Plans

- FFTW_ESTIMATE - no run-time tuning, probably suboptimal
- FFTW_MEASURE - experiment with different algorithms and choose fastest
- FFTW_PATIENT - experiment with more algorithms, ...
- FFTW_EXHAUSTIVE - even more, ...

Creating a plan for an array of size 823467 in `FFTW_EXHAUSTIVE` modes takes 25 minutes.

Wisdom

A central or local database of plans for various array sizes for a given hardware configuration.

- Helps greatly in the estimate mode
- Reduces computation time for the exhaustive mode
- Wisdom from the other array sizes is also utilised
- Can be exported from one architecture to another

Matlab uses a wisdom database by default.

FFTW Compiler

- Various basic algorithms written in CamL (a functional language)
- Directed acyclic graph (DAG) created for all these algorithms
- Optimisation rules defined especially suited for FFT algorithms (these also include standard optimisations carried out by a compiler)
- Small snippets of code, “codelets”, generated for FFT algorithms of size $N = 2 : 16$

Multi-dimensional Arrays

FFTW does not accept multidimensional arrays.

- A single dimensional array in the "row-major" format (C standard)
- Use of `fftw_malloc` for contiguous array allocation

MKL routines accept multi-dimensinal arrays. Various compact formats are also defined.

NAG FFT routines

- Complex-Complex, Real-Complex routines
- FFT algorithm (Brigham 1974)
- Scaling factor of $\frac{1}{\sqrt{n}}$
- Array dimension n s.t. the largest prime factor of n does not exceed 19 and total number of factors of n including repetitions, does not exceed 20.

MKL FFT routines

- Complex-Complex & real-complex transforms
- Multi-dimensional arrays upto order 7
- Interface for both FORTRAN and C
- Mixed-redix transforms !

MKL also provides an API for FFTW 3.x !!!

MATLAB uses FFTW3 internally for all FFT calculations. Interface for setting plan options available.

Henrici

icc used with following flags:

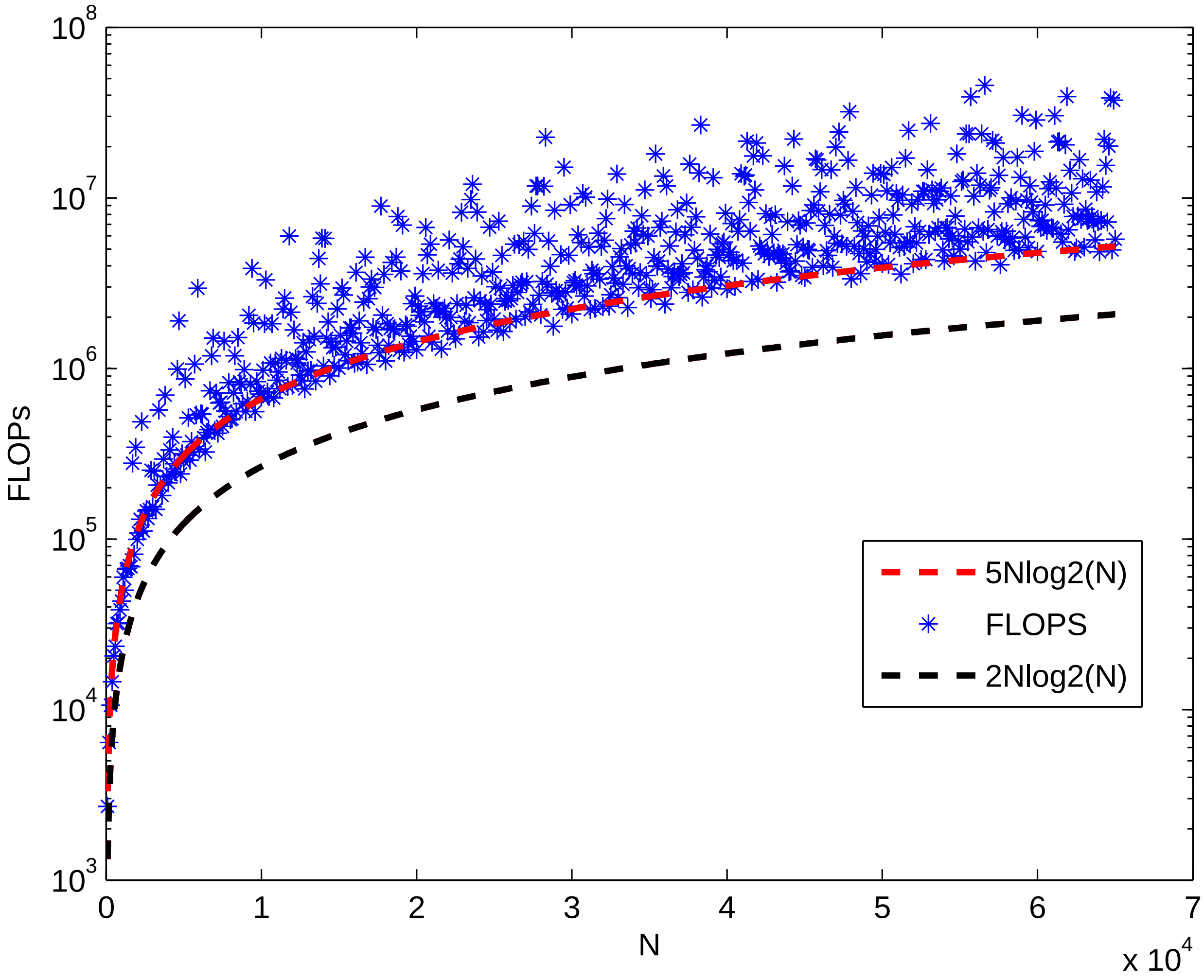
- -O3 : Level-3 opitization
- -c99 : Support for complex numbers using C9X standards
- -ipo : Multi file inlining
- -funroll-loops : Unrolling the loops
- -xN : Compile and optimise code specifically for Pentium-4
- -align: Analyse and reorder the memory layout

Benchmark

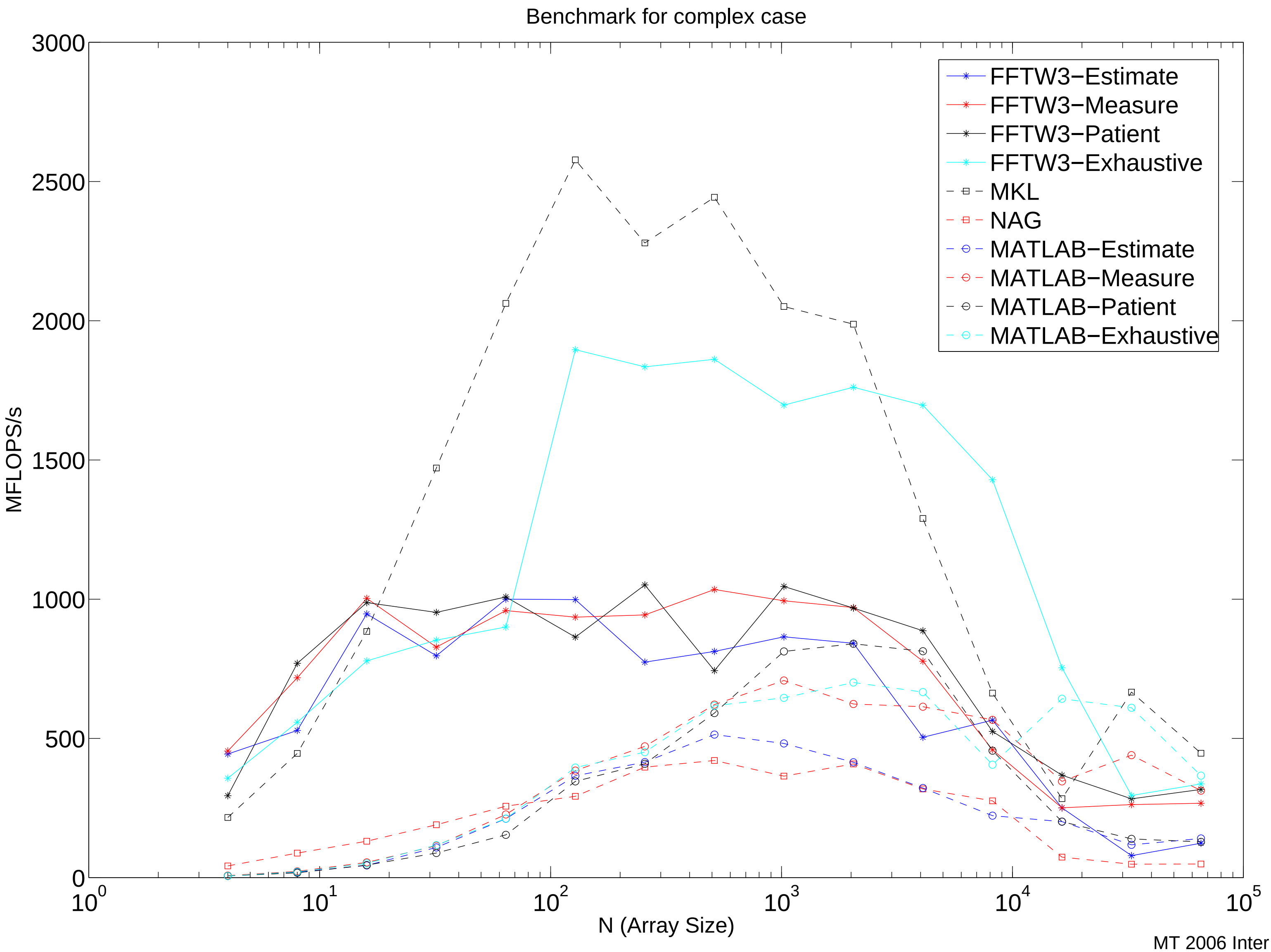
- Time calculated by average over 1000 runs for each array size N
- Minimum time from 8 trials
- FLOPS = $5 \times N \times \log_2 N$
- Compiler: icc

FLOPS

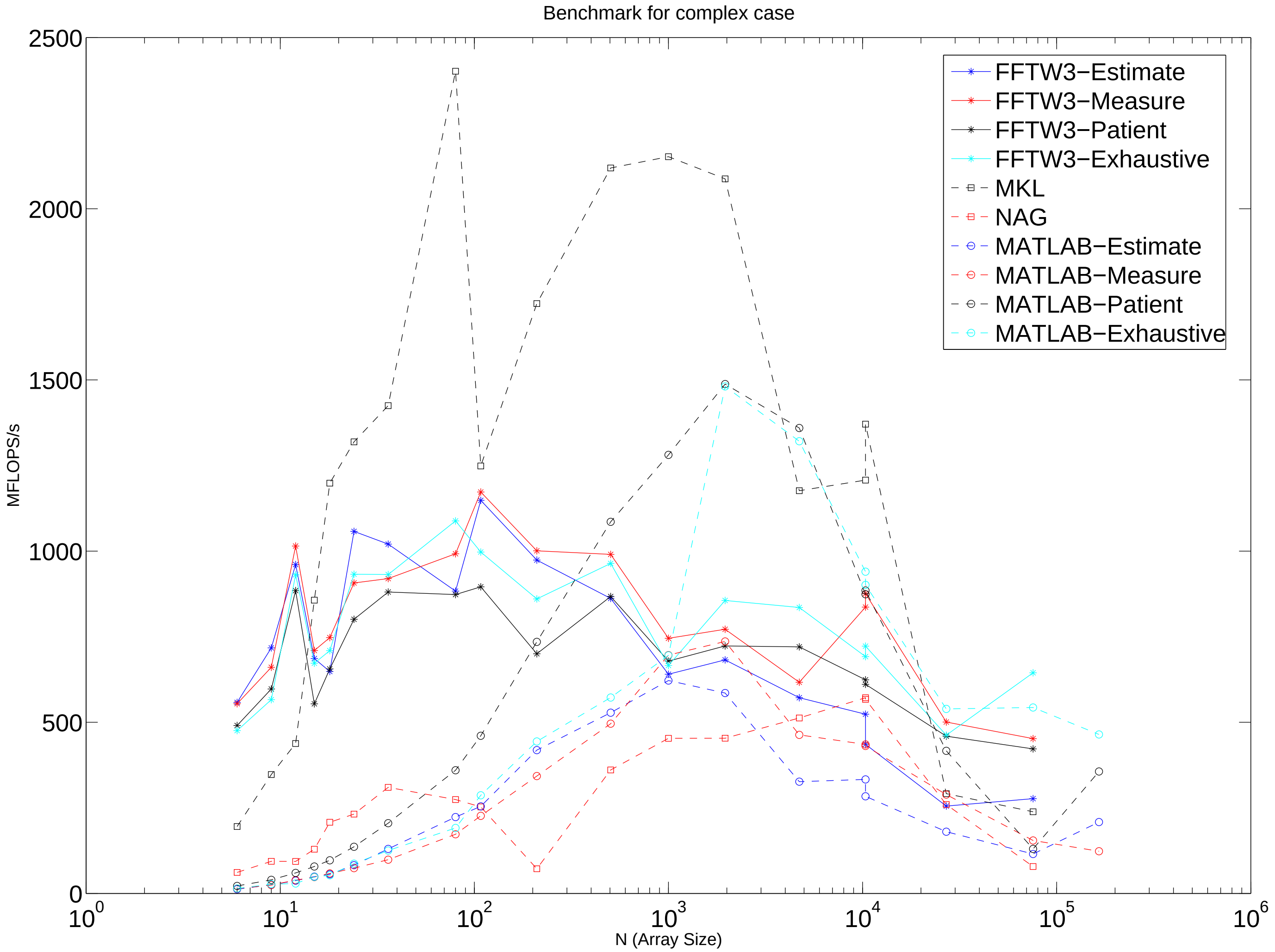
```
void fftw_flops(plan, add, mul, fma)
```



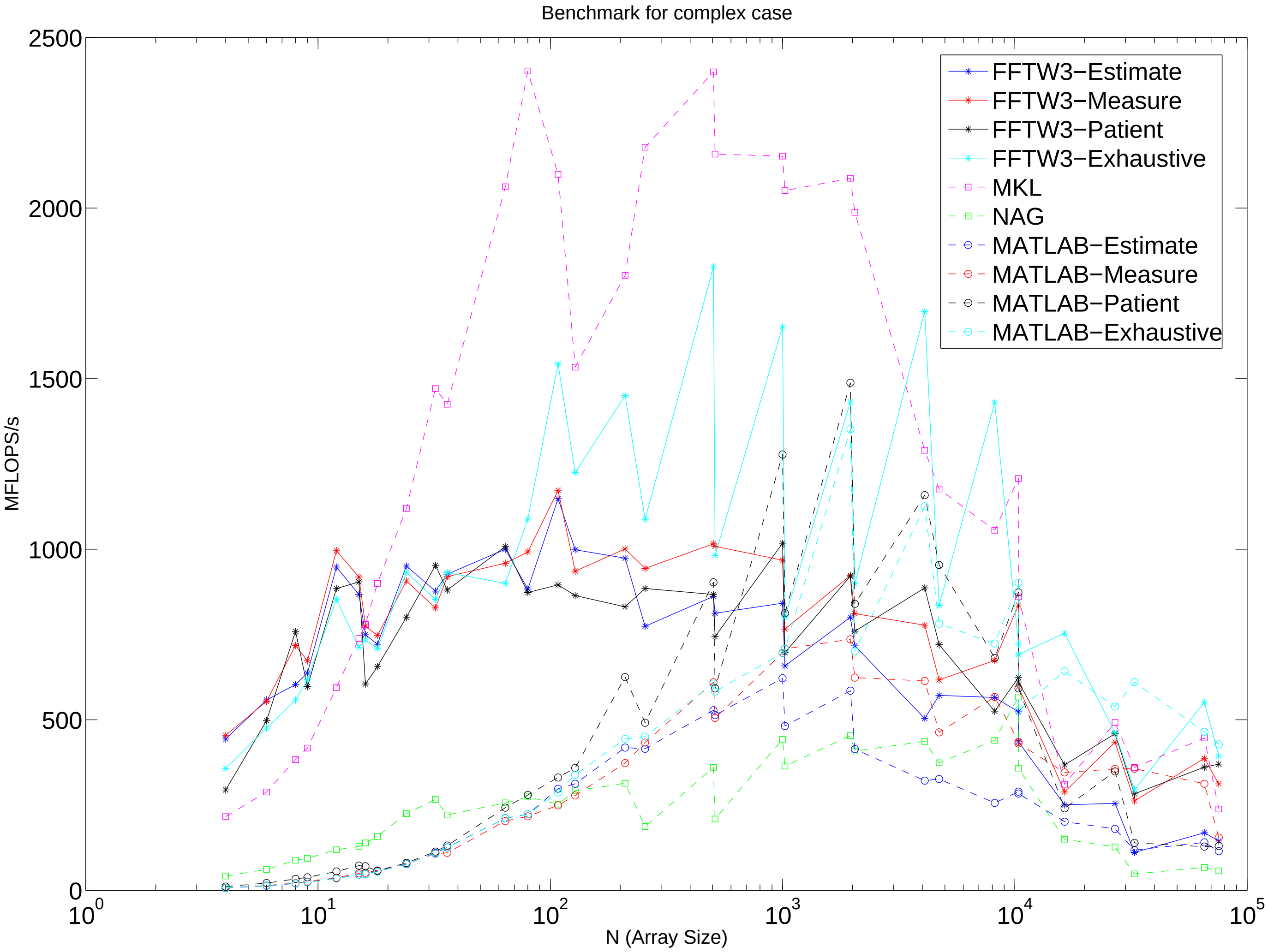
1D Complex Bench - Powers of 2



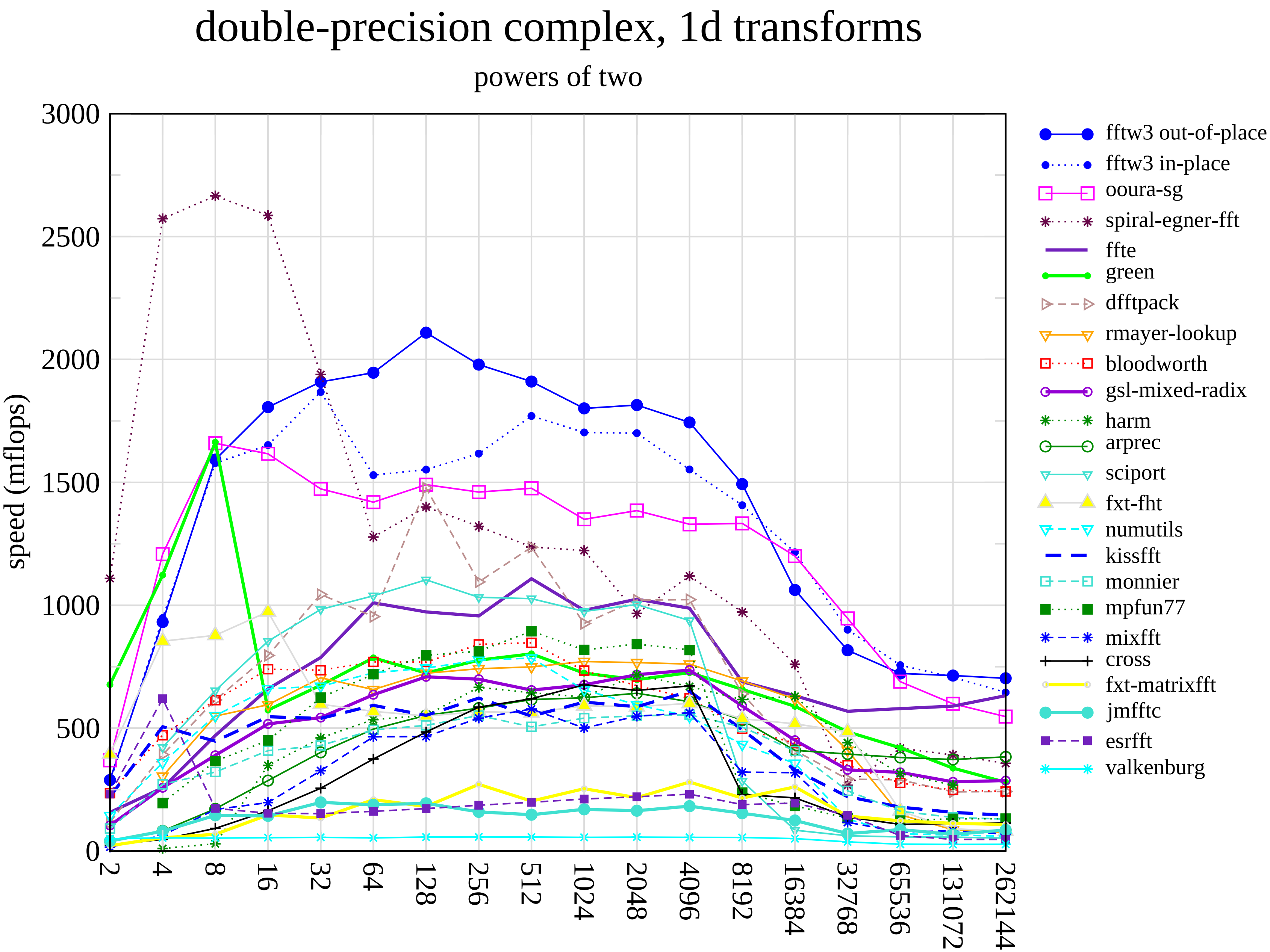
1D Complex Bench - General



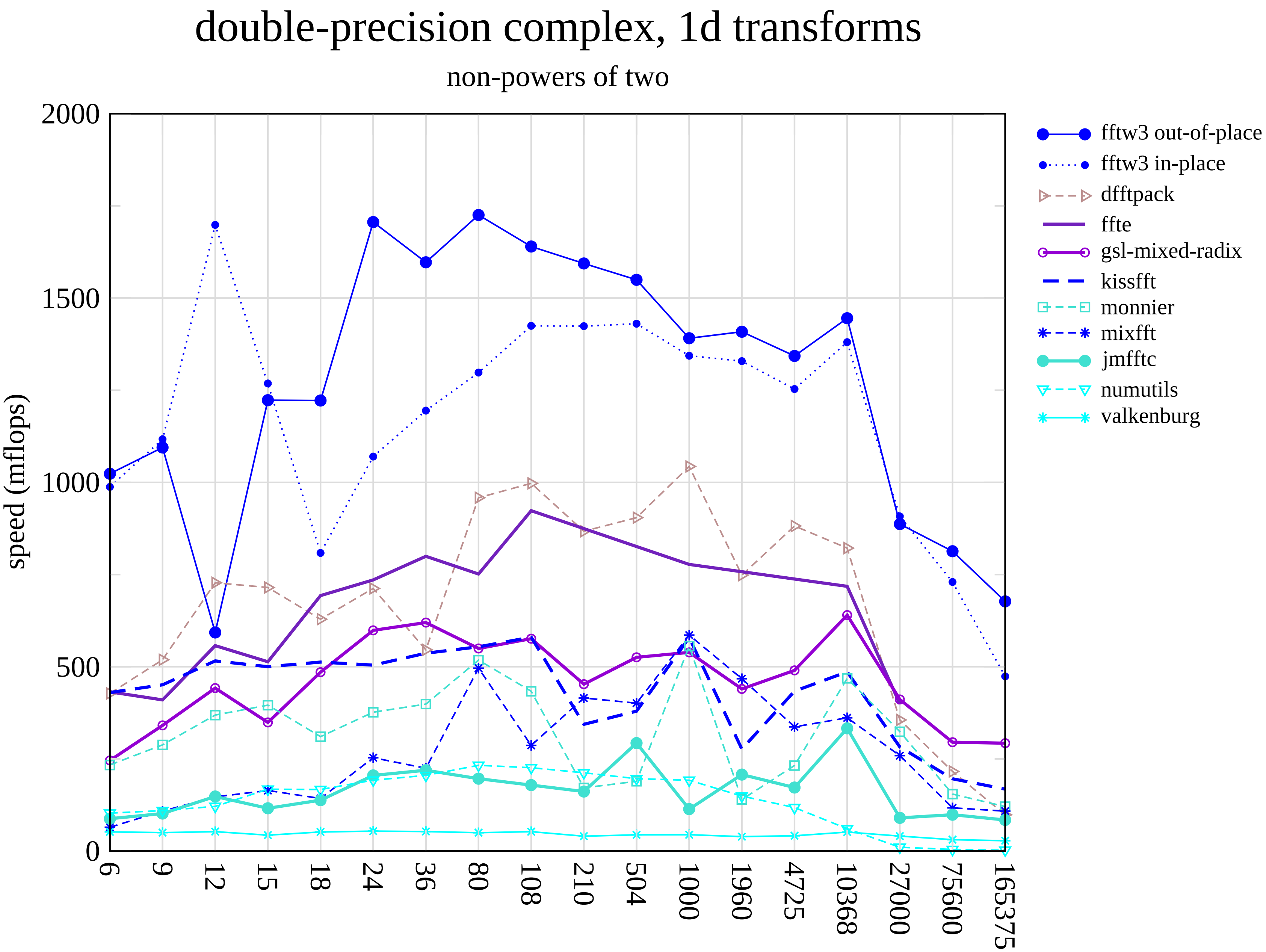
1D Complex Bench - Combined



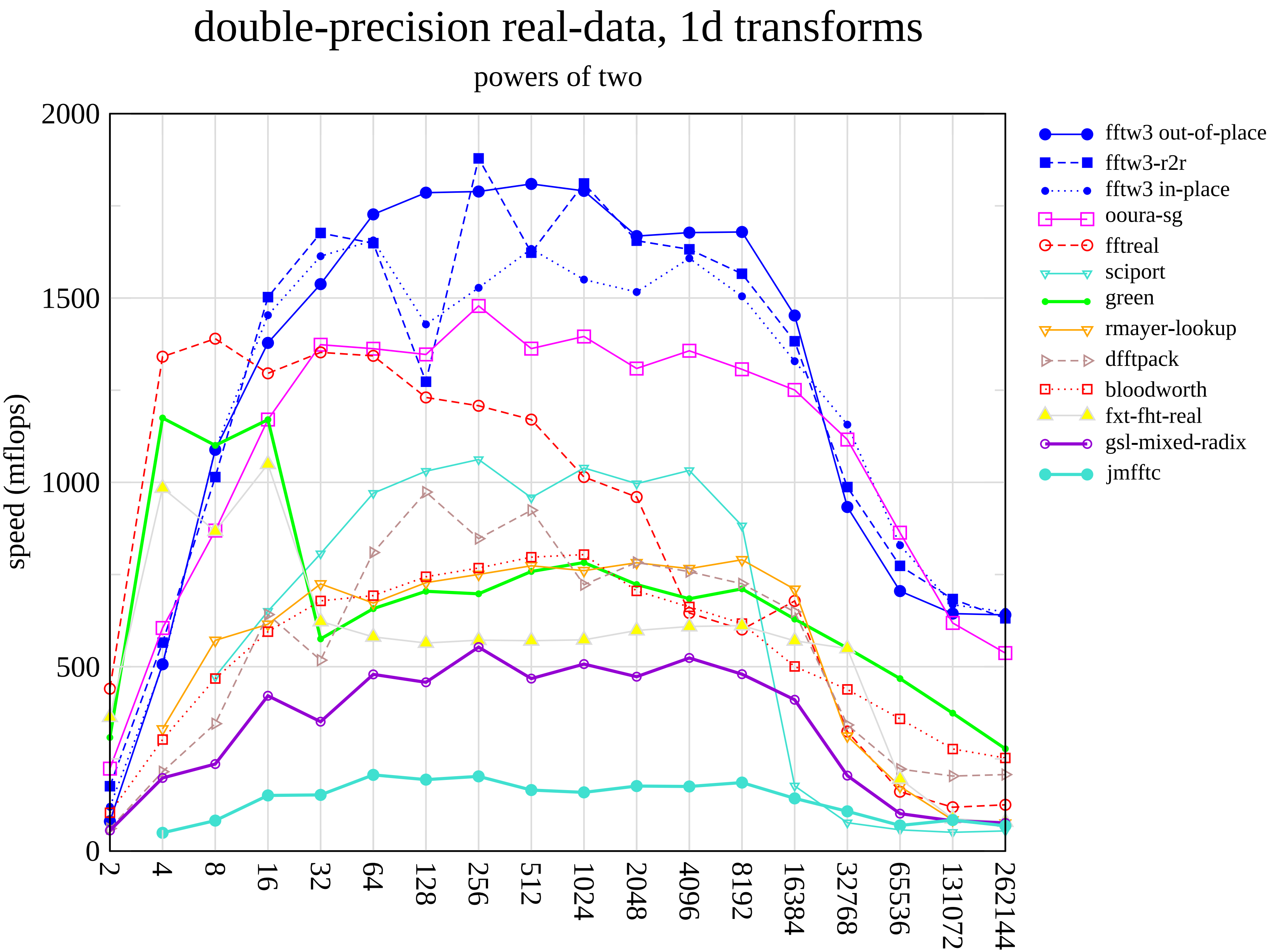
BenchFFT Results on Henrici1



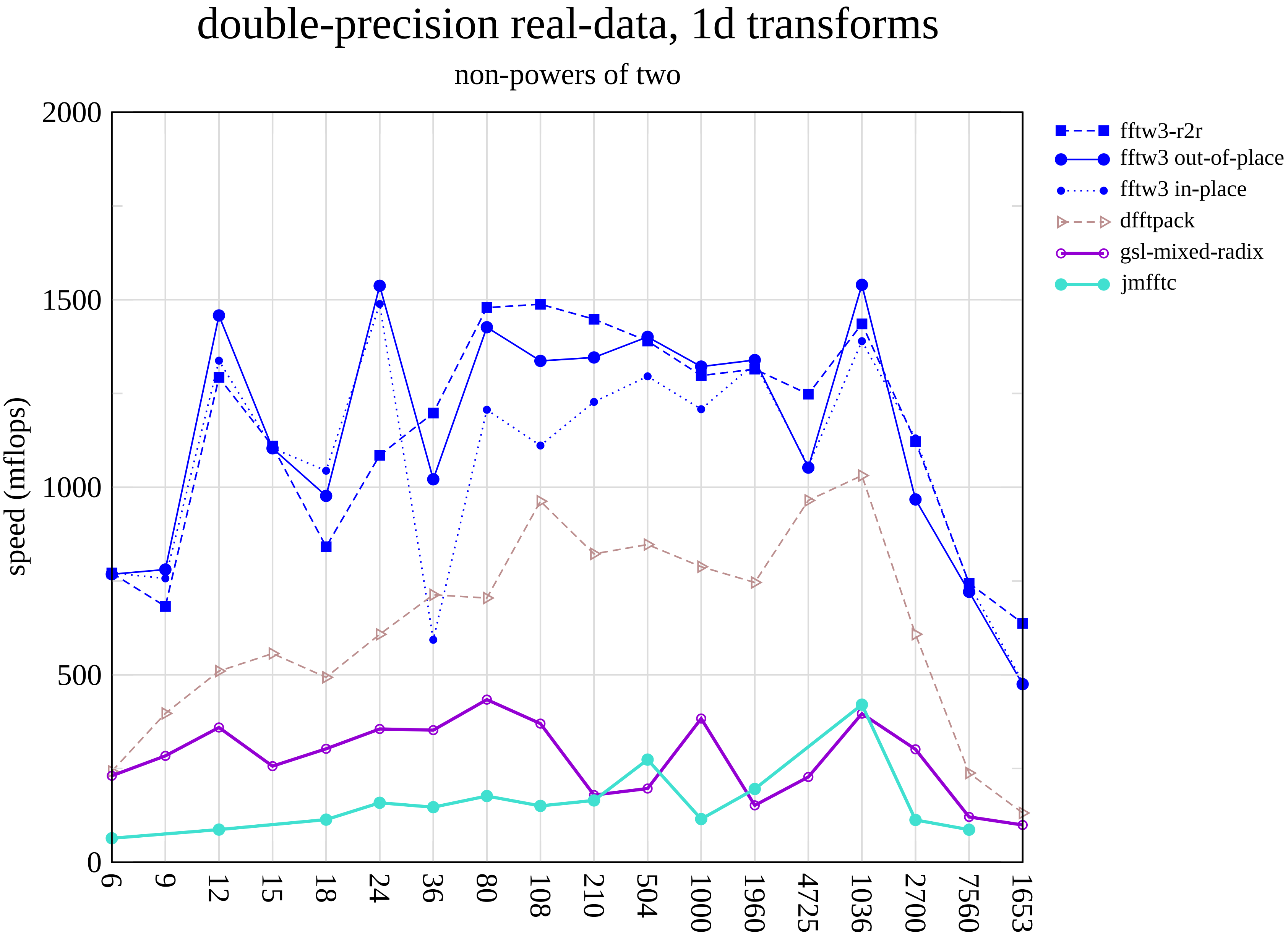
BenchFFT Results on Henrici1



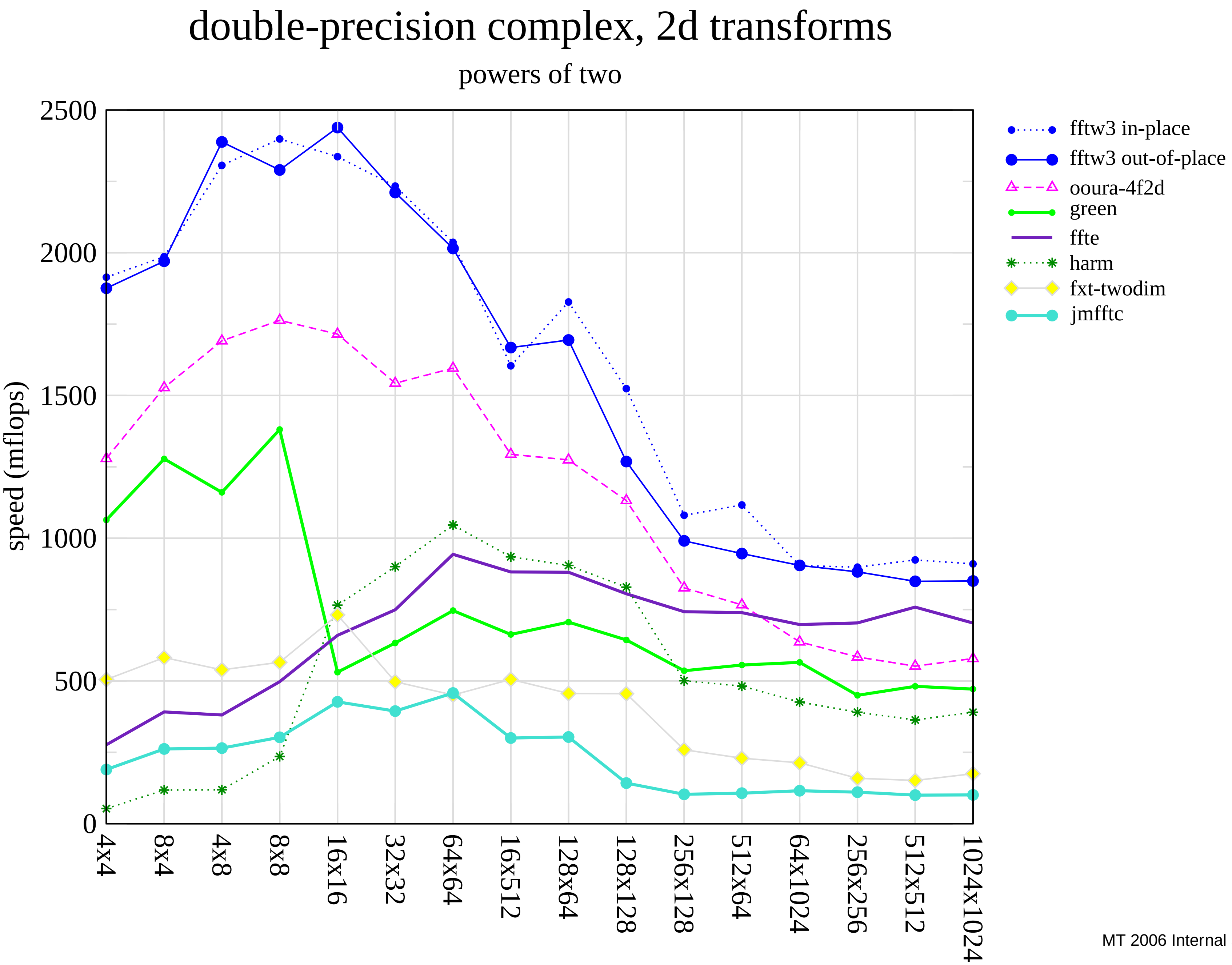
BenchFFT Results on Henrici1



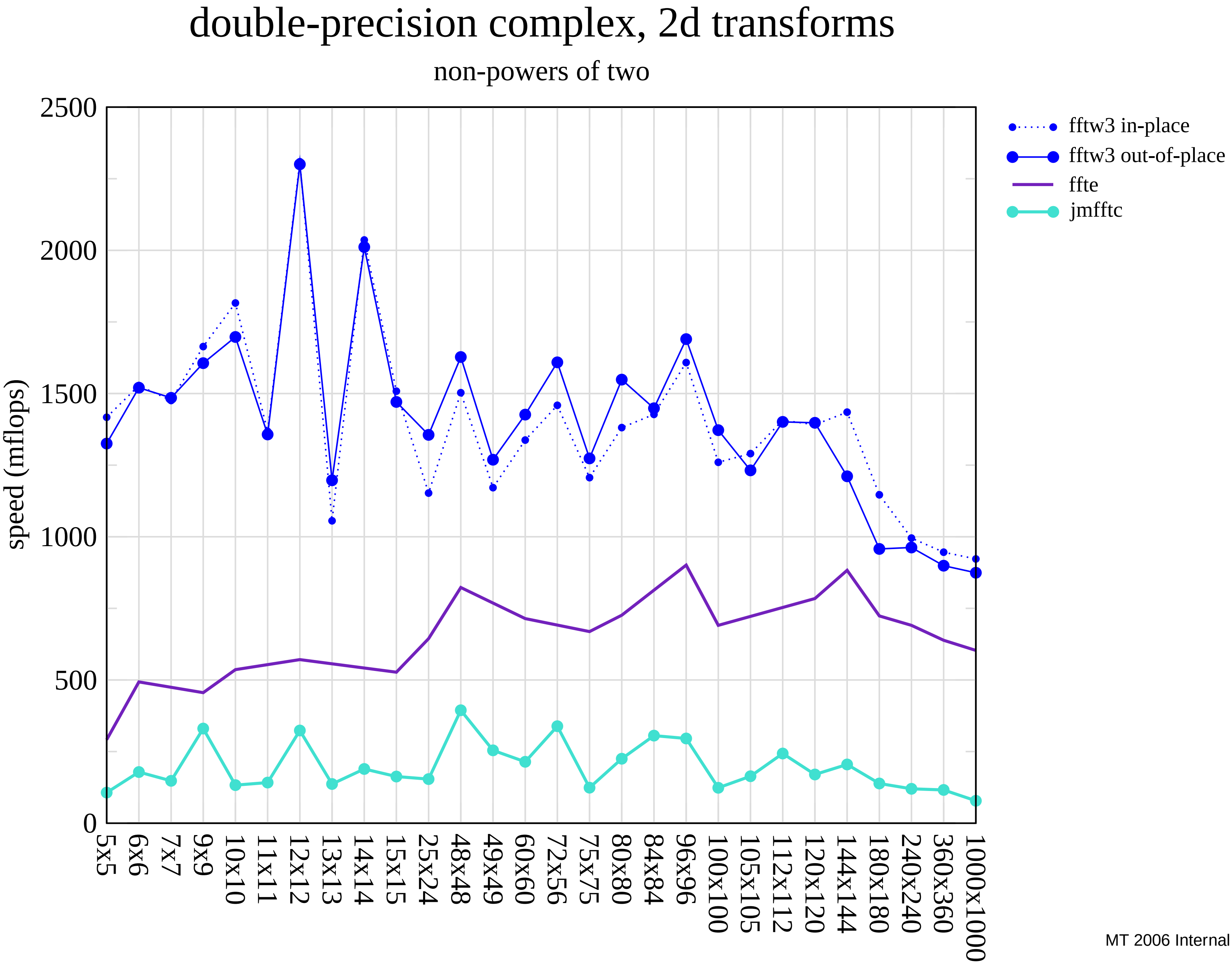
BenchFFT Results on Henrici1



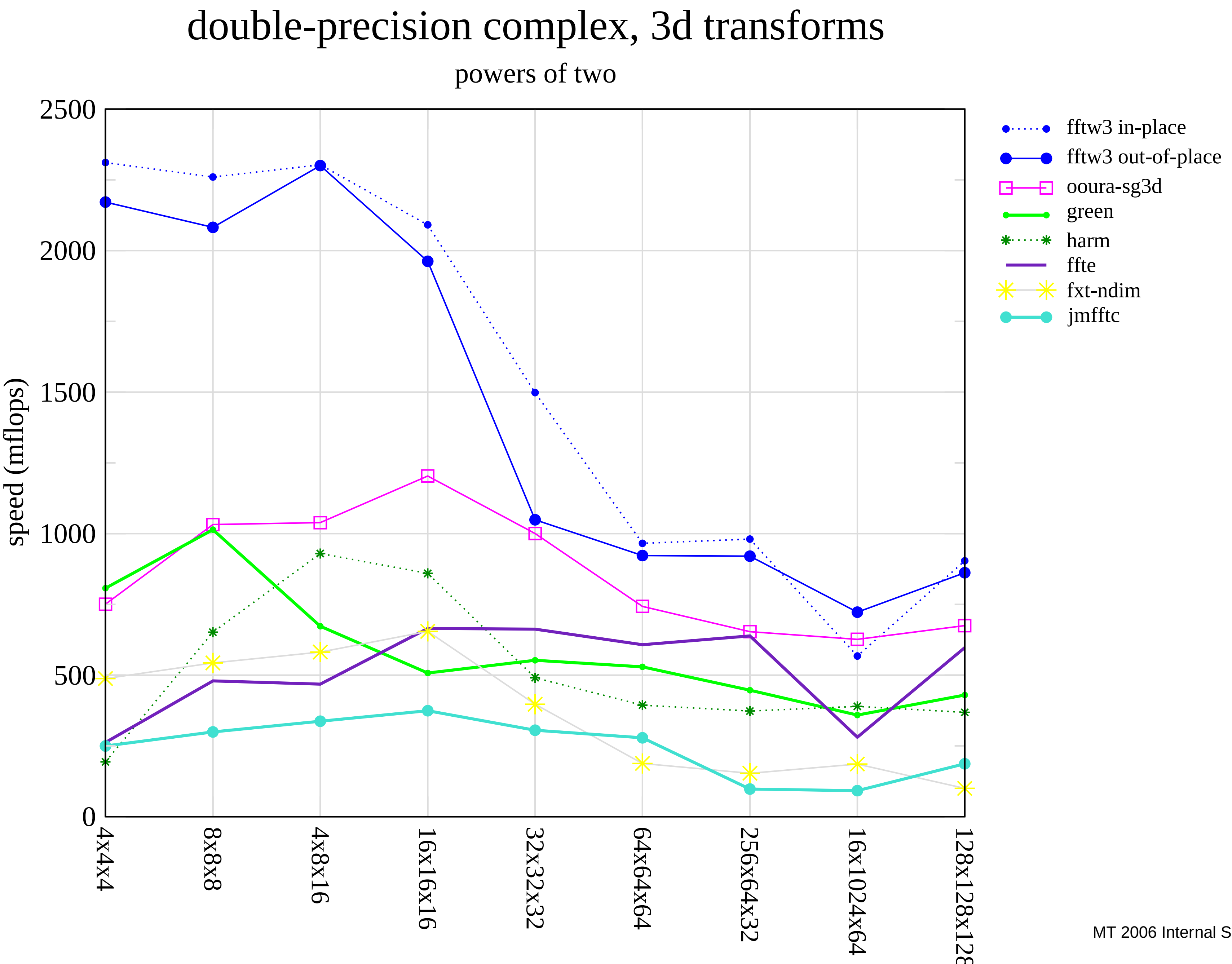
BenchFFT Results on Henrici1



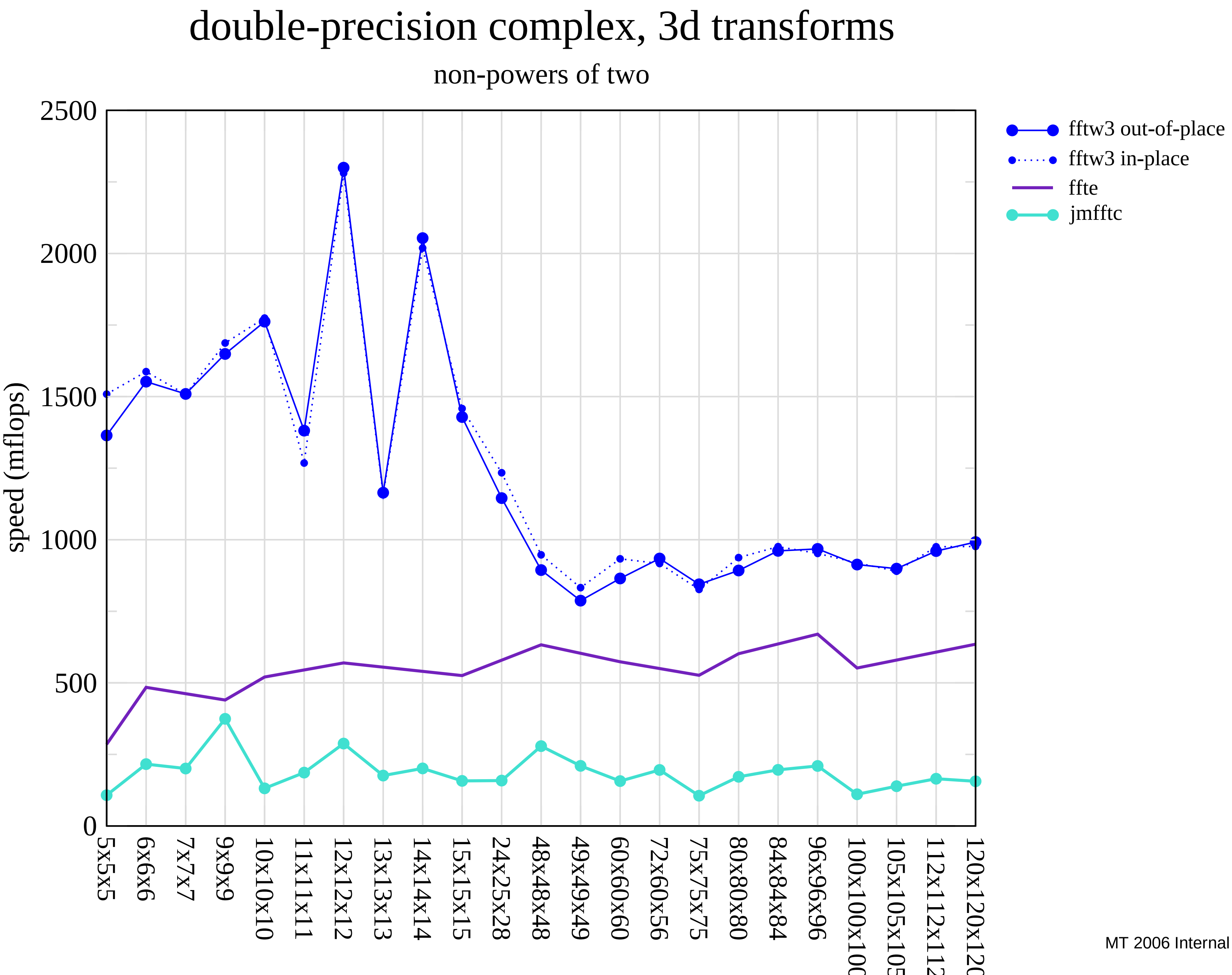
BenchFFT Results on Henrici1



BenchFFT Results on Henrici1



BenchFFT Results on Henrici1



Conclusions

Based on the experiments we conducted on Henrici1

- MKL DFT routines are faster than FFTW3.
- FFTW has competitive performance.
- NAG performed poorly.

We prefer FFTW for C/C++/Fortran because of its ease of use, portability, additional routines, documentation, and the many other features. This is the package that we would recommend.

In general, we find the Matlab routine to be very easy to use and recommend this if we are able to accept the performance overhead.